

Construction and Management of Large-Scale and Complex Virtual Manufacturing Environments

A thesis submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy

By

ZHIJIE XU BSc.

**School of Engineering
University of Derby**

September 2000

“If I have been seen further ..., it is by standing
upon the shoulders of Giants.”

Sir Issac Newton

ACKNOWLEDGEMENTS

I publicly express my thanks to the University of Derby for its support and to the School of Engineering for granting me the opportunity to carry out this project.

I would like to express sincere gratitude to Professor Zhengxu Zhao, my director of studies, who has guided the work and for giving me confidence in my research abilities when I needed it most.

My appreciation goes to Professor Ray W Baines who, as a second supervisor, provided his help, support and encouragement.

I thank the staff and my colleagues in the School of Engineering at the University of Derby for giving me help, ideas and a camaraderie which made my research so satisfying.

The motivation to pursue this degree is a result of the guidance and personality-shaping given by my parents, Jinfu Xu and Meiling Hu, during my upbringing. Thanks to them for molding and sharing my dreams and for supporting my ambition. Thanks to brothers, friends and relatives for their reliance and assistance.

I thank my wife, Yan Chen, who deserve a major share of this degree and its potential benefits. As my own personal fan, Yan is a loving wife who challenged me to do my best and provided comfort in times of doubt.

AUTHOR'S BIOGRAPHY

Zhijie Xu was born in Xi'an City, ShaanXi Province, P. R. China, on 13th May 1969. He married Yan Chen in 1995. He received his BSc degree in communication engineering in 1991 from Xi'an Mining Institute, Xi'an City, ShaanXi Province, P. R. China. From 1991 to 1995, he was an Electric Engineer in the State-run HuangHe Machinery and Electrical Co.Ltd in Xi'an City. In 1995 he worked on developing an industrial robot local area network communication and intelligent control system at the University of Derby as a Visiting Scholar. From 1996, he started his research on Construction and Management of Large-Scale and Complex Virtual Manufacturing Environments within the School of Engineering at Derby, where he completed this thesis. His research interests are Virtual Reality, Virtual Environment, Database Development, Robotics, and Computer Networks. He is now a lecturer in the School of Engineering at the University of Huddersfield.

RELATED PUBLICATIONS

Xu, Z. J., Zhao, Z. X., and Baines, R. W., 2000. Constructing Virtual Environments for Manufacturing Simulation. Accepted for including in the Special Edition of *the International Journal of Production Research*.

Xu, Z. J., Zhao, Z. X., and Baines, R. W., 2000. Application Oriented Configurable Virtual Manufacturing Environment Construction. *Proceedings of the 33rd International MATDOR Conference*, ISBN 1-85233-323-5, pp145-150.

Xu, Z. J., Zhao, Z. X., and Baines, R. W., 1999. A Virtual Environment for Manufacturing Simulation. *Proceedings of the Fifteenth Conference of the International Foundation for Production Research*, ISBN 1-874653-56-9.

Xu, Z. J., Zhao, Z. X., and Baines, R. W., 1999. Rapid Modelling Virtual Environment for Manufacturing Simulation. *Proceedings of the 15th National Conference on Manufacturing Research*, ISBN 1-86-058227-3, pp131-135.

Xu, Z. J. and Zhao, Z. X., 1999. Configurable Virtual Manufacturing Environment Modelling and Simulation. *Proceedings of the 1999 Chinese Automation Conference in the UK (CACUK'99)*, ISBN 0-9533890-2-2, pp71-76.

Xu, Z. J., Zhao, Z. X., and Baines, R. W., 1998. Variant Virtual Environment Construction and Manufacturing Information Representation Methodology. *Proceedings of the 14th National Conference on Manufacturing Research*, ISBN 1-86058-172-2, pp315-320.

Xu, Z. J., Zhao, Z. X., and Baines, R. W., 1998. Virtual Environment Construction and Applications. *Proceedings of the 1998 Chinese Automation Conference in the UK (CACUK'98)*, ISBN 0-953-38900-6, pp59-64.

Xu, Z. J., Zhao, Z. X., and Baines, R. W., 1997. Manufacturing Knowledge Acquisition and Database Management using Virtual Reality Techniques. *Proceedings of the 32nd International MATDOR Conference*, ISBN 0-333-71655-8, pp233-238.

Xu, Z. J., Zhao, Z. X., and Wu, M. H., 1997. Virtual Reality Based Robot Graphic Simulation and Virtual Manufacturing System. *Proceedings of the 1997 Chinese Automation Conference in the UK (CACUK'97)*, UMIST, Manchester, 13-14th, September 1997, pp195-200.

Wu, M. H., Xu, Z. J. and Baines, R. W., 1997. Integrate the LANCING Robot to PC Computer. *Proceedings of CIRP Symposium - Advanced Design and Manufacturing Era*, Hong Kong.

Wu, M. H., Xu, Z. J., 1996. Development of Robot Language Conversion System. *International Journal of INGENIUM*, Volume (1), p81-p86.

Wu, M. H., Xu, Z. J., 1996. Intelligent PCB Board Assembly System Based on PUMA Robot. *Proceedings of the 12th International conference of CAD/CAM Robotics and Factories of the Future*, Middlesex University, London, ISBN 1-89-825303-X.

ABSTRACT

The major challenges in designing and implementing an applicable virtual environment for industrial applications are to enhance the environment-based knowledge representation and its acquisition capacity and, paradoxically, the simplification of the environment construction, configuration and information management processes. This paradox has led to a search for an appropriate strategy for a practical environment construction method and related implementation platform.

This thesis describes such a new virtual environment construction approach – domain-analysis that is based on a top-down environment construction - for manufacturing applications. This approach reduces the effort to rapidly construct a virtual manufacturing environment using two steps: (i) application domain analysis, which classifies the application to identify the environment specification, and top-down construction that is based on a ready-built template as the starting point; (ii) The development of an integrated application development platform with various modules to enable a virtual environment and its virtual objects to be organised and managed in a database that can be connected with other data sources.

CONTENTS

ACKNOWLEDGEMENTS	I
AUTHOR'S BIOGRAPHY	III
RELATED PUBLICATIONS	IV
ABSTRACT	V
CONTENTS	VI
LIST OF FIGURES	XIII
LIST OF TABLES AND LISTS	XVII
CHAPTER 1 INTRODUCTION	1
1.1 THE VIRTUAL REALITY TECHNOLOGY	2
1.2 VIRTUAL REALITY SYSTEMS	4
1.2.1 Immersive VR systems	4
1.2.2 Augmented VR systems	5
1.2.3 Desktop VR system	6
1.3 VIRTUAL ENVIRONMENT DEVELOPMENT	6
1.3.1 Application programme interface (API) methods	7
1.3.2 Importing model methods	7
1.3.3 Graphical environment authoriser methods	8
1.3.4 Virtual reality modelling language method	8
1.4 VIRTUAL MANUFACTURING ENVIRONMENTS	9
1.4.1 Large-scale and complex environment	9
1.4.2 Virtual manufacturing environment knowledge	10
1.4.3 VE-based knowledge representation and acquisition	10
1.5 APPLICATION PROBLEMS	11
1.6 RESEARCH OBJECTIVES	13
1.7 THESIS STRUCTURE	14

CHAPTER 2 LITERATURE REVIEW	15
2.1 CONVENTIONAL MANUFACTURING SIMULATION	16
2.2 VR MANUFACTURING APPLICATIONS	17
2.2.1 VR based rapid prototyping	17
2.2.2 Process simulation and design validation	18
2.2.3 Assembly planning and test	19
2.2.4 NC programming and machining simulation	19
2.2.5 Factory layout design and cell control simulation	20
2.3 RELATED SYSTEMS	21
2.4 DISCUSSION	30
CHAPTER 3 AN OVERVIEW OF THE VE CONSTRUCTION APPROACHES	31
3.1 CONVENTIONAL APPROACHES TO CONSTRUCTING VIRTUAL ENVIRONMENTS	32
3.1.1 Bottom-up generative approach	32
3.1.2 Building-block approach	33
3.1.3 Variant construction approach	34
3.2 A DOMAIN-ANALYSIS BASED TOP-DOWN APPROACH	36
3.2.1 Application domain-analysis	36
3.2.2 Top-down VE construction	36
3.2.3 Operation mechanism	37
3.3 IMPLEMENTATION	39
3.3.1 Design application task coding scheme	39
3.3.2 Constructing template environments	41
3.3.3 Database Design	43
3.3.4 Linking VE properties and database records	44
3.3.5 Connect virtual and physical world	45
3.3.6 Integrate function modules under a unified system structure	45

3.4	CONCLUSION	45
CHAPTER 4 SYSTEM ARCHITECTURE		47
4.1	KAMVR SYSTEM ARCHITECTURE	48
4.2	INTERACTIVE APPLICATION INTERFACE	49
4.2.1	Visualiser interaction	50
4.2.2	VE Control module interaction	55
4.3	KAMVR SYSTEM MANAGER	58
4.3.1	Task description and coding module	58
4.3.2	Knowledge representation and acquisition module	63
4.3.3	Device communication and control	66
4.3.4	Network and data communication module	67
4.4	VIRTUAL ENVIRONMENT DATABASE	68
4.4.1	Database files	68
4.4.2	VE Database management system	69
4.5	REAL APPLICATION ENVIRONMENT	69
4.6	CONCLUSIONS	70
CHAPTER 5 TEMPLATE ENVIRONMENT CONSTRUCTION AND ANALYSIS		71
5.1	VIRTUAL ENVIRONMENT MODELLING	72
5.1.1	Virtual object modelling	72
5.1.2	Virtual template environment modelling	72
5.1.3	State simulation modelling	73
5.1.4	Interaction modelling	73
5.1.5	Knowledge capture	74
5.2	VIRTUAL OBJECT MODELLING	74
5.2.1	Virtual lathe model	75
5.2.2	Virtual milling machine model	77

5.2.3	Virtual robot model	78
5.3	MODELLING TEMPLATE ENVIRONMENTS	80
5.3.1	The modelling criteria	80
5.3.2	The construction of template environments	81
5.4	TEMPLATE ENVIRONMENT SIMULATION	85
5.5	INTERACTION WITH A VIRTUAL TEMPLATE ENVIRONMENT	86
5.5.1	Environment navigation	86
5.5.2	Environment exploration	89
5.5.3	Object control	89
5.6	KNOWLEDGE SOURCES AND CAPTURE	89
5.6.1	Data interpretation	90
5.6.2	Knowledge representation	90
5.7	CONCLUSION	91
 CHAPTER 6 DATA MANAGEMENT WITHIN VIRTUAL ENVIRONMENTS		 92
6.1	OVERVIEW OF THE ENVIRONMENT DATA	93
6.2	THE HIERARCHY OF ENVIRONMENT DATA	93
6.3	THE DEVELOPMENT OF AN ENVIRONMENT DATA STRUCTURE	100
6.3.1	The database of the environment	100
6.3.2	General template environment reference file	102
6.3.3	Virtual object file	102
6.3.4	Standard information	103
6.3.5	Dynamic information	104
6.3.6	Static information	105
6.3.7	Shape information	105
6.4	RECORDING TEMPLATE ENVIRONMENTS	106
6.4.1	Recording the data of a single object	106
6.4.2	Scan all objects in an environment	108

6.5	ENVIRONMENT CONSTRUCTION FACILITATED BY THE DATABASE	110
6.5.1	Constructing the scene graph of an environment	110
6.5.2	Retrieving a scene graph of template environments	111
6.5.3	Modify scene graph	112
6.5.4	Assigning object properties	114
6.6	CONJUGATING MANUFACTURING DATA AND VE	116
6.6.1	Static manufacturing data	117
6.6.2	Dynamic machining activities	118
6.7	CONCLUSIONS	118
 CHAPTER 7 ENVIRONMENT CONFIGURATION AND COMMUNICATION		 120
7.1	CONFIGURABLE VIRTUAL ENVIRONMENTS	121
7.2	ACCESSING ENVIRONMENT PROPERTIES	121
7.2.1	Configuring environment data structure	121
7.2.2	Configuring object shape properties	123
7.2.3	Configuring object static properties	125
7.2.4	Configuring dynamic properties	128
7.2.5	Configuring general environment properties	130
7.3	MIGRATING ENVIRONMENT PROPERTIES	132
7.3.1	Extracting and migrating shape properties	132
7.3.2	Extracting and migrating object static properties	133
7.3.3	Extracting and migrating object dynamic properties	134
7.3.4	Extracting and migrating global environment properties	134
7.4	SETTING AND UTILISING PROPERTIES	135
7.4.1	Setting the simulation triggers for an environment	135
7.4.2	Setting environment counters	136
7.4.3	Setting object properties	136
7.4.4	Receiving environment events	137

7.5	PLATFORM-BASED DATABASE ACCESS	138
7.5.1	Registering the environment database	139
7.5.2	Connecting the database	139
7.5.3	Binding environment properties with database records	140
7.6	INTERACTION BETWEEN VIRTUAL AND PHYSICAL ENVIRONMENTS	141
7.6.1	Communication with the Puma robot	141
7.6.2	Communication with the LANSING robot	143
7.7	CONCLUSIONS	146
CHAPTER 8 THE RUN-TIME IMPLEMENTATION		147
8.1	INTRODUCTION	148
8.2	KAMVR RUN-TIME PLATFORM	149
8.2.1	User-environment interaction controls	149
8.2.2	Control environment and database interactions	150
8.2.3	Standalone machine controls	152
8.2.4	Cell controller	154
8.3	WORKING MECHANISM	155
8.3.1	Monitoring and control an environment in active mode	156
8.3.2	Monitoring and controlling an environment in passive mode	156
8.4	RUNNING THE SYSTEM	158
8.4.1	Forming the task code	158
8.4.2	Environment modification and initialisation	160
8.4.3	Application configuration and simulation execution	163
CHAPTER 9 CONCLUSIONS AND FUTURE WORK		165
9.1	CONCLUSIONS	166
9.2	FUTURE RESEARCH	169

REFERENCES	172
GLOSSARY OF TERMS	180
APPENDIX A	A1
APPENDIX B	B1 - B15
APPENDIX C	C1 – C11
APPENDIX D	D1 – D53

LIST OF FIGURES

Figure 3.1	Illustration of the bottom-up approach	32
Figure 3.2	Building-block VE construction approach	33
Figure 3.3	Variant environment construction approach	34
Figure 3.4(a)	3D coding panel	40
Figure 3.4(b)	Task-based environment retrieval	40
Figure 4.1	KAMVR system architecture	48
Figure 4.2	VR devices used in the KAMVR system	
	(a) i – Glasses	50
	(b) Data Glove	50
	(c) Digitiser	50
Figure 4.3	Environment visualisers	
	(a) Superscape Visualiser	51
	(b) Superscape Viscap	51
	(c) Superscape 3D Control	51
	(d) COSMO VRML Player	51
Figure 4.4	Constructing virtual workshop using the World Editor	52
Figure 4.5	Modelled handwheel in the Shape Editor	53
Figure 4.6	Superscape Resource Editor	53
Figure 4.7	KAMVR system workbench	55
Figure 4.8	Data and command flows in the KAMVR Module 1	56
Figure 4.9	Synthetic VE control mode	58
Figure 4.10	Domain-analysis coding scheme	60
Figure 4.11	Module 2 internal structure	63
Figure 4.12	Adjusting the position of virtual objects	64
Figure 4.13	Superscape VRT API structure (<i>Courtesy of Superscape Co.Ltd.</i>)	65
Figure 4.14	Device communication	67

Figure 4.15	WWW On-line VME configuration	68
Figure 4.16(a)	Real robot cell (photo)	70
Figure 4.16(b)	Virtual robot cell	70
Figure 5.1	VE state transition model	73
Figure 5.2	Modelling structure of a virtual lathe	75
Figure 5.3	Snapshot of the virtual lathe	77
Figure 5.4	Modelling structure of a virtual milling machine	77
Figure 5.5	Virtual milling machine	78
Figure 5.6	Modelling structure of the virtual robot	79
Figure 5.7	A snapshot of virtual robot	80
Figure 5.8	Virtual template of robot cells	82
Figure 5.9	Virtual template of lathe cell	83
Figure 5.10	Virtual template of milling cell	83
Figure 5.11	Virtual template of CIMs rooms	84
Figure 5.12	Virtual template of large scale virtual manufacturing environment	85
Figure 5.13	The simulation loop imposed on the virtual lathe objects	87
Figure 5.14	The simulation loop imposed on the virtual miller objects	87
Figure 6.1(a)	Virtual environment scene graph	94
Figure 6.1(b)	A snapshot of the environment	94
Figure 6.2	KAMVR system database structure	101
Figure 6.3	Template environment reference file	102
Figure 6.4	Virtual object data file	103
Figure 6.5	Virtual object standard information file	104
Figure 6.6	Virtual object dynamic data file	104
Figure 6.7	Object static data file	105
Figure 6.8	Object shape data file	106
Figure 6.9	Environment standard information table	110
Figure 6.10	Template environment coding scheme	111
Figure 6.11	Environment constructed from a template VE	116
Figure 6.12	Manufacturing information composition	116

Figure 7.1	Virtual environment script header	122
Figure 7.2	Example of accessing shape property data	
	(a) Original object shape	125
	(b) Modified object shape	125
Figure 7.3	Accessing object static properties	125
Figure 7.4	Virtual object static data	128
Figure 7.5	Interface for setting object shape properties	132
Figure 7.6	Interface of object static property	133
Figure 7.7	Interface of object dynamic property	134
Figure 7.8	Environment property interface	135
Figure 7.9	Firing event from a virtual environment	
	(a) SCL firing event carrying two arguments	138
	(b) An application receiving the two arguments	138
Figure 7.10	Registering the environment database	139
Figure 7.11	Linking database and the VE application	
	(a) Referring a database in the application programme	140
	(b) Selecting data files in the database	140
Figure 7.12	Connecting PUMA robot	142
Figure 7.13	Lansing robot control panel	143
Figure 7.14	The keyboard control combinations	144
Figure 7.15	Virtual control panel signal processing board	144
Figure 7.16	Virtual keyboard ASCII value configuration	145
Figure 8.1	KAMVR run-time platform	148
Figure 8.2	Viewpoint control menu	150
Figure 8.3	The VE navigation bar	150
Figure 8.4	Updated general object information	151
Figure 8.5	3D machine controller	152
Figure 8.6	Virtual robot control dialog	153
Figure 8.7	System platform device controller	154
Figure 8.8	Cell controller operation sequence editor	153
Figure 8.9	KAMVR run-time platform task coding environment	159

Figure 8.10	Loaded environment	160
Figure 8.11	Modified virtual manufacturing environment	161
Figure 8.12(a)	VE Initialiser	162
Figure 8.12(b)	Initialised application environment	162
Figure 8.13	Application environment configuration	163
Figure 8.14	Simulation action – mounting different parts	164
Figure 8.15	Toggle the tool for part inspection	164

TABLES AND LISTS

Table 5.1	Features of the virtual template environments	81
Table 6.1	Object static and dynamic properties	115
Table 6.2	Machine tool knowledge	118
Table 7.1	Application platform utilities	136
List 6.1(a)	Environment definition in the scene graph script	95
List 6.1(b)	Object definitions	99
List 7.1	Virtual environment object data definition	122
List 7.2	Virtual environment data type	123
List 7.3	Object shape information	124
List 7.4	Object standard data section	126
List 7.5	Object dynamic data structure	129
List 7.6	Object rotational data properties definition	129
List 7.7	Collision data properties definition	129
List 7.8	Configurable environment properties	131
List 7.9	Data structure for accessing and configuring environment viewpoints	131

CHAPTER 1

INTRODUCTION

This introductory chapter provides a brief description of virtual reality and an overview of the thesis. It highlights the relevant areas about virtual reality including the technology itself, virtual reality systems, virtual environments and their applications. Following is an overview of the research problems dealt within the thesis and the research objectives.

1.1 THE VIRTUAL REALITY TECHNOLOGY

Virtual Reality (VR) is an emerging human-computer interface (HCI) technology. It provides computer generated 3D digital environments in which users can realistically interact with objects [Rheingold 1992 and Earnshaw *et al.* 1993]. The term “Virtual Reality” has various meanings. In some cases, VR is considered a specific collection of technologies, for example, a Head Mounted Display (HMD), Glove Input and Three Dimensional (3D) Audio. In other cases, the term is stretched to include conventional books, movies or pure fantasy and imagination. The United States National Science Foundation (NSF) taxonomy [NSF 1992] has covered these. However, the author for this research has adopted the definition of virtual reality given by Zhao [1997]:

“Virtual Reality is a computer mediated system that deals with a three dimensional digital environment that interacts with users in vision, sound, touch, smell and taste. The users should be able to manipulate, control and reconfigure this virtual environment and its complex data”

The three dimensional digital environment refers to computer generated visual, auditory or other sensual outputs to the users within the computer, it may be a CAD model, a scientific simulation, or a view into a database [Vince 1995].

Some people object to the term “Virtual Reality”, saying it is an oxymoron. Alternative terms that have been used are Synthetic Environments, Cyberspace, Artificial Reality, and Simulator Technology, to name a few.

VR applications have a wide spectrum, from games to architectural and business planning, from medicine to design and manufacturing, from training to testing and military use [Stampe *et al.* 1993]. One category of applications is mainly focussed on virtual environments that are similar to real ones, for example, 3D CAD or architecture modelling, factory layout and process simulation [Banerjee *et al.* 1992, Barnes 1996, Trika *et al.* 1997, and Gausemeier *et al.* 1998]. Another category of applications provides ways of viewing from an advantageous perspective not possible within the physical world, like scientific simulators, telepresence systems, air traffic

control systems, space programmes, hazardous environmental operations and Nano-engineering [Evans *et al.* 1994, Massie and Salisbury 1994, Wilson *et al.* 1996, Sun and Clapworthy 1996, Bennett 1997, and Zhao 1998]. Other applications are much different from anything that people have ever directly experienced before, for example, visualising the ebb and flow of the world's financial markets, navigating large corporate information, digitising behaviour of micro-organisms, and predicting engineering risk [Walczak 1996, Aouad *et al.* 1997, and Takalo *et al.* 1998].

Unlike technologies such as CAD, CAM and simulation, VR has manifested itself as being an enabling technology recognised by the academic community, industry, commerce and society as a whole, to have great potential in various, if not all, disciplines. Despite the high expectation, and sometimes the hyper-exaggeration (thanks to the media) about the state-of-the-art and the future of VR, the real implementation of VR technology in engineering, particular for design and manufacturing, is still in its incubation stage [Wilson 1996]. This is because VR suffers from problems that come from the end users. In broad terms, those problems can be viewed as three questions that require answers. (i) How can real-world data, information, knowledge and experience be captured and usefully represented and managed in VR form? (ii) How can virtual environments be constructed by the user and not programmers and system developers? (iii) How can engineering disciplines and situations be transposed from and to virtual environments?

The research presented in this work is to investigate solutions to the above problems. Its aim is to provide an efficient virtual environment construction approach and VR based application framework to facilitate manufacturing knowledge acquisition, representation and management. The knowledge includes both simulation data and information and intuitive engineering experience that are difficult to deal with using conventional modelling and simulation techniques. The research has been conducted from the perspective of VR applications rather than VR theory and related computing techniques, but, to enable readers to follow the context of this work, some basics of virtual reality and VR systems have been provided in the following sections.

1.2 VIRTUAL REALITY SYSTEMS

Generally, VR systems can be classified as: (i) immersive VR systems, (ii) desktop VR systems, and (iii) augmented VR systems.

1.2.1 Immersive VR systems

Immersive VR is where the users are physically isolated from the real world [Mckenna and Zeltzer 1992, Boman 1995, Roussos *et al.* 1999]. Immersion is achieved by providing a spatial relationship between the users and the environment through location and orientation tracking devices [Sowizral *et al.* 1993, Adam *et al.* 1995]. For example, a head tracking system allows the computer to generate correct images for the user. The images are then sent to special display devices, such as, i-glasses or CAVE systems [Neira *et al.* 1992] to real-time update and render the virtual scene according to the tracked head movement to provide user with the feeling of “being there”. Immersive VR devices can be classified into four types:

(1) Position and orientation trackers: These can be further divided into sourced or sourceless device [Meyer 1992]. Sourced trackers uses spatial relations between a pre-defined base and sensor to determine the sensor’s position and orientation relative to a base. For example, an electromagnetic (EM) tracker uses a transmitter and receiver to monitor users’ movement and an ultrasonic tracker uses speaker and microphone to record the spatial changes. Sourceless trackers do not need a pre-defined base but use a ‘global’ reference such as the gravity or magnetic field of the earth. Current commercial sourceless trackers are mainly used for measuring orientation, but not position due to inaccuracies [Bricken 1994].

(2) Input devices: VR input devices are used to import customised data to update a virtual environment [Jones 1999]. Typical VR input equipment includes data gloves, data suits and space mouse. When used as an input device in VR applications, a data glove controls a virtual hand within the synthetic environment. The user then interact with the environment as if they were using natural hand gestures and spatial movements, for instance, to select a virtual object by reaching out and grasping it.

(3) Display devices: VR display systems, depend on their relationship to the user, can be classified as:

- Wall projection display: These devices use either a large computer screen or rear projection display. Examples include ImmersaDesk and CAVE [Neira *et al.* 1992]. In the latter system, stereo-glasses are used for depth perception.
- Helmet mounted display (HMD): HMDs consist of a pair of Liquid Crystal Display (LCD) or Cathode Ray Tube (CRT) display devices mounted on a helmet worn by the user.
- Mechanical arm mounted display: These are similar to helmet-mounted displays except that the display device is mounted on a mechanical arm that acts as both a tracker and a support. Accuracy can be very high but freedom of movement is restricted.
- Desktop display: These devices allow mono-scopic or stereo-scopic viewing on computer screens. They can provide a “see-through-window” visualisation sense, but the feeling of immersion is minimal.

(4) Haptic feedback devices: These provide a user with physical feeling such as touch, warmth, coldness, smoothness and resistance [McNeely 1995]. These devices can be classified as touch (tactile) feedback devices and force feedback devices. Touch feedback provides synthetic sensation at the moment of contact with a virtual object. Unlike the surface detail provided by touch feedback, force feedback gives a much larger impression of the physics of the virtual world. Force feedback devices are designed to imitate the forces that could be applied in the virtual world. The imitated forces can be anything from the elastic resistance (when squeezing a rubber ball) to large forces (that prevent the user’s hand from penetrating a wall).

1.2.2 *Augmented VR systems*

Instead of immersing the user completely in a virtual environment, augmented reality systems combine computer-generated imagery with a view of the real world [Boman 1995]. This is achieved with a spatially tracked, partially transparent head-mounted-device (HMD). An augmented VR system can be used to overlay information on real-

world objects, such as showing the location of a component on the inside of a machine. Boeing Computer Services is developing such a system for use in manufacturing, assembly, and repair work to replace the large form boards previously used for these tasks [Mizel and David 1994].

1.2.3 Desktop VR systems

Immersion and augment VR are generally associated with various novel interface and display techniques, they are not necessarily a pre-requisite for virtual reality. A reasonably high level of involvement or feeling of immersion within a virtual environment can be achieved using standard interfaces and displays [Young 1996]. Systems that work in this style are called desktop VR systems. For example, Superscape VRT is a desktop VR system [VRT 5.0 Manual].

1.3 VIRTUAL ENVIRONMENT DEVELOPMENT

To develop a virtual reality system, the modelling of a realistic virtual environment is one of the most difficult and demanding tasks [Codella *et al.* 1993]. Despite system functions claimed by various vendors, the utilities for constructing VR environments are still left for users because commercial VR systems need to remain as general as possible to accommodate different users. The construction of VR environments is still application specific and consequently involves time consuming processes. Different types of virtual environment development tools are available on the market. From the users' point of view they can be classified as desktop or immersive, and either can be stand-alone or networked (distributed) or both. According to the modelling methods, those tools fall into three categories. One is the modelling toolkits that allow users to construct virtual environments and then to use them from within the VR systems or to export the file to other formats. Typical examples of this type are Superscape VRT and Division dVS [Vince 1995]. The second category is the editorial software that uses an independent description languages, such as Virtual Reality Modelling Language (VRML) (an ISO standard file format) that describes virtual worlds specially used on the Internet [3D Web Consortium Incorporated 1997]. The third category uses a VR Application Programmer Interface (API) which typically provides a large number of facilities or programming functions to write 3D graphics

applications, for example, WorldToolkit [Sense8 Corporation 1992]. Some of these systems can import virtual objects from other graphical software packages such as AutoCAD, 3D Studio and ProEngineer.

Despite the fact that a great variety of virtual environment modelling tools are commercially available, an efficient virtual environment constructing method is still to be seen. Currently, virtual environment, especially large scale and complex ones are created based on the following methods.

1.3.1 Application programme interface (API) methods

This essentially relies on a built-in VR system programming language library or interface functions that enables advanced users to write system code using, say, C/C++ or Java languages. The code developed by the users is normally compiled into an executable program that generates a dedicated virtual environment. Examples of VR systems that provide such a construction method are WorldToolKit, MR toolkit [Shaw *et al.* 1993] and Reality-Built-for-Two [Blanchard *et al.* 1990].

These methods provide flexibility for environment developers to create a large and complex virtual environment. However, they demand high-level programming skills and the generated environments provide no flexibility for users to change and reconfigure, when different applications requirements are involved.

1.3.2 Importing model methods

This method makes use of the modelling functions of other graphical packages to create individual virtual objects or sub-models and then imports them into the data buffer of a VR system. Some VR systems provide such methods to re-use existing virtual models created elsewhere. These can significantly reduce the effect of constructing a virtual environment from scratch.

Examples of this method can be found in the world editing process of Superscape VRT, during which AutoCAD 3D models created in 'data exchange format' (DXF) can be migrated into a virtual environment under construction. An advantage of this

method is that it allows input of other data forms such as sound data, video clips, photo images and other media forms. For example, the Sense8 WUP system can import a sound wave file and video clips from 3D Studio. It can also import VRML based 3D models from Pro/Engineer [WorldUp User Guide 1997].

However there are technical difficulties with this method, which seem to be formidable to overcome as follows.

- (i) Data converters are necessary and must be made available for the importing. Technically, each data format requires its own converter to import the expected models from other systems to a VR system. However, generally, in a VR system, not all converters are made available.
- (ii) Most imported data, unless in the VRML format, are not manageable once it transfer into a VR system. The data imported to the VR system is often not transparent and users can not control its 3D attributes.

1.3.3 Graphical environment authoriser methods

Currently, the most common way to build a virtual environment is using an environment authoriser, which is a graphical user interface that works in a drag-and-drop manner similar to an ordinary CAD system.

This method relies on users to create a virtual environment with simple geometric primitives such as points, lines, facets and volumes. Obviously it is time-consuming and demands but for a great deal of CAD modelling skills. This method is reasonably effective for constructing simple and regular geometric shapes, but for complex virtual environment, it is a difficult method.

1.3.4 Virtual reality modelling language method

The Virtual Reality Modelling Language (VRML) is a script language to describe VEs and an ISO standard (International Standard ISO/IEC 14772) for specifying 3D virtual worlds networked via the Internet. The VRML environment on the World Wide Web (WWW) can link to or be linked from other sources on the Internet, but

the use of VRML for constructing virtual environments is very much similar to the API method (See Section 1.3.1).

1.4 VIRTUAL MANUFACTURING ENVIRONMENTS

VR manufacturing applications usually require one or more virtual manufacturing environments (referred in the following text as VME) with meaningful (in both graphical and simulation terms) constituent objects. The virtual objects can be constrained to behave in a similar way to that of physical objects. A successful implementation of VME is more informative and realistic than conventional 2D simulation programs, particularly for acquiring cognitive information, representing scenario knowledge, understanding sophisticated problems and eliminating risks of complex processes [Charitos and Rutherford 1996]. It is justified by what can be perceived and what can be done with it. The former is determined by its scale and complexity, and the latter by the environment knowledge [Macedonia *et al.* 1995].

1.4.1 Large-scale and complex environment

According to the Advanced Interface Group (AIG) in the University of Manchester [Cook *et al.* 1998], a Large Scale and Complex Virtual Environment can be defined by the following:

- Complexity of graphical display: the complexities of a virtual environment are defined by the detail and realistic level of virtual objects that it contains (it is understood even the illusion of complexity can be achieved with "wall-papering" techniques such as texture maps, this is not adequate if the VE is to be composed of real objects offering the possibility of interaction).
- Number of objects in the environment: The real world is characterised by large numbers of objects with many possibilities for interaction among them. For instance, a complete manufacturing cell may include hundreds of objects, each composed by various number of polygons.
- Complicated behaviour: The behaviour of objects in VEs are described in terms of geometric transformations, and are often defined on a per object basis [Cook *et*

al. 1998]. Another common mechanism is to provide simulation code (e.g. SCL from Superscape or Java) to individual objects. For a manufacturing application this often does not work well because too many objects, activities and interactions are involved.

- Number of users and their locations: A large-scale environment can be distributed and shared through a computer network. The communication mode and data interpretation will also contribute to its complexity.

1.4.2 *Virtual manufacturing environment knowledge*

Virtual manufacturing environment knowledge includes the information concerning environment geometric information – for example, object size and position; environment dynamic information - such as machine tool simulation and virtual assembly data; manufacturing information – including manufacturing rules and regulations, and the relationships among them. The early presentation type of application projects is being replaced by more ‘real’ applications, which are used on a day-to-day base and aim to solve real problems. For example, a German company has developed a manufacturing knowledge-oriented virtual environment for BMW to evaluate large press tools, which decreases the average evaluation time substantially [Bullinger and Roessler 1998].

1.4.3 *VE-based knowledge representation and acquisition*

Many current implementations suffer problems mainly because manufacturing data and information are defined from different knowledge sources, such as databases, spreadsheets and drawings, that has to be collected and formally represented in the application. Furthermore, a large proportion of manufacturing data and information is based on experience, which is empirical and non-generic. The interpretation of the majority of manufacturing knowledge entirely depends on the cognitive understanding of humans. Systems such as modelling, planning, simulation and artificial intelligence (AI) expert systems have employed rather formal and simplified models to represent such knowledge in the form of formula, text, logic and rules. In most cases, they have failed to solve the difficult problem of using empirical knowledge and still have to rely on human intervention.

In contrast, the theme of VR and VE is not only to bring about automatic computerised system functions to engineering applications, but also importantly to incorporate human abilities into those computerised systems in a way that human cognisance and senses to the real world can be utilised in a virtual environment, where non-generic knowledge can be dealt with using formal computer programs.

1.5 APPLICATION PROBLEMS

In spite of the great expectation of applying VR to factory layout, manufacturing process planning, operation training, system testing and control validation. There are few cases of practical industrial use of VR in manufacturing that have been reported in the literature – apart from those described as prototypes and on-going projects. Until recently, it is becoming more obvious that this is not because of the lack of understanding of VR technology by manufacturing industry, rather it is due to the difficulties in construction of virtual manufacturing environments and the lack of effective methods and techniques to do so. Those difficulties can be described as follows.

(1) A virtual manufacturing environment and its contents (virtual machines, tools, and systems) must be abstracted from their physical forms and existence to low-level hyper (or digital) details such as VR codes, 3D polygon rendering geometry, animation elements and other virtual environment sensory factors. Those abstracted low-level VR elements must have manufacturing semantic meanings so that a virtual manufacturing environment can be constructed with not only visual resemblance, but also functional similarity, to its physical counterpart. For example, a virtual model must be constructed to have virtual dynamic control mechanisms that are able to follow programmed instructions and perform manufacturing tasks in a virtual environment. Abstraction of manufacturing reality into a VR model is not a generic problem due to the diversity of manufacturing systems and application requirements. So far little research method towards this problem has been reported in the literature.

(2) Most virtual manufacturing environments require large-scale and complex VR models. Applying currently available construction methods to create even a reasonably small-scale VR model, for instance, a machine cell, is a tedious and time-

consuming and expensive process [Zhao 1997]. Considerable research has been done in this area. Recently a few well-known industrial VR system software houses have shifted their efforts from developing shell VR systems to application-specific virtual environments, but a satisfactory environment building method is still not available.

(3) A unique difficulty with virtual manufacturing environments is that they need to be modelled in a higher degree of engineering accuracy than that of, say, virtual environments for civil engineering and entertainment applications.

Precision in VR modelling is mainly constrained by the internal manufacturing data rather than by the VR rendering or graphical visualisation effect. For example, an artistically and realistically rendered virtual CNC machining cell may not be useful if it is unable to detect collisions to avoid tool-breaking or wrongly access a machined part feature.

(4) Another difficulty with virtual manufacturing environments is the formalisation or representation of manufacturing knowledge in VR terms. A large proportion of manufacturing data and information in a VR model is empirical and non-generic. Although one of the strengths of a VR model is that it can deal with such knowledge without a knowledge base, the data need to be strictly monitored and managed within the model.

Incorporating a graphically correct (or even crude) VR model with complex, diversity but well managed manufacturing database is a challenging problem [Zhao 1997].

(5) As a universal requirement for all virtual environments, user-based reconfigurability is also a necessity of most virtual manufacturing environments. At present, VR models are mostly created by VR specialists or programmers. The kind of model structure and non-transparent modelling data left users little flexibility to make any change on the model.

The research described in this thesis attempts to reduce these difficulties, with a specific emphasis on, (i) establishing effective methods for rapid creation of large and complex virtual manufacturing environments; (ii) providing techniques for acquiring

manufacturing knowledge and managing manufacturing data within a virtual manufacturing environment. In the first part, it is hoped that the methods could lead to reconfigurable models. The research motivation of the second part is to make non-generic and empirical manufacturing knowledge accessible by future VR based manufacturing systems such as VR process planning, system modelling, factory layout, operation testing and control. A more detailed specification of the research objectives is provided in Section 1.6.

1.6 RESEARCH OBJECTIVES

The research objectives can be summarised as follows:

- (1) To establish a concept by which a virtual environment can be constructed rapidly by users as well as VR specialists. This is to reduce the demand for users to learn specialised virtual environments authorising toolkits and VR systems.
- (2) To investigate virtual environment data structure and its modelling relationships with manufacturing knowledge by constructing and analysing a series of template manufacturing environments.
- (3) To investigate a method of defining and formalising manufacturing knowledge in VR terms. This is to be used to identify the data link between environment contents (and contexts) and manufacturing data, for example, machining parameters in a milling operation, thus enabling environment design and manufacturing information management to be unified into a single constructing process.
- (4) To develop a database system that monitors and manages environment data as well as manufacturing information both during and after the environment is being built. The template environments created in (2) are used to help in populating this database with specific manufacturing data.
- (5) To establish an environment reconfiguration mechanism. This is to expose environment and object properties to users for migrating VR models with run-time control.

(6) To develop a prototype system which accommodates the essential programming modules resulted from the above objectives. It is used as a run-time platform to test the entire process of virtual manufacturing environment construction, reconfiguration and use.

(7) To explore the virtual and real world communication by linking a virtual model with its physical counterpart.

1.7 THESIS STRUCTURE

The work reported in this thesis starts from a general description of virtual reality technology, systems and manufacturing applications in Chapter 1. The difficulties on virtual environment construction in terms of geometric and simulation modelling, manufacturing knowledge acquisition and representation are also briefly discussed in this chapter. The research objectives of this project are then discussed. Chapter 2 presents a literature review of virtual reality applications in a manufacturing life-cycle and VME construction systems. A novel approach for constructing VME based on the classification and analysis of current conventional approaches is introduced in Chapter 3. A modular system which uses the new approach and support environment and application data management is described in Chapter 4. The construction of a series of so-called template manufacturing environments for populating the system is detailed in Chapter 5. Based on the template environment construction processes, Chapter 6 introduces the data structure of virtual environments, and a database that stores and manages the information. The database enables the migration of environment and objects data with simulation control instructions, which allows a VME to be modified and reconfigured for different applications. Chapter 7 describes the programming techniques used for the purpose. The integrated computing platform of the addressed system and an example to illustrate its working procedures are presented in Chapter 8. The work is concluded in Chapter 9 with the recommendation for future research.

CHAPTER 2

LITERATURE REVIEW

This chapter provides an overview of the key virtual reality and virtual environment manufacturing applications, and existing approaches to constructing virtual manufacturing environments that influenced the research.

2.1 CONVENTIONAL MANUFACTURING SIMULATION

Modern manufacturing systems are capital-intensive due to their hardware and software requirements [Viswanadham and Narahari 1992]. Simulating those systems to gain a profound understanding of their complexities and to predict their performance is critical in both system design and implementation, and is often valuable for system management during their use [Narayanan 1997].

Conventional manufacturing simulation aims to facilitate this process by simulating real-time systems that have a large number of interrelated processes and events on a computer-based platform. These simulated activities occur sequentially and/or concurrently under stringent time constraints, and require modelling the system behaviour accurately to assure that they satisfy the application requirements.

Although conventional manufacturing simulation has been widely considered one of the most useful tools for analysing and designing complex manufacturing systems [Ozdemirel *et al.* 1993], it has two fundamental problems.

First, simulation modellers often encounter difficulties in transforming the real world multi-dimensional, visual and dynamic characteristics into the one-dimensional, textual and static representation required by traditional simulation languages (which are usually represented by tables, graphics and statistics on a computer screen) [Ülgen and Thomasma 1990, Adiga and Glassey 1991]. The result of such presentation produced from the conventional simulation process is often hard to understand, manipulate and utilise [Jones, 1993].

The second problem is due to the integration of design and manufacturing data [Barnes 1996, Bejczy 1997, Gray 1997] for visualising large system models [Hirota and Hirose 1995 and Kerttula *et al.* 1997]. For conventional simulation systems to define manufacturing-related problems, visualise manufacturing facilities and layout, evaluate environmental and ergonomic issues, for instance, are still difficult tasks [Angster *et al.* 1994 and Vance 1996].

One challenge in constructing a virtual environment for manufacturing simulation is the representation of environment-based manufacturing knowledge and, paradoxically, the simplification of data modelling, VE configuration and information management processes.

Due to the diversity and complexity of virtual manufacturing environments, especially for manufacturing simulation problems, the construction of those VEs needs to be straightforward and involve as little computer programming as possible. Considering the state-of-the-art of current VR systems and the modelling demands of most manufacturing simulation tasks, an enabling system is a necessity to facilitate manufacturing users, not computing programmers, to construct easily and reconfigure rapidly large and complex VEs for a specific simulation task. This observation has been made based on the extensive literature study on VR and its application in engineering in general and manufacturing in particular. The following section in this chapter documents these studies.

2.2 VR MANUFACTURING APPLICATIONS

To appreciate why and how VR and VE can be applied in various manufacturing areas, this section describes a few typical cases that show the use of virtual reality in manufacturing.

2.2.1 *VR based rapid prototyping*

To achieve accurate manufacturing processes and high product quality, industry is applying various technologies and concepts for dealing with manufacturing problems in the beginning of the product design stage. Concurrent engineering design for manufacturing and integrated manufacturing are the most commonly applied practices [Maxfield *et al.* 1995, Angster 1996, Barrus 1994]. A more recently developed technology compliant with these technologies is rapid prototyping.

The Rapid Prototyping technique produces parts from a CAD modelling database using some form of polymer or plastic material, these physical prototypes are often expensive, especially for large components [Rosen *et al.* 1995 and Xiao *et al.* 1997].

The ultimate development of rapid prototyping in terms of both speed, cost and flexibility is virtual prototyping – the use of a 3D VR model of the product to explore the proper manufacturing processes and to evaluate ergonomic performance of the product prior to its physical construction [Chapman and Coddington 1994, Bennett 1997].

Kerttula [1997] and other researchers in the University of Oulu at Finland developed a virtual prototyping environment for developing electronics and telecommunication products. Immersive VR devices including PHANTOM Haptic Device [Massie and Salisbury 1994], Logitech 3D Head Tracker, body tracking device, stereoscopic system and data glove were used in the project. The creation and building of a virtual prototype using their system starts from defining the geometry of the virtual prototype by converting data from a computer-aided-design (CAD) model, or by using modelling tools to construct from scratch. The geometric model is represented using OpenInventor graphic format. The virtual environment produces a visualisation model by adding surface properties (colours, materials and textures). Then it generates the haptic rendering model (force-feedback model) from the geometric model by assigning the tactile and force feedback simulation parameters to each virtual plane in the geometric model. The constructed geometric model and haptic model are used to create the final simulation model, which by integrating with the VR hardware and software form a comprehensive VE-based virtual prototyping system.

2.2.2 Process simulation and design validation

Very often, a constructed virtual prototype needs to be evaluated against real manufacturing methods and processes to ensure its validity [Hollands and Mort 1994, Jones and Iuliano 1997, Lee and Noh 1997]. Depending on the data format of the virtual prototype, it is possible for a manufacturing operation to be driven directly by the data from the virtual prototype. When the design is updated, the changes can be implemented automatically in the manufacturing processes (such as machining and inspection), that can avoid expensive redesign and physical testing.

Researchers in Washington State University developed a system that allows a product design to be tested on a series of virtual machines. These machines are modelled

according to their physical counterparts and are fully functional as real machines. A product model was created using Pro/Engineer and was imported through an internal data translator of the system [Angster *et al.* 1994]. Similar projects were reported by the University of Illinois and the Purdue University in the USA, the focus of those VR based projects is to simplify the design of complex mechanical parts and to achieve a one-off manufacturing [Trika *et al.* 1997].

2.2.3 *Assembly planning and test*

Assembly planning is another application of VR in manufacturing. Grasping and moving virtual parts in a VE and placing them in the right position provides significant insight into the assembly process [Ritchie *et al.* 1999]. VEs for virtual assembly rely on a full kinematics simulation of a proposed assembly sequence to evaluate potential insertion paths and to avoid clashes within an assembly path [Connacher *et al.* 1995].

Banerjee [1999] at the University of Illinois used VR techniques in conjunction with assembly constraints generated by heuristic rules to test parts features in the assembly. The reasons for using VR in his project were: (1) the assembly processes are highly visual, (2) a majority of assembly operations in factories are performed manually due to difficulties in automation, therefore, demanding much involvement, and (3) there are a number of assembly operations which require dextrous operation training. Hence, VE becomes an ideal candidate for this highly visual and interactive task, especially when operator training is becoming important [Banerjee *et al.* 1999].

2.2.4 *NC programming and machining simulation*

Another application of VR in manufacturing is to simulate the Computer-Numerical-Control (CNC) machine operations to verify machining or fixtures prior to the actual machining. Current CAD/CAM systems provide tools for automatic Numerical-Control (NC) code generation. Those systems allow viewing of the tool path for verification of NC code. Often, it is not the cutting tool itself but the chuck, head stock, and other parts of the machine tool that could cause problems, for instance, a tool-holder ramming the workpiece could cause several thousands of dollars of

damage [Matsuda and Kimura 1997]. In VR-based machining simulation, this type of problem can be alleviated.

Iuliano and Jones [1996] in the National Institute of Standards and Technology (NIST) and Watanabe [1997] at Mitsubishi Electric Corporation proposed their individual VR based NC programming and simulation systems. The former focused on integrating different manufacturing systems into a VR computing framework to simulate the complete loop of virtual manufacturing, where a part can be designed, validated, machined, and assembled in a single environment. The latter was dedicated to verify the virtual machining NC code that can be exported to the physical equipment.

2.2.5 *Factory layout design and cell control simulation*

VR has been applied to factory and cell layout and automated cell control simulation. Validation of the layout of a manufacturing facility through a VE allows the engineer to visualise the facility dynamically, and modifying the design by real-time interacting with the virtual objects for economic, ergonomic and safety reasons [Zetu et. al 1997]. Smith and Heim [1999] have specifically pointed out that for manufacturing environments where the third dimension is critical to system performance, interactive three-dimensional display systems are much better to convey the information by facility designers.

Gausemeier (*et al.*) [1998] from Heinz Nixdorf Institute in Germany has developed a VE-based manufacturing cell layout system using the so-called “Construction Set”. It was applied to the development of a modular monorail transport system. The essential components of the Construction Set include 3D models of the fundamental construction elements such as building blocks, which are created using a conventional 3D CAD system. For modelling the behaviours of the system, a commercial VE authoriser, SmartScene, was used [Multigen Inc., USA.]. A manufacturing cell layout planning system has been developed by Korves and Loftus [1999] in the University of Birmingham. They applied a similar building block-based approach using a pre-built shop-floor model library. At system run-time, users select an equipment model by using an immersive menu and then drag-and-drop into the virtual environment.

In the aspect of cell control simulation, Orady and Osman [1997] in the University of Michigan developed software for manufacturing cell control simulation by integrating discrete event simulation techniques with virtual reality software, where the simulator acts as the cell-activity administrator and the environment as the interaction front-end.

2.3 RELATED SYSTEMS

The VR and VME developing systems reviewed in this section cover the research-based VR hardware and software, and systems useful for developing manufacturing application environments.

Researchers at the National University of Singapore followed an interactive and visual approach to virtual world construction, which allows virtual world designers to work with high-level environment design concepts [Singh *et al.* 1993, 1994 and 1995]. The system was developed in two stages: Bricks and BrickNet.

- Bricks emphasises on modelling application knowledge to avoid only providing a wide range of low-level primitive geometry. It used a self-developed frame-based representation language [Loo 1991], and C for the modelling processes, the application knowledge were possessed by the virtual environments in the form of representative virtual objects and stored object activities.
- The BrickNet toolkit extends the sharing of objects on the computer network to include dynamic object behaviour. This is achieved by combining a structured organisational paradigm for virtual worlds with an interpreting language. Sharing in virtual worlds is handled by transferring the program code that builds the structure and executes the behaviour. The range of behaviours shared in BrickNet include simple behaviour, virtual world dependent behaviour, reactive behaviour and capability-based behaviour. Those object activities can be used to trigger each other to form an action network.

The above research provided a method of constructing a virtual environment following a logical concept. For example, to create a virtual kitchen, the process starts

from sorting the relationships among kitchen objects and the possible ways a human would interact with those objects. The related operational knowledge was created as task procedures and stored with the referring objects. This approach overcomes the shortcomings of “programming” an environment (which is normally a job for expert programmers rather than application developers). The problem with this method, however, is that the environment construction still requires the user to have considerable object modelling knowledge, and the expected environment activities still have to rely on system developers to code in which having limited run-time reconfiguring capacity.

Environment Manager (EM) for VE construction and application data management was developed by Wang and Green [1995] in the University of Alberta as a toolkit for constructing single or multi-user VEs. A script file was developed for handling the activities of VEs and objects, for example, to initialise or run the VE simulations. Individual applications share information and cooperate with each other across the Internet. EM reduces the effort required to produce a networked virtual world by providing high-level support for application replication, network configuration, communication management and concurrency control. It's composite modules included:

- MR Toolkit: for virtual environments and objects geometric modelling. MR applications are written in C and Fortran. The graphics programming adopted a computer graphics library including GL, Phigs and Starbase.
- JACAD: a solid modelling and animation computer aided design system. It has a key frame animation facility that is used to animate various motions of an object without writing an OML object.
- OML: Object Modelling Language is a procedural programming language. It is used to generate 3D objects, control object appearance, and behaviour. OML behaviour is a procedure (method) that reacts to an incoming event or combination of events, and typically generates some sort of change in the state and appearance of the 3D object. Behaviour triggers other behaviour by sending an event to the behaviour to be triggered.

When using such a system, an application environment is created using OML. A MR Toolkit program loads the compiled OML code for displaying each of the objects on the computer display. The VR input and output (IO) devices are then setup, the OML dispatches the device-related events to an event interpreter and calls the interpreter every frame to update the graphical and other IO display.

Similar with the aforementioned “Brick and Bricknet” system, this system also provided the utilities for managing virtual environment objects through a computer network, and the environment animation and simulation are pre-programmed as task scripts. The system uses graphical libraries and procedural programming languages to model objects and environments and therefore requires users to have a strong programming background.

Researchers in the University of Maryland [Turner *et al.* 1999] have developed VE construction software – Metis, a toolkit for building immersive virtual environment with environment-independent application computing components. The Metis toolkit defines an application programming interface (API) on the simulator side, which communicates via a network with a standalone viewer program that handles all immersive display and interactivity. The aims of Metis are to create a simple software structure that enables the rapid construction and efficient running of immersive virtual reality applications. It provides several key functional components for rendering virtual objects and environments on an immersive display using an application programming interface (API) and a standalone viewer. An application built by the API works in a client-server mode, where the environment simulation resides on the server side and the user input and output on the client side. This design decouples the application simulation and the visualisation, so they can be performed at different rates and profits from parallel processing. The virtual environment developed using Metis API specifies scenes in 3D space by creating a scene data structure containing geometry, appearance and hierarchical information similar to that constructed in VRML. Rather than using procedural programs (like most of the current commercial VE editors) to control object activities, Metis uses a declarative approach that constructs a network of predefined constraints. Thus, enabling complex relationships and interactivity among virtual objects and users to be logically expressed.

The Metis system enhanced virtual environment knowledge acquisition and representation capabilities by allowing external data and knowledge sources to be connected with the virtual environment through a separate knowledge processing mechanism. This design embraced the power from professional simulation and modelling software and reduced the overhead a virtual environment-rendering package has to carry. However the system still needs its own environment modelling language (a scripting language) to model an environment.

West and other researchers [1993] in the University of Manchester developed a VR application developing system called AVIARY. It has followed an object-oriented (OO) hierarchy where a defined virtual environment with a set of attributes can be inherited by all virtual objects in an instance of that environment. It also specifies a set of constraints which govern the behaviour of those objects. This approach forms a multiple inheritance hierarchy allowing new worlds to be defined in terms of existing worlds rather than from scratch. AVIARY allows multiple worlds with different laws to be concurrently activated. The implementation is composed of loosely connected autonomous objects which execute simultaneously. Some objects will represent objects in the virtual world, other objects act as device drivers for input and output or provide services. An example of one type of object which performs a service is the 'object server' which provides an execution environment for other objects. In the AVIARY system, a user of the environment was treated the same as any other object, the object representing the user will normally communicate with at least one input object and at least one output object with no restriction of the number of system users.

Based on the AVIARY system, researchers at the Advanced Interface Group (AIG) in the University of Manchester has taken the research forward and developed a system which tackles problems on constructing large-scale applications [Cook *et al.* 1998]. This system aimed to deliver the performance and flexibility required by the large-scale complex application environments, which addresses the graphics, interaction, distribution and systems architecture problems. Similar to the Metis system described above, this proposed system also separates the simulation task and environment rendering process into two core components, Maverik and Deva: Maverik manages the world as a participant perceives it, and Deva manages the "reality" behind this perception. The Maverik has functions for optimised display management including

culling, spatial management, interaction and navigation, and control of VR input and output devices. It allows data exchange between its core functions and external application data so that optimal representation and algorithms can be employed. Deva extends Maverik to support the distributed applications and controls the environment simulation.

This approach has merits over other aforementioned systems in two aspects, it provided a structure which allows a large number of virtual objects with reasonable geometric detail having dynamic and interactive features. The system relies on its own geometric, simulation and interaction modelling functions (micro-kernal services) for developing applications which might restrict its compatibility with other environment authoring toolkits.

In contrast with the above systems, manufacturing applications using VR techniques require more specific environment construction and knowledge management approach. VR-SIM developed by Gimenez and Kirner (1997) integrated a manufacturing simulation knowledge base with virtual reality software to validate real time simulation system. It consists of a set of reusable software components to facilitate the preparation of VR simulation sessions. It aimed to analyse the behaviour of both the software system and the environment around it. VR-SIM provides its users with pre-built environment objects and routines to connect these objects with the user's software system. The system was composed by a real-time scheduler, an environment manager, a communication manager and virtual reality components. The prototype system was implemented with WorldToolKit (WTK).

An early model of a virtual environment based mechanical design environment was developed by Barrus (1994) in Massachusetts Institute of Technology (MIT). The goal of the project was to develop a simulated workshop for designers to do conceptual design work while taking into account manufacturing processes information. The virtual workshop contains a set of manually operated machines, for instance, band saw, drill press, milling machine, radial arm saw, and table saw. Using the handles and locks on each of the machines the user is able to create components from a selection of materials in various sizes. The system does not present the workshop as a whole environment but with only one machine visible at a time. The

required machine being selected from a menu and the display is updated to show that machine.

Washington State University developed a virtual reality based manufacturing design system called VEDAM (virtual environment for design and manufacturing) [Angster 1996]. It intended to extend the capabilities of current parametric CAD/CAM systems, and has been partially implemented to support virtual design, virtual manufacturing and virtual assembly. The objectives were described as creating a system that allows designers to incorporate virtual reality techniques into the design and process planning stages of the product, while providing a flexible, expandable and customisable environment directly linked to a parametric CAD/CAM system to analyse the entire system and design and create a prototype of the integrated virtual product development system.

As stated in their publications, a test implementation was tested with the following features:

- Functioning virtual manufacturing equipment including a lathe, a mill, and a water jet.
- Data converter which enables importing objects from parametric computer-aided-design (CAD) system, such as AutoCAD from AutoDesk Ltd., into virtual environments.
- Allowing loading and verifying numerical control code created from a computer-aided-manufacturing (CAM) system.
- Real-time graphical presentation of machining processes.
- Automatic design modification based on the virtual manufacturing result.

VEDAM system explored several areas in which virtual reality can assist in the design and manufacture of a product. These include parametric design changes within a virtual design environment, virtual assembly, virtual manufacturing, and human-integrated design. The system has been linked to parametric design software - Pro/Engineer. The proposed system has five main components, the Machine Modelling Environment (MME), the Virtual Design Environment (VDE), the Virtual

Assembly Environment (VAE), and the Virtual Manufacturing Environment (VME). During a design session, the user would enter into the virtual environments via the main interface to test designs or manufacturing ideas. VEDAM, combined with a parametric CAD/CAM system, would provide a complete system for engineers to evaluate potential designs and process plans. There are several areas in which further research and development can be conducted, including the user-part interaction, the user-machine interaction, and the control system for the virtual manufacturing environment. The system was created on a Silicon Graphics Crimson workstation with Reality Engine graphics. All classes were developed using C++ and the graphics were created using Performer 2.0. The virtual reality hardware used in this implementation include a Virtual Research VR4 helmet, a Virtual Technologies 22-sensor Cyberglove, and an Ascension Flock of Birds tracking system with an ERT and six birds.

The University of Bath developed an interactive virtual manufacturing environment for part design based on a real workshop [Taylor *et al.* 1995]. It contains a 3-axis MatchMaker milling machine, a Cyclone lathe, a robot, and a Roland modelling machine. The system users can model a virtual environment by customising these manufacturing resources and processes available in the real world. The user is then able to manufacture new components adhering to their design specification using only the available processes. The output of the processes is a geometric model of the new component. It also claimed that the designer's actions could be translated into machine codes for each of the processes that have been carried out. This method of design has been termed Design By Virtual Manufacture, as the designer works within a computer-generated factory and is constrained to produce components using only the machines available within the factory, which contrasts with the tradition of design then check for manufacturability. The system has been developed on a Silicon Graphics Onyx RealityEngine2, using the Silicon Graphics OpenInventor 3D graphics toolkit and a kernel geometric modeller.

Researchers in the University of Texas [Chuter *et al.* 1995] developed the methodology of using an agent-based framework to specify a virtual environment. It claimed it was flexible enough to support scheduling, planning, and behaviour modelling. The implementation consisted of a heterogeneous, distributed interactive

virtual environment using the virtual environment tool – DIVE. In their research, agents referred to grouping a set of resources in the common operations, akin to a class definition in object-oriented programming languages, and objects refer to class instantiations. The VE development was used in the design or redesign of a instantiation. The VE development was used in the design and redesign of a manufacturing shop-floor. Information required for the construction of the VE included: (i) domain specific information, such as the purpose of the VE, important activities to be performed, resources involved in performing these activities, and the constraints involved, (ii) tools and resources required to construct the VE, and (iii) tools and resources required for interaction with and manipulation within the VE.

Objects that comprise an agent description collaborate with each other and each was associated with an agent specification. In each mode the agent has three macro-states of operation: Idle, Active and Down. The design of agents along with their communication and behavioural models defines the final VM prototype. A complete VM prototype defines a test-bed for the analysis of the system. For example, a surface model is created using the system. A robot is constructed with the base of the robot being defined as the top object and the joints being defined as sub-objects. Forward and inverse kinematics algorithms were applied on the virtual robot. It has simulation and communication functions supported by various agents. Communication within the virtual manufacturing system is between agents. Every agent can generate input events for other agents and each agent can become active upon receiving an input event. The definition of the robot in an environment is using an ASCII definition file similar to VRML.

Tian (*et al.*) [1997] in the Northwestern Polytechnic University developed a simulation program, with a message-driven object-oriented programming system, which promised a solution to two problems in VR programming - the complexity of the world construction and the organisation of the structure of the program. In the proposed system, all the descriptions of objects data were stored in a database which had a direct link between a description and its corresponding geometrical model, making it ready to be browsed. The system used object-oriented programming for object definition, and adopted a message-driven mechanism to define and manage object interactions. It relied on a message-driven object interaction management

(MDOIM) system to deal with triggering messages and object responding functions. MDOIM helps programmers to concentrate on the following aspects during the software development, (a) find and define the necessary flow in the real world, (b) construct related message response functions. When the definition task of a message is completed, the system will run automatically with Message-Manager continuously dispatching messages and objects to react with the message they receive. All messages, including hardware messages and user-defined messages, are processed in the same way so that all VR worlds built up with MDOIM share a uniform interface of information exchange and processes.

MDOIM was different from WTK. The former was not a mere set of functions but a program environment which was completely compatible with OOP and had a uniform template for VR software development. Working with MDOIM, programmers can concentrate on a child object's particular message response functions without repeating coding inherited from its parents. The second benefit of MDOIM is its ability to be expanded. As all the reaction in the system was triggered by messages, the programs are organised by message response functions and the real driving force in the system is the flow of messages. For communication between different virtual worlds, a message translator was created. All virtual worlds register to a central message translator that their messages related to interactions that may take place, and during operation the trans-world interaction is realised by the delivering or interpreting of messages by the message translator. This means the possibility of merging different worlds without the need to rewrite programs completely will be an aid for distributive software development.

Polis and other researchers [1995] in the Carnegie Mellon University in the USA used a database for constructing large-scale virtual worlds by integrating information from various sources. Such virtual world databases have significant applications in training, planning, and autonomous-agent simulation. Virtual environment construction is based on transforming heterogeneous sources data at multiple spatial resolutions into a consistent geometric representation with a single scale and base line spatial resolution.

Sequeira [1999] has developed an integrated approach to the construction of textured 3D scene models of building interiors from laser range data and visual images. This approach was a collection of algorithms and sensors within a prototype device for 3D reconstruction, called Environmental Sensor for Telepresence (EST), which takes the form of an autonomous mobile platform. The Autonomous EST (AEST) provides an integrated solution for automating the creation of complete models. Embedded software modules perform functions to triangulate the range data, register video texture, and integrate data acquired from different capture points. The reconstructed model is encoded in VRML format to access and view via the World-Wide-Web (WWW).

2.4 DISCUSSION

The survey of VE applications show VE has promised many advantages over other solutions, but (i) current VR software that builds the synthetic environment is a tedious and painstaking process similar to that of the CAD drawing process. The development work of virtual environments is mostly dependent on experienced computer programmers, and (ii) the created virtual environments generally have limited real application domain knowledge and are mainly for presentation with pre-defined animation and simulation routines, which generally have little flexibility for re-configuration. The first problem is crucial for rapid application environment construction and the second is critical to environment usage.

CHAPTER 3

**AN OVERVIEW OF THE VE CONSTRUCTION
APPROACHES**

This Chapter is dedicated to the problems of rapid modelling and constructing virtual manufacturing environments highlighted in Section 1.5 and to achieving the research objectives specified in Section 1.6. It starts with an evaluation of existing approaches and presents a novel approach called 'domain-analysis based top-down construction', which overcomes some of the difficulties in the conventional VE construction approaches. Techniques for representing manufacturing knowledge within virtual environments are then reported based on this approach including the methods for acquiring knowledge from the environments. The chapter concludes with a description of the implementation of the approach.

3.1 CONVENTIONAL APPROACHES TO CONSTRUCTING VIRTUAL ENVIRONMENTS

The literature review showed that approaches to constructing virtual environments fall into only three categories: (i) bottom-up generative approach, (ii) building-block approach, and (iii) variant constructing approach.

3.1.1 Bottom-up generative approach

As shown in Figure 3.1, the bottom-up approach constructs a virtual environment by starting with a basic shape design using geometric primitives like points, arcs, lines and facets. The shapes are often in 3D and used directly to construct more complex 3D virtual objects.

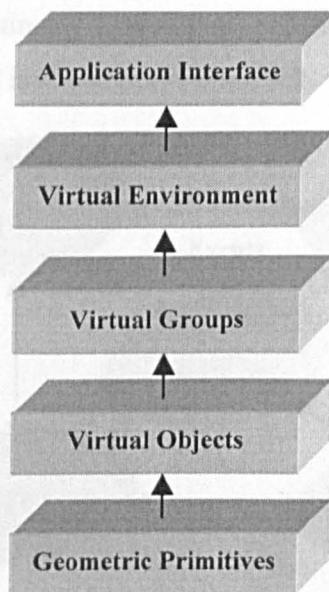


Figure 3.1 Illustration of the bottom-up approach

The virtual objects are 'accumulated' to form a virtual group that represents an independent and meaningful virtual sub-world, for example, a virtual lathe. The resulting virtual environment normally contains a single or a few virtual groups with detailed geometric data and has a clear hierarchical structure. However, this approach is time consuming when it is used for constructing large-scale complex virtual environments, mainly due to its intensive graphical interactions that are found in most

CAD systems. The constructed virtual environment often lacks of re-usability because of its fixed hierarchy and is therefore difficult to reconfigure. Most commercial application-program-interface (API) based virtual environment toolkits adopt this approach, for example, WorldToolKit (WTK) from Sense'8.

3.1.2 Building-block approach

To overcome problems with bottom-up construction, a so-called 'building-block approach' was proposed by Singh *et al.* [1995], Chuter *et al.* [1995], Wang *et al.* [1995] for virtual environments with a large number of objects. It works in an event mechanism in programming terms. This approach relies on a library of pre-built virtual objects that can be 'dragged' into the virtual environment and assigned properties to about which events can be received, under what conditions and how to react. When running the virtual environment, the virtual objects are activated by event-chains which are simply a series of executable messages [Singh 1994]. The structure of this approach is showed in Figure 3.2.

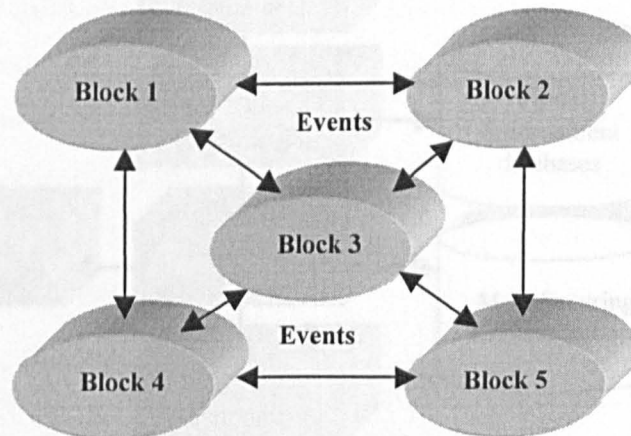


Figure 3.2 Building-block VE construction approach

This approach reduces the time to build virtual environments by using the virtual object library, and the virtual environment can be easily expanded by adding or changing event links [Karacali 1995]. However, the number of events, their types and directions have to be pre-set at construction time and cannot be changed later. This significantly restricts the capacity of environment re-configuration.

3.1.3 Variant construction approach

According to Zhao [1998], the variant approach to constructing virtual manufacturing environments uses a set of similar manufacturing tasks (for example, machining a part with certain features) that can be represented by a single representative task. If a specific virtual environment (for example, a lathe and a robot) exists in the database that can accomplish this representative task, then it should be able to accomplish all marginally similar tasks. A VR environment can be developed from this representative environment. Figure 3.3 shows a simplified diagram of an implementation based on this approach. The representative environment is called a 'master environment' and the representative task is called 'master task'. The master task and the master environment are related to each other and stored together in the virtual environment database. When a specific manufacturing task is required, it is described in a similar format to that of the master task.

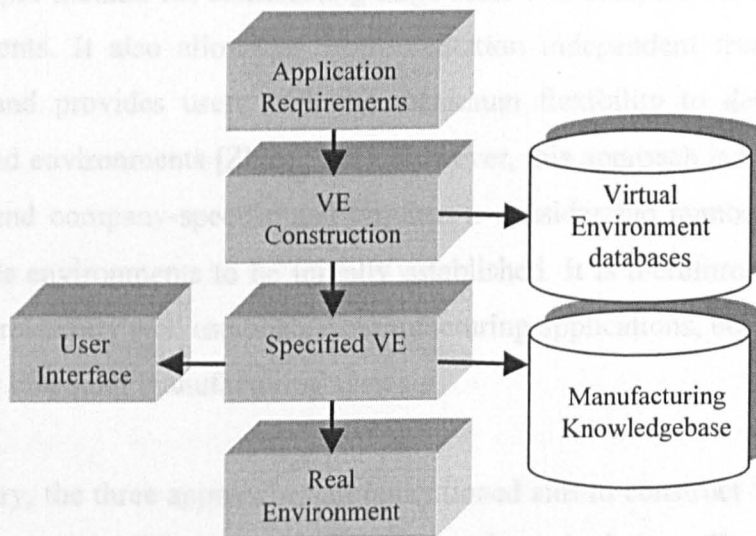


Figure 3.3 Variant environment construction approach

To construct a given VR environment for a given manufacturing task, the index of the formatted manufacturing task is first used to search the virtual environment database, the system then performs the following operations:

- (1) If a master task exists in the virtual environment database that is exactly the same as the given task, then the master environments attached to that task will be retrieved as a required environment.

(2) If, in the virtual environment database, a master task that is the most similar to the given manufacturing task, then the master environment attached to that master task will be retrieved and modified to perform the required task. The amount of modification depends on the similarity between the two tasks - the more similar, the less modification required.

(3) If there is not a similar master task in the virtual environment database to the given manufacturing task, then a new virtual environment has to be created from scratch for the given manufacturing task. The given manufacturing task and the newly created virtual environment will then be stored in the virtual environment database as a new master task and master environment pair.

This variant construction approach is superior to the previous two approaches in that it eliminates the needs for low level programming and graphical work and provides users a rapid method for constructing large scale and complex virtual manufacturing environments. It also allows an implementation independent from individual VR systems and provides users with the maximum flexibility to design and use the constructed environments [Zhao 1997]. However, this approach is highly application-oriented and company-specific and requires a considerable number of master tasks and master environments to be initially established. It is therefore mainly useful for large and relatively well established manufacturing applications, but less applicable to frequently changing manufacturing systems.

In summary, the three approaches aforementioned aim to construct “exhibition-style” environments, i.e., 3D models for display and manipulation. The literature survey found the VE construction systems that can facilitate knowledge acquisition and data management, especially for manufacturing applications, are often large and complex both in their visual form and their internal data repository. Thus a more efficient approach is needed.

3.2 A DOMAIN-ANALYSIS BASED TOP-DOWN APPROACH

A domain analysis-based top-down approach is proposed in this research that can satisfy three needs: (i) to provide a user-friendly and rapid method for the construction of virtual manufacturing environments, (ii) to provide a reusable and re-configurable mechanism of virtual environments for different manufacturing applications and (iii) to allow the capture and presentation of manufacturing knowledge and manage manufacturing data. The proposed approach was implemented with the aid of two concepts: application domain analysis and top-down VE construction.

3.2.1 *Application domain-analysis*

Application domain-analysis attempts to understand, classify and abstract the real world knowledge into a virtual environment and provide realistic guidance for the constructing of the applied environment. For example, a virtual lathe would carry the knowledge of its spindle speeds and the chuck would carry the knowledge of the diameter range it could hold. The approach negates the need to encapsulate all the knowledge in to a single environment.

Consider an environment designed for a process planner, the knowledge needed on a machine tool would relate to spindle power available, job-type and machine capacity. On the other hand, a plant layout designer will merely require data concerning the machine footprint, and possibly, some safety factors. In this way the application domain-analysis actually performs the task of identifying which part of the manufacturing knowledge is going to be applied in a specific environment. In this research, the domain analysis was based on a GT-like coding scheme (see Section 3.3.1 and Section 4.3.1).

3.2.2 *Top-down VE construction*

The top-down approach intends to avoid the time-consuming processes of constructing an environment from scratch, or struggling with mastering an environment authoriser. A set of representative manufacturing environments for different application domains are constructed that can be used as templates as a

starting point to develop an environment. A template environment has defined properties such as suitable tasks, environment size, simulation level and interactive mode which can be used to compare with the required application task and modified to suit.

3.2.3 *Operation mechanism*

The operation of the approach can be described in four steps:

(1) Determining the application domain

Throughout this research, it is assumed that any virtual environment is constructed with one or more specific application purposes in mind. Therefore, before a virtual environment can be constructed, the domain of its applications needs to be known. The importance of this step can be explained in the following example.

The lighting background of a virtual manufacturing workshop is irrelevant if the virtual workshop is to be used for simulating a sequence of machining operations to find out how a robot controlled loading and unloading system works. However, if a virtual environment is for simulating a robot-based vision system for part recognition in a machining cell, the accuracy of the background modelling will be extremely important - the parameters that can be attributed to an accurate virtual background, such as lighting, colouring and texture mapping, will decide the viability and usability of the constructed virtual workshop.

In this research, determining the application domain of a virtual environment means (i) specifying the application requirements, (ii) defining the environment attributes, and (iii) determining a description scheme. Application requirements are specified from the real world. Each participating aspect of this world is assigned an attribute and those attributes are described by a description scheme (a detailed explanation of such a scheme is in given Section 4.3). In this work, all applications are defined by a range of application attributes that are in turn encoded in VR modelling data.

(2) Selecting a candidate environment

This step requires a series of so-called template environments being constructed as ‘foundations’ for constructing new environments. These template environments (each representing a typical manufacturing application) hold the essential manufacturing resources with realistic geometric detail and simulation verified by real world experiences and knowledge. Template environments are ideally stored in a highly structured and logical computing format so that their data can be efficiently managed (a database system was used in this research to complete this task).

Selecting a candidate (template) environment is actually the processes of abstracting the application into a "construction task", which is encoded into a so-called ‘task code’. Since the code is based on Group Technology (GT) code, it can be used to compare with a master task in the database, and to determine a task family within which the application falls. The master task is used to retrieve the associated template environment through an environment meta-code (see Section 4.3).

(3) Environment modification and configuration

The role of the template environments is to facilitate the environment construction rather than to replace it. Once an environment is retrieved from the database, it normally needs to be modified to suit a given application, for example, adding, deleting, and changing objects. In some other cases, the environment and its simulation will need to be configured either by setting the virtual environment through directly manipulating the virtual objects and their properties at system run-time or through configuring the object simulation functions at environment design time. Both ways have been implemented in this research (described in Chapter 7).

(4) Utilising the constructed environment

Conventional virtual environments built for specific manufacturing applications usually design and store simulation functions as task procedures. The realism and richness of the VE activities is a task for the application developer and has little to do with the end-users. Since the target system attempts to shift the environment

construction process from a development-centred approach to a more user-centred approach, a real-time data exchanging mechanism is used to connect the database with the environment properties to allow environments to be altered by updating database records. Such a design opens the potential to control environment appearances and activities by external data and knowledge sources, such as, modelling and simulation packages, and expert systems, through the database.

3.3 IMPLEMENTATION

The implementation of the domain-analysis based top-down VE construction can be divided into six phases, task coding design, template environment development, database design, environment and database communication, virtual and physical environment links and the development of a unified computing infrastructure.

3.3.1 *Design application task coding scheme*

The implementation of the new VE construction approach starts from designing a task coding description system. A task in this research is taken to mean producing a part. The task code is based on Group Technology (GT) for classifying task models to take advantage of the parallelism between the GT code for virtual environment simulation models and the GT code for manufactured parts. Given the part GT code, the simulation environment should be able to suggest alternative manufacturing methods and simulation scenarios based on the user requirements.

The format of a computerised task is a string of digits where each digit stands for a specific task property. These properties are organised in three groups: (i) General application information, including the type of environment application, the level of detail, and the interaction mode. (ii) Part geometry information, including part type, external shape and external shape elements, internal shape and internal shape elements, plain surface machining, auxiliary holes and gear teeth. Rotational parts are classified by their length/diameter ratio and non-rotational parts by their length and length/height ratios. (iii) Environment scale information decides the scale of the environment. The task coding scheme provides the foundation for identifying VE

applications in a rapid and accurate way, and will be used to weight the significance of each of the environment properties.

The task coding process has been implemented as a classification programme that enables the user to identify task attributes and describe them in a coded format. This programme uses a 3D virtual encoding panel, as shown in Figure 3.4(a) that is similar to a remote controller with a virtual liquid-crystal-display (LCD). Users have to interpret application task attributes by pressing buttons on the panel to record it. At any stage, a user can delete or insert new data. A built-in task encoding processor is activated to record, interpret, and order the user task. The output from this process is a string of task codes that can be used as a searching index to find a suitable environment from the database.

Figure 3.4(b) shows an encoding interface to the coded environment. By dragging and moving the slide bar users can browse the available tasks in the database. By highlighting a task number, a graphical view will be displayed in the preview window. The matched environment can be loaded by click the "Load Master Environment" button.

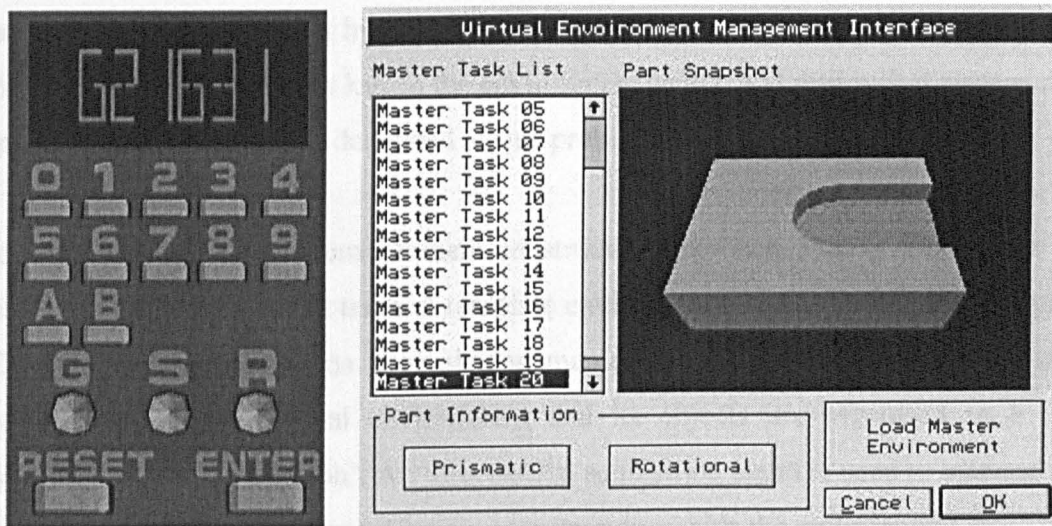


Figure 3.4 (a) 3D encoding panel. (b) Task-based environment retrieval

3.3.2 *Constructing template environments*

Like the variant approach described in Section 3.1.3, the top-down approach also relies on a set of pre-defined virtual environments called templates. A template encapsulates all the basic functions of similar environments. Various possibilities for using an environment have been considered and designed into the templates. All a user needs to do is to determine the best-fit template for a given application and to use the application requirements to populate the template, which results the expected environment.

From the system developing point of view, these possibilities relate to environment comprehensiveness, object detail-level, simulation execution speed, information exchange between software packages, and the communication between a virtual and the real world.

The benefits of this approach come from the environment capabilities themselves and their ability to be modified. The challenge of this work resides in the creation of an environment coding system which can represent both parts of the information in an integrated format. Because the application task is also represented by a string of codes based on information input by the user, so the relation between the task code and the environment code holds the key to the environment search and retrieval at system run-time. The program module dedicated to this problem is explained in Chapter 6.

To test the proposed new environment construction approach, and to accomplish the formation of representative tasks, 5 template environments were created and analysed (Chapter 5). Each contains a particular manufacturing device to fulfil the task requirements. Every virtual environment and its objects are organised in a tree structure called 'scene graph'. An information acquisition agent is used to manage the environment data divided into 4 levels corresponding with the environment structure. An agent is a dynamically linked program module (DLL) which encapsulates certain knowledge acquisition skills and is able to communicate with other agents and programs. The knowledge infrastructure of the virtual environment is classified and explained in the following sections.

(1) Environment level knowledge

Environment level knowledge is initially assigned when a manufacturing environment is generated through the top-down approach. This concerns information such as name, size and master object. It also deals with workshop spatial layout information, including object size, position, and object relationships. Information such as dynamic viewpoint route and VR peripheral configurations also belongs to this knowledge level. This level information is vital when the environment is used to simulate manufacturing cell level activities such as designing a factory or cell layout, controlling inter-cell manufacturing resource flow, or scheduling production.

(2) Object level knowledge

Objects are not primitive shapes used as building blocks. They are complete working systems where their physical counterparts are easily identifiable in the real world. For instance, for a lathe or a robot, object level knowledge would include machine size, weight, power input, machining capacity, tooling and fixture type information. Machine operation procedures, for instance, detailed system behaviours are also included.

(3) Element level knowledge

Element level knowledge encompasses physical information about virtual devices such as element scaling, rotation, translation, and physical constraints, for instance, the maximum distance a part moves. The physical characteristics such as restitution, gravity, friction, velocity, acceleration, and collision are also attached at this knowledge level.

(4) Object property knowledge

The object property knowledge is the lowest level in the knowledge management hierarchy. It deals with the static properties of an object such as shape, size, position, center of rotation, colour, lighting, and texture.

The main application program, responsible for planning, scheduling, and simulation software can interact with the virtual environment at different levels, so to provide virtual environment flexibility and re-usability.

3.3.3 Database design

Most commercial VR software packages separate the environment editor from the environment executor (or visualiser), so that once the design is finalised and distributed, the user cannot change it except to perform the pre-defined tasks. Also, environment authorisers store the environment as a readable script file or unreadable binary code. For handling a few virtual environments, this method may be fine. However, when the disk file list gets very long, the name of each makes less sense to the user who wants to find and explore an environment. Due to this, it is unwise to save each of the environments as a separate copy rather than just an index for all of the distinctive objects and their static and dynamic features. Users can easily browse through all the required information and construct the environment at run-time. This research has an embedded database connection that enables users to retrieve an environment or save changes into the database. This allows environment modification at system run-time. Chapter 6 described such a structure in detail.

In computing terms, a virtual environment is a set of descriptive programming code managed in a specific format and stored in a data file. The information involved in the environment construction and knowledge acquisition is in many formats and therefore difficult to manage. An introduction to the research on using database technology to manage this information follows, a full explanation is given in Chapter 6.

(1) Analysis of stored data

The database is divided into virtual environment data for the rapid virtual environment construction, and manufacturing data for the environment implementation. The virtual environment data includes the world layout data, main manufacturing equipment group information, object geometrical data, and dynamic information. The manufacturing data includes master task description data, virtual machine command and machine specification data.

(2) Evaluation of database capabilities

A relational database model has been adopted whereby every template environment in the developed database is described by a data file that has a cluster of data fields. The manufacturing information such as tooling and machines is stored in separate files. Every data file is related to other files in one-to-one, one-to-many, and many-to-many relationships as required.

(3) Database interface and integration

A virtual environment manager has been developed to obtain data from the database in response to a request from the user. To maintain consistency, all requests are channelled through the Manager, although once the data access channel been set up the application is free to access it as it wishes, without the necessity to incur the overhead of the Manager. The Database Manager has three logical parts. The first is concerned with physical data retrieval – the Database Interface. The central section, Query Base, contains the rules and regulations for filtering, sorting, managing and storing retrieved virtual environment data. Finally, the Application Interface is concerned with representing data to the application and responding to requests through dynamic link library modules.

(4) Database actions

The database and environment manager contain rules to automatically trigger database actions when specific conditions are detected, to alert users when unusual or interesting data conditions arise, or to automatically maintain pre-specified constraints against the data.

3.3.4 Linking VE properties and database records

Every virtual environment is composed of a number of virtual object that form the spatial layout of the environment. These virtual objects can have different tasks, some provide landmarks to give users a visual impact when the environment is applied. Others have dynamic characteristics such as animation or simulation. Every virtual

object is composed of many low-level elements such as primitive shapes that have static properties such as colour, lighting, texture, and even shape geometry that determine their appearances in the environment. The layered structure used in this allow the connection of the virtual environment information to the manufacturing environment and facility knowledge so that the object in an environment is no longer just a graphical representation of the real object but also has a meaningful counterpart. Depending on the application, the system at run-time allows user to interact with the environment in one of three modes: object-level interaction, machine-level interaction or cell-level interaction.

3.3.5 Connecting the virtual and physical world

This research explored connecting the virtual and physical worlds through testing an interface between the virtual and real robots (Puma 560 and LANSING IOII). An interface card was designed to translate communication signals between the computer and the robot control units. A communication scheme programmed using C language initially connected the virtual environment with a physical manufacturing cell, which contained an Audit lathe, a Bridgeport milling machine, a Puma560 robot and a Lansing Robot. The virtual workshop receives and sends information from or to the real equipment through RS-232 serial ports on a multi-port PC card (detailed in Section 7.6).

3.3.6 Integrate function modules under a unified system structure

The proposed approach encompasses virtual environment construction, knowledge representation and acquisition, manufacturing simulation and machine cell control into a single platform. The name of the proposed system is 'Knowledge Acquisition and Management in Virtual Reality', abbreviated to KAMVR. Its structure and development is described in Chapter 4.

3.4 CONCLUSION

KAMVR avoids the tedious environment construction work of traditional virtual environment systems and allows environment information handling using a structured

computing tool. Conventional VR systems need to import data into their own format, but KAMVR avoids this by interpreting application internal data structures in the environment database system, which can take advantage of application specific knowledge tied to environment presentations.

CHAPTER 4

SYSTEM ARCHITECTURE

This chapter describes a VR system for knowledge acquisition and management (KAMVR) that demonstrates the viability of the domain-analysis based top-down VE construction approach devised in Chapter 3. It provides an introduction to the structure of the system and then describes each of its components.

4.1 KAMVR SYSTEM ARCHITECTURE

KAMVR is an acronym for knowledge acquisition and management in VR. The architecture of the system is shown in Figure 4.1. It has four layers. Each with its related functional modules. Layer 1 is an interactive virtual environment interface used to create, visualise and interact with the virtual environments. It also hosts the input/output (I/O) of virtual reality peripheral devices such as HMD, electronic gloves, sound devices and imaging systems.

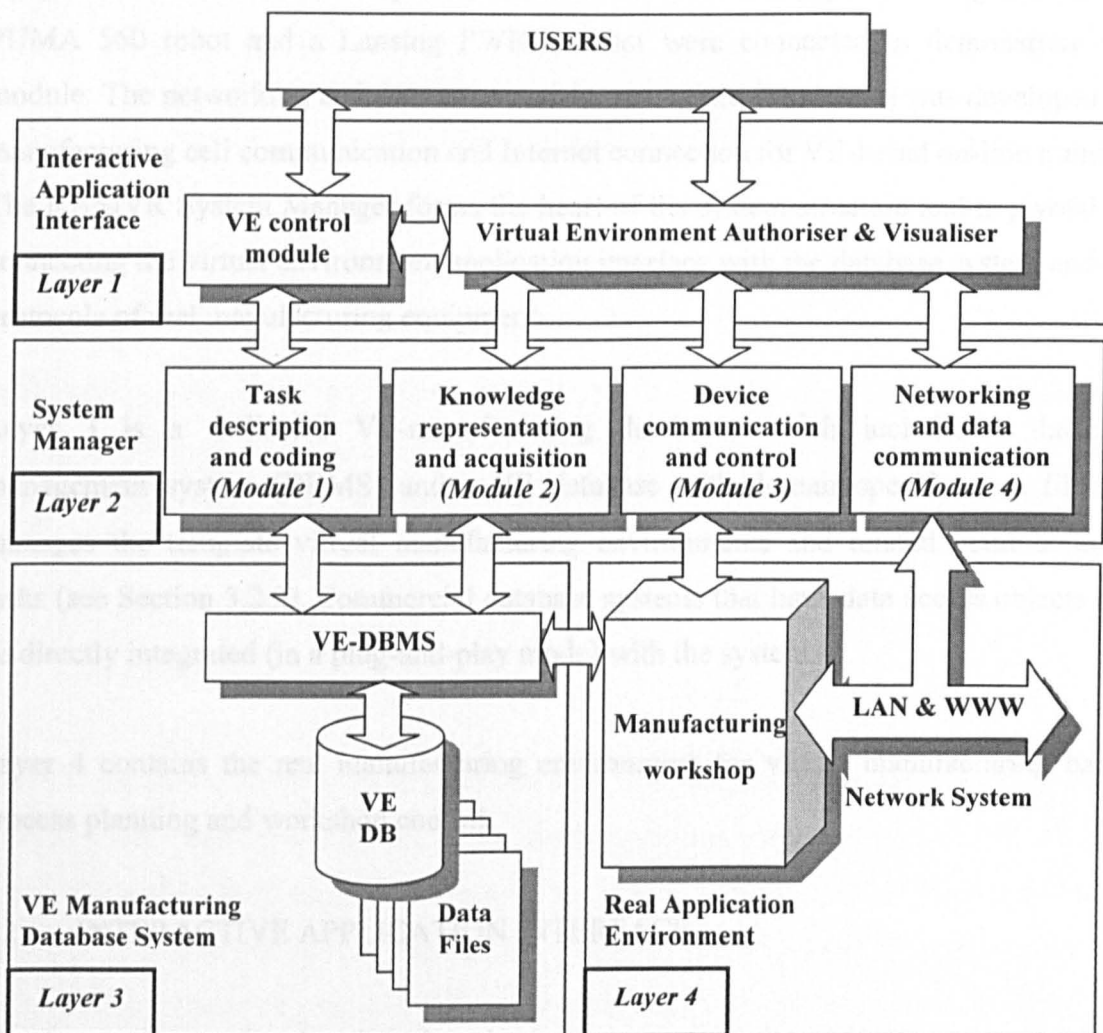


Figure 4.1 KAMVR system architecture

Layer 2 - the 'System Manager' - is the main knowledge handling and data management layer with four constituent modules. The task description and coding module (Module 1) is a programming utility for application domain-analysis and task coding based on the template environments and manufacturing information stored in a VE database. The knowledge representation and acquisition module (Module 2) supports the real-time data exchange between the virtual environment content and the database for environment retrieval, modification, and configuration. The device communication and control module (Module 3) is a program for processing signals received from real world manufacturing equipment. In the KAMVR system, an Audit lathe, a Bridgeport milling machine, a PUMA 560 robot and a Lansing PWIOII robot were connected to demonstrate this module. The networking and data communication module (Module 4) was developed for manufacturing cell communication and Internet connection for VE-based on-line training. The KAMVR System Manager forms the heart of the system structure and is pivotal for connecting the virtual environment application interface with the database system and the protocols of real manufacturing equipment.

Layer 3 is a dedicated VE-manufacturing database, which includes a database management system (DBMS) and a VE database with domain-specific data files. It manages the template virtual manufacturing environments and related manufacturing tasks (see Section 3.2.2). Commercial database systems that have data access objects can be directly integrated (in a plug-and-play mode) with the system.

Layer 4 contains the real manufacturing environment for virtual manufacturing based process planning and workshop control.

4.2 INTERACTIVE APPLICATION INTERFACE

The interactive application interface is the top layer of the KAMVR system. Users visualise and interact with a virtual environment in one of two ways through this layer: (i) directly interact with the virtual environment using the default visualiser navigating and editing tools, or (ii) through the VE control module in Layer 1.

4.2.1 Visualiser interaction

This method is used to navigate and explore the manufacturing environment. To do this, it supports VR peripherals, such as i-glasses, data gloves, and a 3D-digitiser as shown in Figure 4.2.

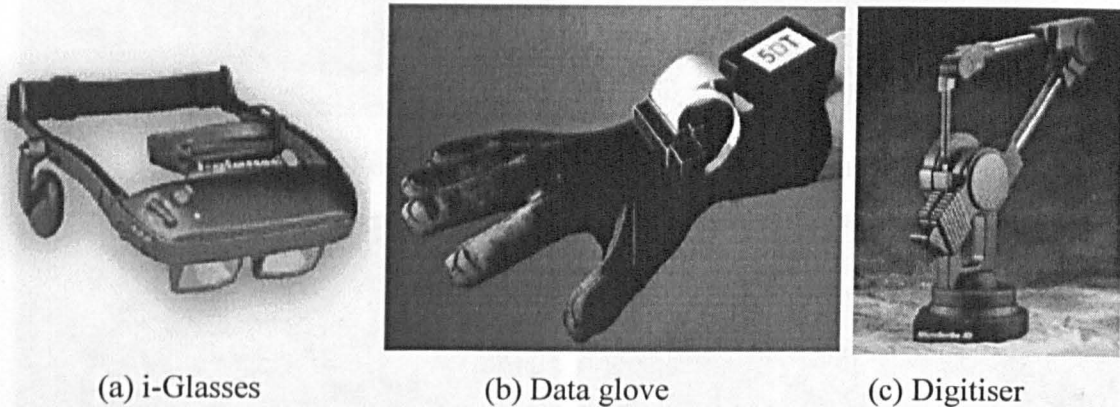
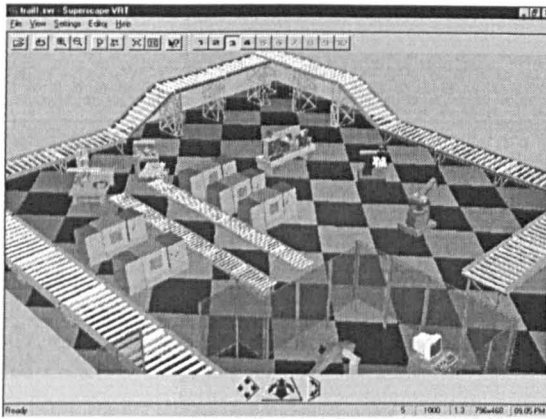


Figure 4.2 VR devices used in the KAMVR system

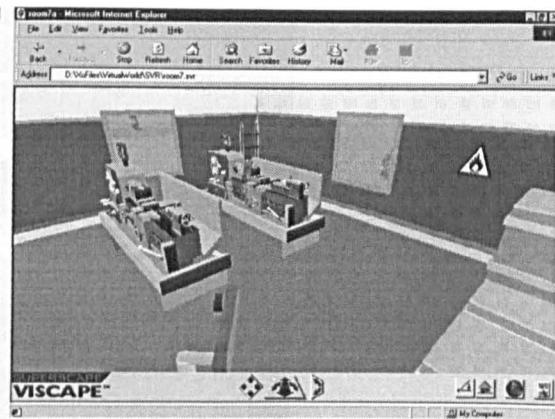
Additionally, the basic software components of Visualiser, virtual world editors, graphical libraries, I/O configuration utilities form a complete VR system. Superscape VRT software [VRT Manual 1997] was adopted for the visualiser interaction which provided the following functions.

(1) Visualiser

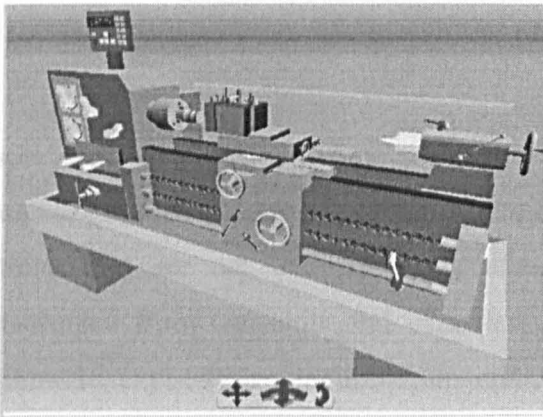
The VE Visualiser provided by Superscape VRT is a program interface for users to view, navigate, and manipulate a virtual environment on computers or immersive VR devices. Other commonly used visualisers include Visualiser, Viscape, 3D Control and Cosmo Player as shown in Figure 4.3. The Visualiser employed in the KAMVR system provided a tool set for connecting various VR peripherals, such as i-glasses, shuttle glasses, data gloves, and sound devices. A Navigation Bar and Viewpoint Setting dialog box supports static and dynamic viewpoint changing in a virtual environment. Special device drivers were developed for connecting Visualiser with MicroScribe-3D digitiser, 5th GLOVE data-glove, and VIRTUAL i-glasses.



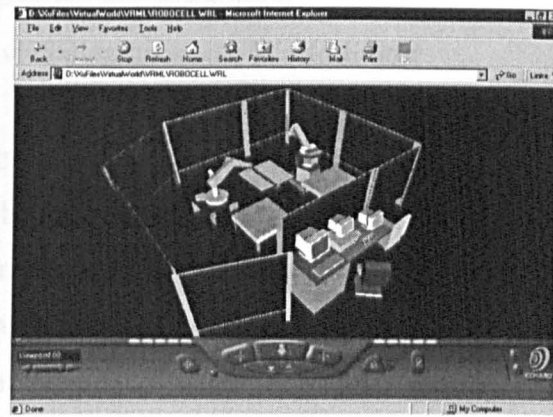
(a) Superscape Visualiser



(b) Superscape Viscape



(c) Superscape 3D Control



(d) COSMO VRML Player

Figure 4.3 Environment Visualisers

(2) World Editor

Superscape VRT provides a World Editor to create, sort, group, organise, and assign virtual objects and functions in a graphical environment. It also enables real-time interactions between the virtual world and users, and amongst virtual objects themselves. The KAMVR system used it for developing the template virtual manufacturing environments (see Chapter 5). Figure 4.4 shows an example of using the World Editor.

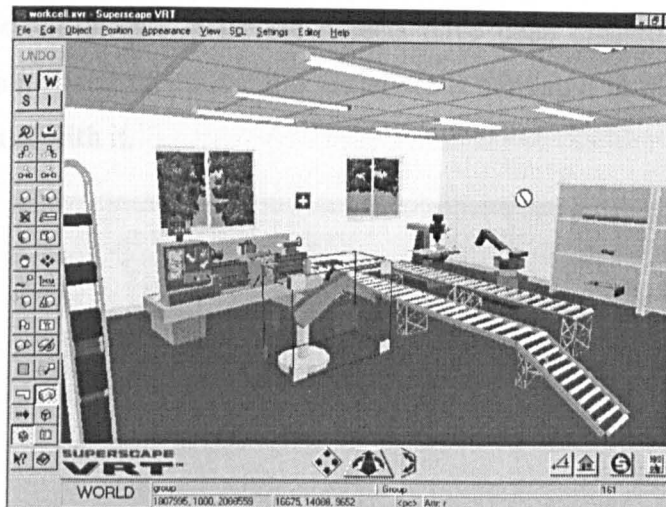


Figure 4.4 Constructing a virtual workshop using the World Editor

Superscape VRT has a C-type simulation control language called Superscape Control Language (SCL) for programming virtual object behaviours. This was used to perform actions that are not undertaken by automatic system routines. SCL programs are constructed from constants and variables, with support for arrays, pointers, functions, arithmetic expressions, input/output, control and conditional structures. However, to implement a complex manufacturing system, virtual events could occur during the environment interactions, which are difficult to predict and simulate. For this reason, an object-oriented environment construction and interaction using System Manager was researched and developed which is explained in Section 4.2.2.

(3) Shape Editor

Every object in an environment uses a specified shape to define its geometry. The template environments developed in KAMVR include a large number of shapes designed to form complex virtual objects such as lathes, robots and other machines. The Superscape Shape Editor uses vertices and facets to create 3D objects as shown in Figure 4.5. A bounding cube is always associated with an object and is used by VRT to sort an object's spatial relationships and detect collisions in the VE. In KAMVR, SCL can be used to construct 3D objects using procedure codes. World Editor uses object descriptions from the Shape Editor to build virtual environments organised as a tree of

objects in the environment. After the objects have been created and given all of their necessary attributes and functions, the visualiser is activated to display the environment, move and interact with it.

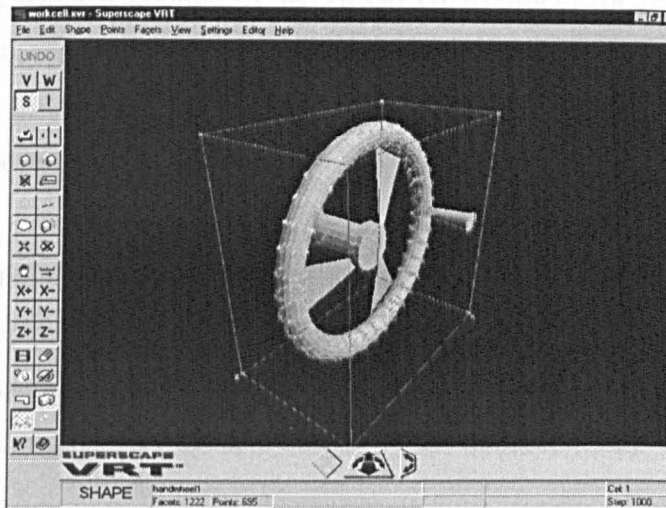


Figure 4.5 Modelled handwheel in the Shape Editor

(4) Resource Editor

The Resource Editor has been designed as an independent VR editor utility to develop dialog boxes, menu bars and other controls for virtual object controls.

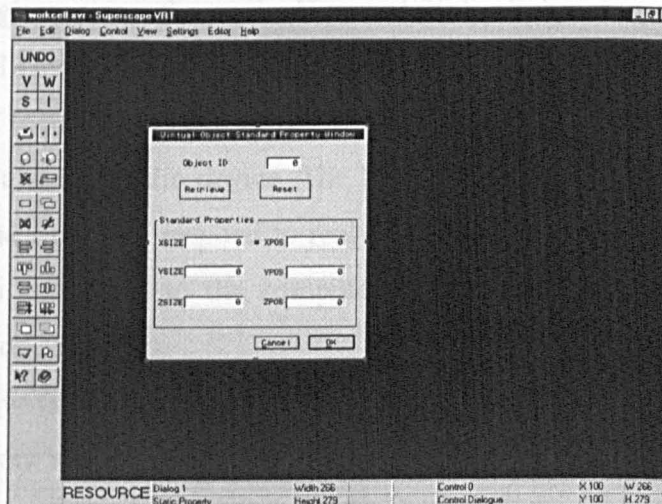


Figure 4.6 Superscape Resource Editor

Combined with a corresponding SCL program, the Resource Editor enables users to allocate buffer size and data addresses for control program parameters. This allows customised interaction with a virtual environment through windows resources. Figure 4.6 shows a dialog window applied in the virtual environment for retrieving virtual objects and standard information such as object size and position.

(5) Other editors

Other editors, including Image Editor, Layout Editor, Sound Editor and Keyboard Editor provided by Superscape VRT system are affiliating editors for serving various virtual environment design purposes. The KAMVR system was designed to allow these tools to be facilitated through the VE control modules (See Section 4.2.2).

(6) System hardware

KAMVR was developed on a PC running under the Windows NT 4 operating system. The configuration was a Pentium 100M CPU, 48M RAM, 1M graphics memory and 2G hard disk. The hardware VR devices used were as follows:

- VIRTUAL i-glasses (as shown in Figure 4.2(a)): It has two full-colour 0.7 inch Liquid Crystal Display (LCD) with a resolution of 180,000 pixels and 30 degree field of view. The focus distance was fixed at 28cm with a 100% stereo overlap. The interface was a stereo audio device. The Motion Tracker can record 3 motion types, Yaw, Pitch and Roll, and respond to real-time LCD display update correspondingly. The device is connected to the computer through the RS 232 serial port with a sampling rate of 250Hz.
- 5DT Data Glove (as shown in Figure 4.2(b)): The device resembles to a normal glove with five fingers and wrist attached with wires and sensors. It has 8 bits to define resolution for each fingers (maximum 256 positions). The axes of the tilt sensor on the hand waist can identify hand gesture for Roll and Pitch. It uses flexor sensor

technology and measures the average flexure of each finger. The Data Glove uses the serial port to communicate with the computer. The sampling rate is 200Hz.

- MicroScribe-3D Digitiser (as shown in Figure 4.2(c)): It has a position resolution of 0.13mm and position accuracy at 0.38mm. The maximum arm stretch distance is 50 centimetre. It also uses serial protocols to communicate, the data measured can be recorded into an Excel datasheet for processing. Figure 4.7 shows the system hardware connections.

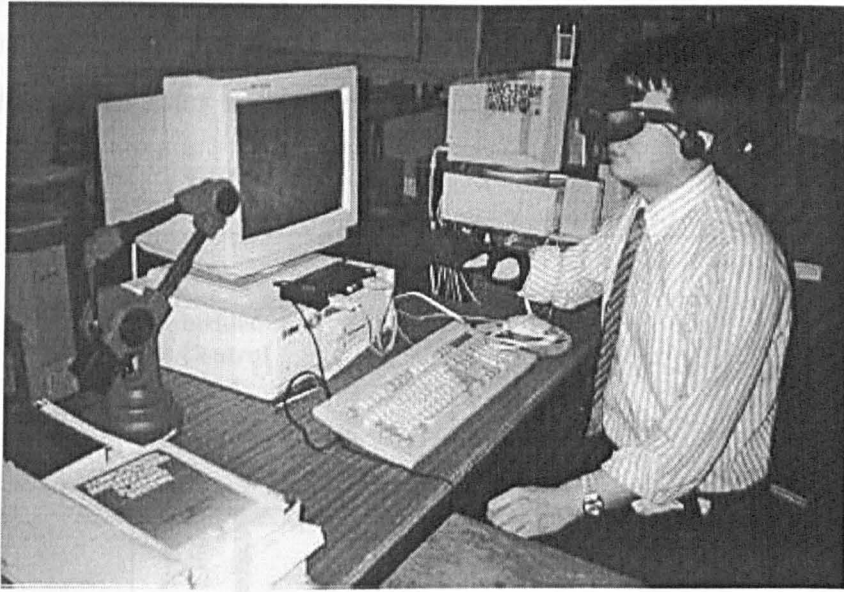


Figure 4.7 KAMVR system workbench

4.2.2 VE Control module interaction

In contrast to the visualiser interaction, the VE control module interaction was designed to be based on “message flow” controlled by a centralised controller. This type of interface still needs a visualiser to display a VE on a displaying device. Superscape VRT is used for this purpose, but it is implemented as an embedded computing object, not a system as in visualiser interaction, in the centralised controller. The centralised controller is a programme that has following functions:

(1) Data flows and communication:

This function has been provided due to two underlying reasons: (i) Most commercial VR systems have their own application program interface (API), programming functions and libraries. This design brings VR programs the strength of integrity. However, it is difficult to gain distributed control for the virtual environment. (ii) The virtual environment related knowledge and data management process takes considerable computer processing time and memory, which should be preserved for rendering the virtual environment. In fact, this kind of data management can be fully carried out by other standalone programs such as spreadsheet, database, and specialised mathematics programs.

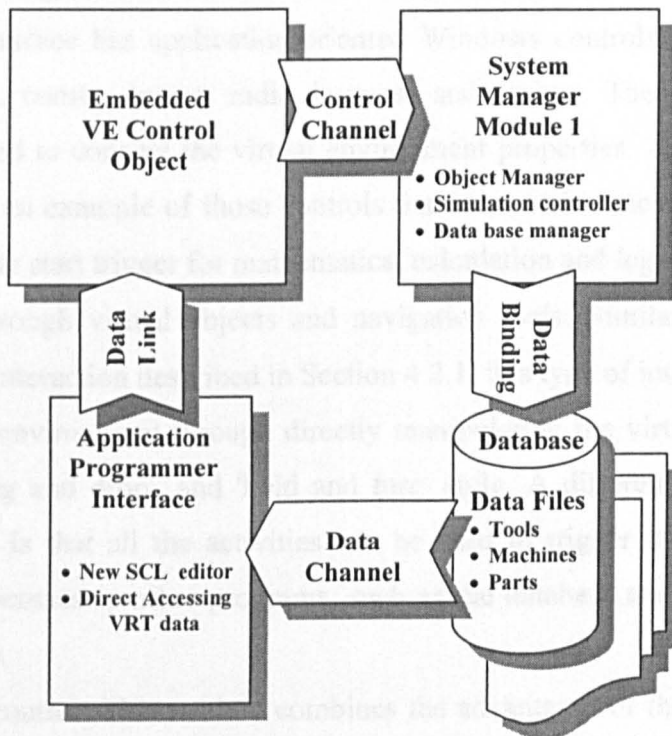


Figure 4.8 Data and command flows in the KAMVR Module 1

Figure 4.8 shows how the centralised controller handling data messaging and communication using four program code blocks and four communication channels in between of each pair. Every program block can transmit/receive command (events) from others, this allows, for example, a user to load an environment on to the visualiser, and

then explores it using the centralised controller functions. The user control activities can be monitored and used for modifying virtual object information into a database using the internal functions of the KAMVR System Manager. Vice versa, the updated environment data retrieved from a database can also be used to modify an environment through the virtual object property and database record links.

(2) Environment interactions

There are three environment interaction methods that have been provided by this research:

- (i) Control through the Windows based control resources: The virtual environment control interface has application-oriented Windows controls such as dialog boxes, file filters, combo boxes, radio buttons, and sliders. These controls have been programmed to connect the virtual environment properties. The right side of Figure 4.9 shows an example of those controls that can receive messages sent by a virtual object as the start trigger for mathematical calculation and logical conjecture.
- (ii) Control through virtual objects and navigation tools: Similar with the standalone visualiser interaction described in Section 4.2.1, this type of interaction interacts with a specific environment through directly manipulating the virtual object in a 'click', 'push', 'drag and drop', and 'hold and turn' style. A difference from the visualiser interaction is that all the activities can be used to trigger the communication and control processes to other programs, such as the database and system manager (see Figure 4.1).
- (iii) Synthetic control: This method combines the advantages of the aforementioned two controls, and gives more flexibility. It was used to create a fully functional virtual lathe as shown in Figure 4.9.

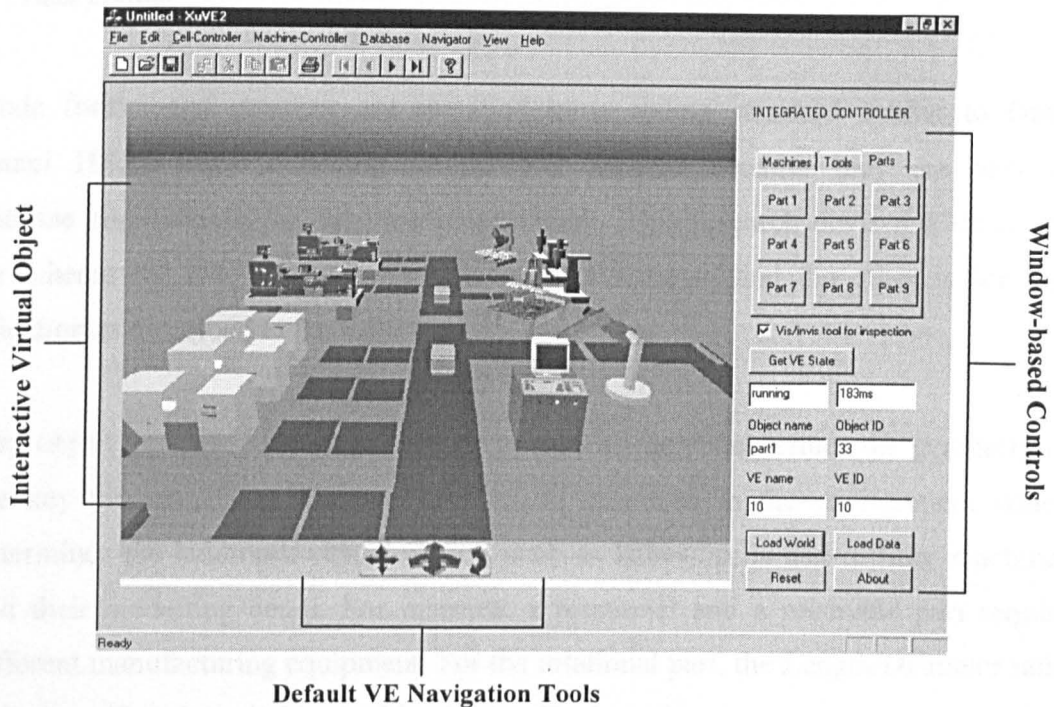


Figure 4.9 Synthetic VE control mode

4.3 KAMVR SYSTEM MANAGER

The KAMVR Virtual Environment Manager connects the virtual environment application interface with the environment database system and the protocols for real manufacturing equipment. It has four separate functional modules.

4.3.1 Task description and coding module

This module is designed as a programming utility for users to define an application into the database and to retrieve template environment for modification. It was designed in three steps. The first step is to design the data format for the task description. The second step is to implement the description with a computer program. The third step determines the procedures of retrieving an environment.

(1) Task coding

The code format and structure are specified by a coding standard similar to Optiz [Rajamani 1993]. Such a coding format was adopted because they are easy to computerise and suitable for database management. This research devised a dedicated coding scheme for two purposes. One is for data retrieval and the other is for task classification as described in the following:

- (i) Key object geometry section: The first section of the code defines the geometry of the key virtual objects that are parts to be produced in the environment which determines the landmark virtual objects such as lathes, mills and drilling machines and their modelling detail. For instance, a rotational and a prismatic part require different manufacturing equipment. For the rotational part, the Length/Diameter ratio will also affect the suitable machine tools. The key object geometry section has five digits G1 to G5 (see Figure 4.10 and Appendix A).
- (ii) Virtual object population section: The number of virtual objects is related to the size, scale, complexity and diversity of the virtual manufacturing system and, for this reason, a secondary coding section defines the population of the virtual objects as shown in Figure 4.10, this section has two digits S1 and S2 (see Figure 4.10 and Appendix A).
- (iii) Application domain type: Application domain type is related to the purpose of a virtual environment and the ways users interact. For instance, if a user attempts to change the geometry and dimensions of a virtual component, as a machine process would, then it must be controlled in a so-called “VR presentation” mechanism. If the user wants to change the geometry or dimensions of the virtual component to only view the changes of prior to post machining, then the virtual component can be more quickly created through an insertion and deletion operation on the memory buffer. KAMVR defines a three-digit code (R1, R2, and R3) to distinguish the type of application (see Figure 4.10 and Appendix A).

Section	Digit	Description of Digit Values	
Machining Part Geometry	G1 (Part Type)	0	Rotational part
		1	Prismatic part
		2	Complex shape
	G2 (Shape)	0	G1 = 0, L/D < 1; G1 = 1, Shape = cube
		1	G1 = 0, 1 < L/D < 15; G1 = 1, Shape = cuboid
		2	G1 = 0, 15 < L/D < 50; G1 = 1, Shape = composite
		3	G1 = 0, L/D > 50; G1 = 1, Shape = triangular
	G3 (Length)	0	Length of the part < 40mm
		1	Length of the part < 80mm
		2	Length of the part < 140mm
		3	Length of the part < 180mm
		4	Length of the part < 200mm
		5	Length of the part < 250mm
		6	Length of the part < 500mm
		7	Length of the part < 800mm
		8	Length of the part < 1000mm
		9	Length of the part < 1300mm
		10(A)	Length of the part < 2300mm
	11(B)	Length of the part < 2900mm	
	G4 (Feature)	0	G1 = 0, null; G1 = 1 or 2, step
		1	G1 = 0, step shaft; G1 = 1, slot
2		G1 = 0, pocket; G1 = 1, through hole	
3		G1 = 0, hole; G1 = 1, blind hole	
G5 (Material)	0	Metal	
	1	Non-metal	
VE Size	S1 (VE Scale)	0	Small scale
		1	Large scale
	S2 (Quantity)	0	S1 = 0, single; S1 = 1, less than 10000
		1	S1 = 0, less than 10; S1 = 1, greater than 10000
Application Domain	R1 (Geo-level)	0	Geometric level low
		1	Geometric level high
	R2 (Sim-level)	0	Simulation level low
		1	Simulation level medium
		2	Simulation level high
	R3 (Inter-mode)	0	Interaction mode: immersive
1		Interaction mode: desktop	

Figure 4.10 Domain-analysis coding scheme

The resulting computer task code is a combination of the three code sections in Figure 4.10. For example, if a key virtual object has the geometry code 13024, virtual object population code 20, and the application domain code 101. The complete task code can then be formed as 1302420101.

(2) Task description program

The task description interface is a 3D-coding panel as shown in Figure 3.4. It has been implemented as an interface that is floating on top of a current VE and serves as an environment built-in control. The pseudo logic code for this interface is shown below.

```

if (Key Object Geometry equals to rotational)
{
  environment primary object digit E2 generated

  if (Virtual Environment Size equal to batch)
  {
    primary object quantity digit E0 generated
    secondary object type and object quantities digit E3 generated

    if (Application Domain Type for simulation)
    {
      Environment primary function type digit E1 generated
      Environment simulation detail level digit E4 generated
    }
    else (Application Domain Type for others)
    {
      Same as above
    }
  }
  else (Virtual Environment Size equal to others)
  {
    Same as above
  }
}
else if (Key Object Geometry equal to others)

```

```

{
  Same as above
}
classification (meta-code)
{
  load template environment list
  get the closest code
  start the best suitable environment
}

```

The above pseudo logic makes use of an automatic sorting function provided by the database system that allows any number of codes to be classified into families. The task description and coding (Module 1) allows the KAMVR system to (i) classify task codes and form families when the database is set up or updated, and (ii) to receive a given task code and use it as a search index to locate the task family into which this given code calls. This leads to the next step, environment retrieval, described below.

(3) Environment retrieval

Before a virtual environment can be visualised or used, it needs to be retrieved from the database and normally modified based on the given task. This is achieved in the following steps:

- All template virtual environments are pointed to by their representative task codes. So a given task code can be used to search the database for a matching template environment.
- The matching environment file records are used to form an environment schema (see Section 4.4) which contains information on the environment to be retrieved.
- The matched VE and its schema are loaded onto a buffer.
- The loaded VE is modified and changed to suit the given task.

In system operation, those steps are mouse clicks on the Window controls, the environment is automatically loaded on the embedded visualising window.

4.3.2 Knowledge representation and acquisition module

This module (Module 2) in the KAMVR system has three functions: run-time VE and database communication, data acquisition and knowledge representation.

(1) Run-time VE and database communication

Each template environment in the database and its objects communicate with the database when: (i) An event occurred and a message is dispatched from the virtual environment to update a database record. For instance, when a certain condition in the environment is fulfilled or a specific command is issued. (ii) Using the data record retrieved from the database, the virtual environment and its object properties can be changed. For example, the change can be made on-line by binding the virtual objects' properties with the database fields, or made with a delay by using time-sequenced control for application requirements. (iii) Saving the modified virtual environment into the database, or adding or deleting specified objects. Figure 4.11 illustrates the data communication interface.

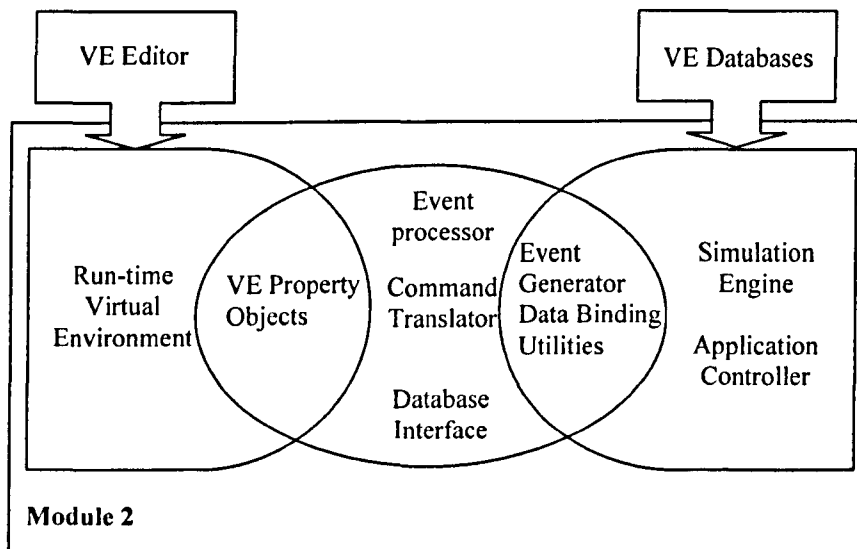


Figure 4.11 Module 2 internal structure

The virtual environment visualising window is implemented in the KAMVR system as an embedded object (see Section 4.2.2) acting as a VE control module interface. This design enables the database record being bound with a virtual environment or virtual objects so that it can be accessed by external application programs. A property of a virtual object is bound with a specific field in a database so that when the users modify this property, the control notifies the database that the property value has been changed and it requests the record field to be updated. The database will then notify the control whether it is a success or failure in response to the request.

This technique is typically used in database visualisation process and provides a visual interface to the state of current database records. It is used by the KAMVR system for VR simulation. For instance, as shown in Figure 4.12, when the user is moving a tool towards a part, the distance between the cutter and the part is constantly checked. When the distance is just about over the collision limit using visual checking, the data are recorded to prevent a collision.

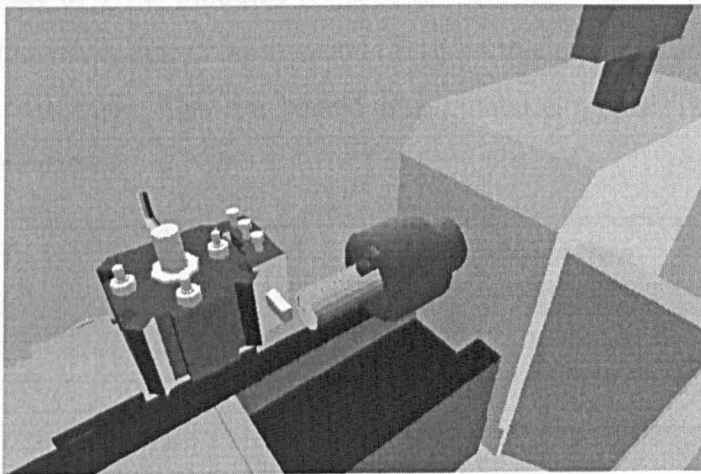


Figure 4.12 Adjusting the position of virtual objects

(2) Data acquisition

The data acquisition functions are developed as individual dynamic-linked-library (DLLs). Users can decide (at system run-time) which set of modules need to be loaded.

These DLLs help reduce unnecessary virtual environment overload and memory requirement.

Figure 4.13 illustrates the implementation of DLLs. Two methods were used in the development:

- (i) Register a new SCL function: The SCL program uses the value returned by the registered function to alter a VE by captured data
- (ii) Direct manipulating the virtual environment data: Used for items that can not be directly accessed by SCL, or for time-related functions. The API vector table contains a set of pointers to all acquired data in VRT (such as world, shapes, and palette data), so they can be located as specific items and be altered directly. This is more closely linked to a particular world than the use of a SCL.

When saving a new template virtual environment into the database as a template environment, a set of DLL modules (SaveStan, SaveStat, SaveDyn and SaveShp) are used to extract the environment information (detailed in chapter 6) and save it in a series of data files. Later these files are parsed into database tables. The DLLs library is presented in Appendix B.

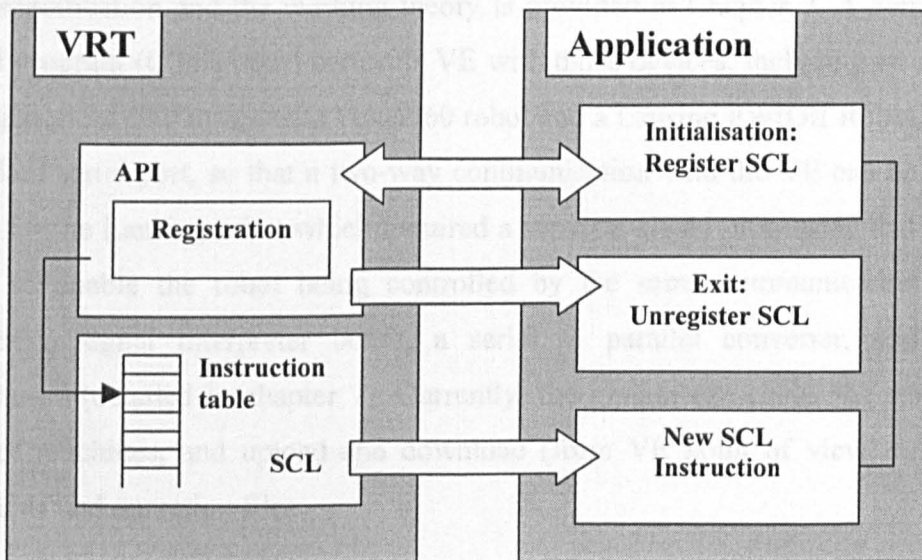


Figure 4.13 Superscape VRT API structure (Courtesy of Superscape Co.Ltd.)

(3) Knowledge representation

The knowledge dealt with by the KAMVR system has been arranged and handled in two levels, the 'self-oriented simulation' and the 'Social' behaviours. The former is programmed within the virtual environment, and coupled with individual objects, groups and controls. For instance, activate the "Start" button on the robot controller to start the robot arm movement. The latter is programmed for advanced simulation activities. In this research the work achieved so far is a virtual environment logic controller that can be manipulated automatically by an application program or manually by users.

4.3.3 Device communication and control module

Module 3 links the virtual environment and objects with their physical counterparts. The data exchange uses the serial port (RS 232). For handling multi-devices, in this research, a multi-port serial card "PCL 844" was used, which provided eight serial ports. The communication does not use handshaking signals, instead it uses a "Question – Answer" mode with time intervals set by the computer CPU clock.

Figure 4.14 shows the connection between the system and the physical devices. The detail specification and the working theory is provided in Chapter 7. A communication control program (C language) connects VE with those devices, including an Audit lathe, a Bridgeport milling machine, a Puma560 robot and a Lansing PWIOII Robot. Most have a standard serial port, so that a two-way communication with the VE can be developed, except for the Lansing robot which required a separate signal processing and controlling device to enable the robot being controlled by the same communication format. It included a signal interpreter board, a serial to parallel converter, and a control switchboard (detailed in chapter 7). Currently, the system can check the start and stop states of machines, and upload and download (from VE point of view) robot control commands and operation files.

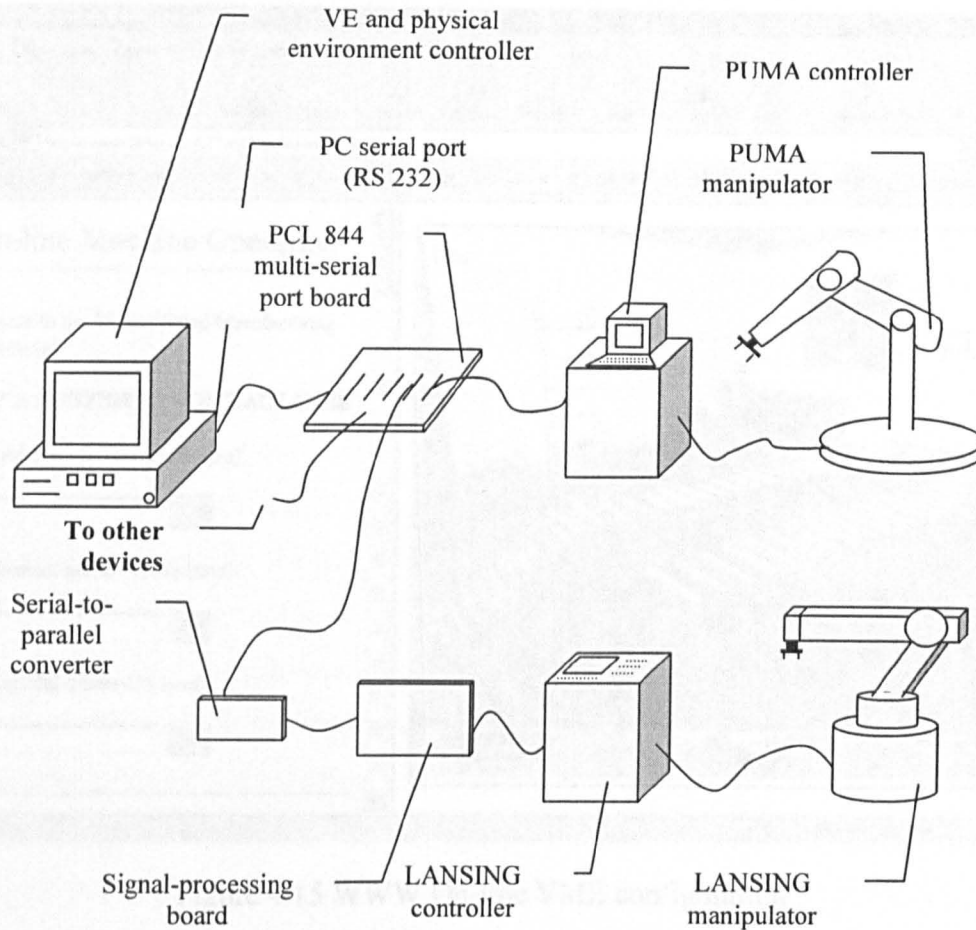


Figure 4.14 Device communication

4.3.4 Network and data communication module

Module 4 allows VME networking in two levels, local area networking (LAN) and wide area networking (WAN). The LAN tackles the communication to a real world computer-integrated-manufacturing system through, for instance, the Ethernet. The WAN deals with the distribution of a virtual environment on the global network, such as the Internet, to allow VE to be accessed, configured and shared by users in geographically dispersed locations. The experimental Website developed for this purpose in the research allows a user to navigate a VE, customise the environment and the object specification, and interactively control a virtual machine. Figure 4.15 shows a snapshot of the Internet-based virtual manufacturing environment.

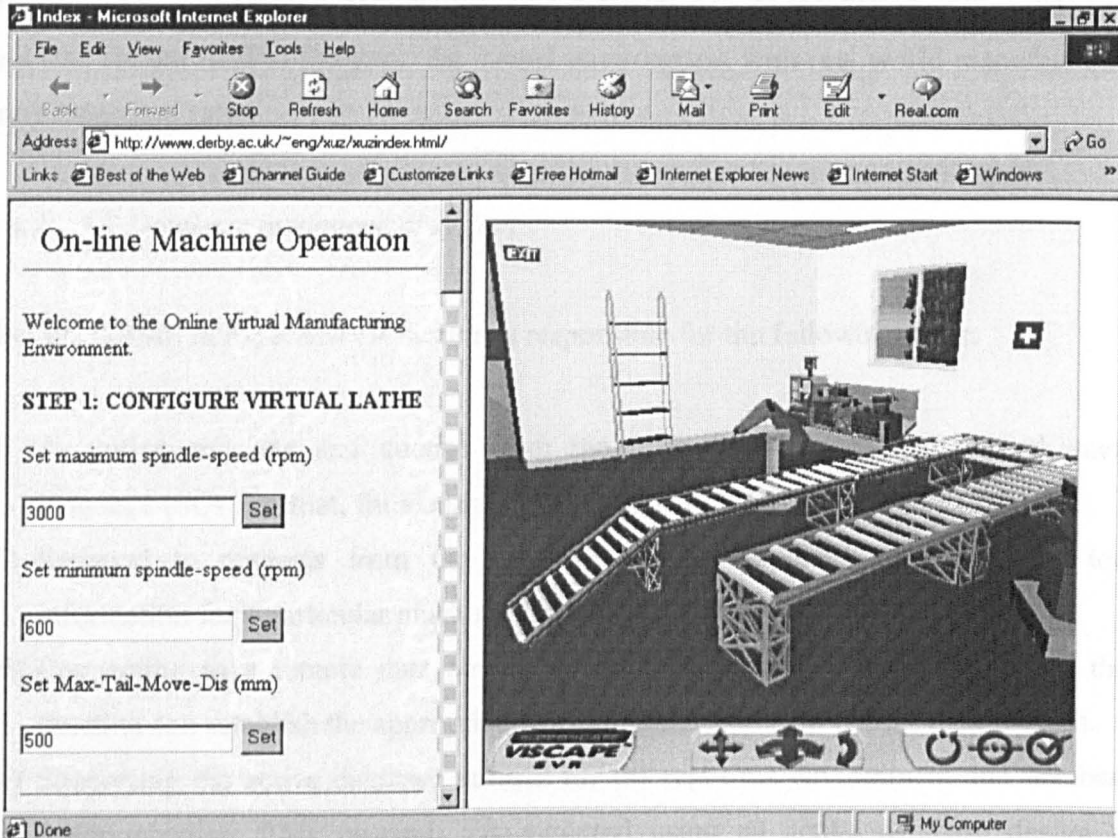


Figure 4.15 WWW On-line VME configuration

4.4 VIRTUAL ENVIRONMENT DATABASE

To deal with the environment information and manufacturing knowledge arranged in a layered structure, a relational database model was designed, and a commercial relational modal based database management system (DBMS) Microsoft Access is adopted.

4.4.1 Data files

In the database design, every virtual environment is represented by its scene graph and virtual object property data, and stored in various database files. The database schema file contains a range of code sections, each formed by various environment data that describe the static and dynamic characteristics of the landmark virtual objects, its detail level and simulation functions. Each of the schema variables forms a database field, and every

virtual object represents a data record. The database can be integrated with a machine, tool and fixture data to integrate the virtual environment with real world manufacturing knowledge and rules.

4.4.2 *VE Database management system*

The VE DBMS in the KAMVR system is responsible for the following tasks:

- (i) Accepting requests and queries from the users in a standard structured query language (SQL) format, for example, "SELECT-FROM-WHERE".
- (ii) Respond to requests from the environment, for instance, retrieving the tool information for a particular machine in the environment.
- (iii) Connecting to a remote data source. Working with the "System Manager", this function can establish the appropriate network connections to remote data sources.
- (iv) Supporting the active database utilities for the real-time environment and database communication. This research has explored using an active database design to support the virtual environment event based database updating. An important part of an active database system is a language for the expression of event – condition – action rules (ECA rules) that defines the active behaviour of an application (Dayal 1988). The event in an ECA rule specifies an operation or situation to be monitored, such as modifying a data value. In time-critical applications, events can also be specified as timing constraints. When an event occurs, the condition is then evaluated. If the condition is true, the action is triggered automatically by the database system.
- (v) Performing the routine database management functions such as creating tables and deleting columns.

4.5 REAL APPLICATION ENVIRONMENT

As a test manufacturing environment, a Puma 560 robot, a Lansing robot, an Audit lathe, and a Bridgeport milling machine have been created in a virtual environment and their

functions simulated. The communication and control implementation will be described in detail in chapter 8. Figure 4.16 (a) and (b) show the real and virtual robot cell layout.

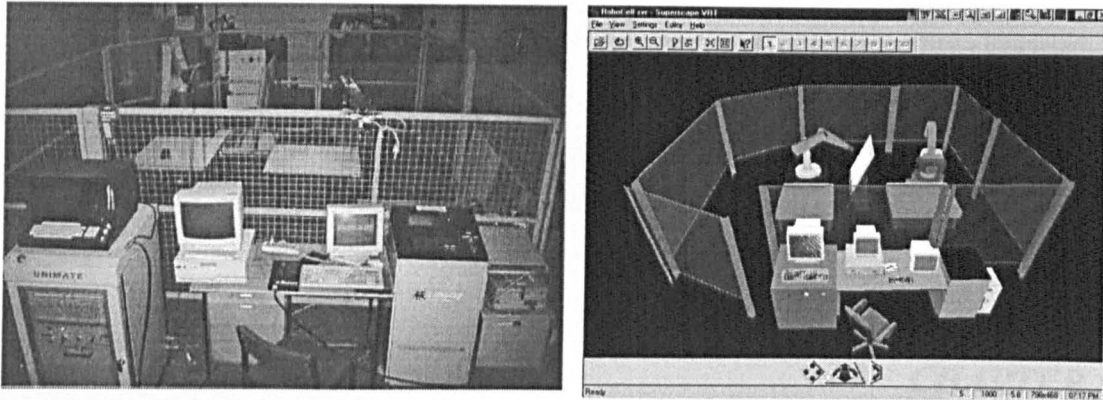


Figure 4.16 (a) Real robot cell (Photo)

(b) Virtual robot cell

4.6 CONCLUSIONS

The KAMVR system architecture allows an environment for a specified task to be rapidly constructed through the application-domain analysis (task coding) and template based environment construction. The constructed virtual environment in this system can be modified and configured using the data from a database, and users can interact with the system through direct manipulation of the virtual environment and its inside objects with or without VR peripheral devices. Users can also interact with Windows based dialog boxes with editable interfacing objects such as a radio button, combo box, and list box. The system communicates with the real world through a standard serial communication port and can be connect with both local and global networks. The KAMVR system has brought the potential of virtual environment reusability and eased the effort in constructing large-scale complex virtual environment and simulation developments. Chapter 5 to chapter 7 explains how the system was researched and developed in more detail, while chapter 8 reports on the validation of the system.

CHAPTER 5

TEMPLATE ENVIRONMENT CONSTRUCTION AND ANALYSIS

The work described in this chapter focuses on the construction of template virtual environments. The content and contexts of an environment are classified as objects, simulation, interaction, knowledge and environment used by most commercial VR systems. The classification is based on common environment modelling procedures. The processes of modelling representative manufacturing machinery such as lathes, milling machines, and robots are explained to explicitly illustrate the object geometric construction and simulation design. Environments generated using templates are also presented. The environment-based knowledge and acquisition representations are investigated with an emphasis on the knowledge type, knowledge source, and accessing knowledge.

5.1 VIRTUAL ENVIRONMENT MODELLING

In the KAMVR System, the template environments were constructed in a one-off process, and stored in the database. To build a specific environment, the user needs only to retrieve and modify a suitable template environment, thus avoiding the need to create an environment from scratch. The creation of a template environment is the work of the system developers rather than the end user. To create those template environments five modelling processes are involved as follows.

5.1.1 *Virtual object modelling*

A virtual object is a set of geometric data, and in most cases, a group of basic shape elements that represents a real world object, i.e., a lathe or a robot. Virtual objects are the building blocks for constructing a large-scale and complex virtual environment. Virtual objects modelling can be done using a CAD system or VR authorising system. If a CAD system is used, an exchange of data is required to the KAMVR database based on one of the exchange standard interfaces like DXF, IGES, STEPS or VRML. At present, KAMVR only support the VRML format. The exchange tools were provided by Superscape VRT.

5.1.2 *Virtual template environment modelling*

The KAMVR database treats a virtual template environment as a collection of associated virtual objects. Those objects are associated in such a way that each possesses specific spatial and animation links with other virtual objects, user controls, communication ports, and physical objects. The template environments were created using the World Editor of Superscape VRT, by associating the virtual objects obtained partially from VRML exchange models and partially created using the shape editor within the Superscape VRT system. The texture, light, sound and other image resources encapsulated within a virtual template environment were imported from various media creating sources such as image editors, sound samplers and video clips.

5.1.3 State simulation modelling

State simulation was used to simulate agents, simulate virtual events and virtual activities within a virtual environment. It also generates behaviour and actions of virtual objects and responds to user interaction and control from physical devices. A simple example is a virtual lathe that is activated by pressing its "start" button or stopped by switching off its virtual power supply. State Simulation functions provided by the KAMVR System come from the simulation kernel of the Superscape VRT system. The mode of a specific simulation state in a template virtual environment within the KAMVR database is a scene graph with user controls. This graph is retained in the database as part of a template environment. Figure 5.1 shows an example of an environment simulation state transition model, it could be two machines in an environment, for example, a lathe sends an event requesting a part to be mounted, a robot receives the event and responds with demand activities. The diagram could also represent active parts of a machine (i.e. an action of one can cause actions of others).

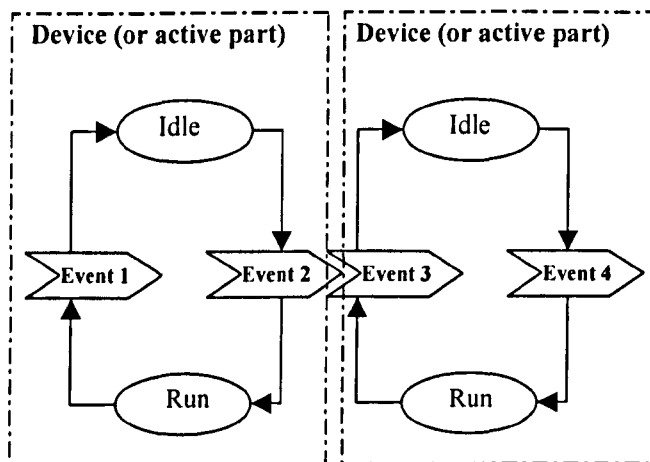


Figure 5.1 VE state transition model

5.1.4 Interaction modelling

The KAMVR system provides all template virtual environments with two interaction models: (i) application-environment interaction, and (ii) user environment interaction. The first model is generated and maintained by modules 3 and 4 (see Figure 4.1), hosting the interface and controls to and from physical devices and systems. The

second model is generated and managed by modules 1 and 2 (see Figure 4.1), providing interface and controls to and from the users. These two interaction models formed part of the interface of a template environment.

5.1.5 Knowledge capture

Knowledge can be captured and stored using knowledge bases and databases. An alternative is to train a neural network. Zhao [1998] proposed that a large-scale and complex virtual environment could be built as a VR based knowledge repository. The benefits of using such a digital environment as a knowledge base can be highlighted as follows:

- (i) Knowledge representation is intuitive and natural and does not rely on knowledge formalisation such as logic, rules and formulae.
- (ii) The users' involvement (immersive) in the environment eliminates the difficulties in dealing with experience based knowledge.
- (iii) Full users' interaction with 3D realism. Users can change and update the knowledge by reconfiguring the virtual environment.

For each of the aforementioned five modelling elements used in the template virtual environment construction, the following sections explain, with examples, use within the research.

5.2 VIRTUAL OBJECT MODELLING

A virtual object is the basic construction block of a complex virtual environment, for example, a virtual lathe or virtual robot. Virtual objects were modelled in a tree structure to construct a virtual environment. In this way, the data structure of each virtual object is similar, especially the geometric data. Other information such as colour, lighting and animation is often rendered around the geometric data. A vector table, which is a reference index to the internal data structure of the Superscape VRT system (see section 7.2) was used to provide pointers to the different object data sections so that an application can access the data of a virtual object directly.

5.2.1 *Virtual lathe model*

A virtual lathe was developed as a virtual object to simulate a real lathe. It included a bed, head-stock assembly, tail-stock assembly, carriage assembly, quick-change gearbox, lead-screw, feed rod, spindle speed selector, feed selector, clutch, cross slide, compound rest, tool post, spindle and chuck. Figure 5.2 shows the virtual lathe model structure.

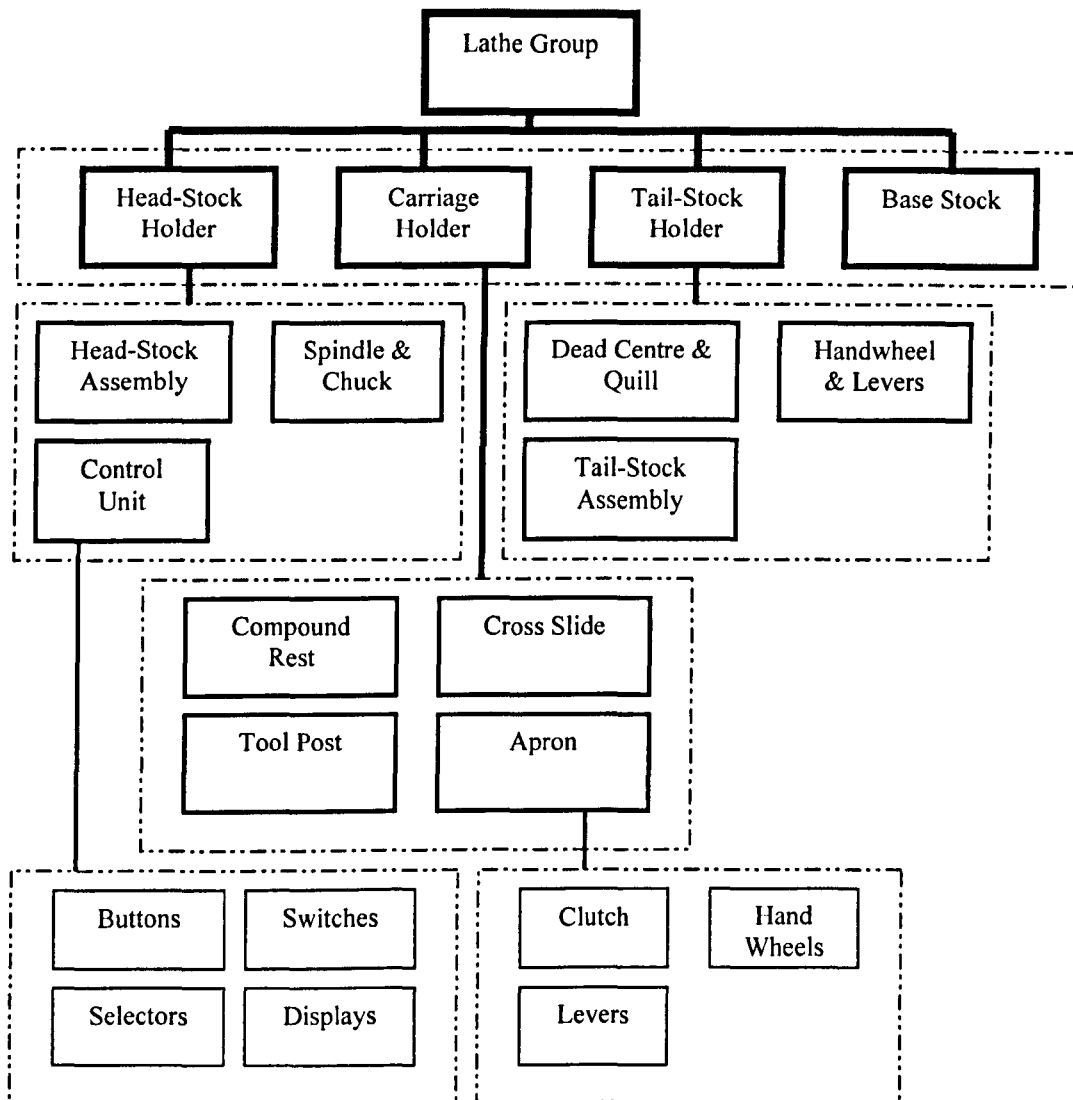


Figure 5.2 Modelling structure of a virtual lathe

The dynamic controls built in this virtual lathe are as follows:

- Spindle speed selector: This control is situated at the front of the Head-Stock. A left mouse click on the disc will rotate the disc anti-clockwise through 36 degrees (one position), whilst a right mouse click will rotate the disc one position in the opposite direction. The speed setting is displayed on the control panel screen and its value is passed to the spindle object when the virtual lathe starts operating.
- Switches and buttons: The control panel and the machine head stock have switches and buttons to facilitate setting up the virtual lathe. Each has two or more animation positions. For a two-state switch, a left mouse click toggles between the two positions. If three or more positions are modelled, the left mouse click will select a lower position, whilst a right click will select a higher position.
- Apron levers/Tail-Stock levers: The lever controls on the Apron and Tail-Stock have two positions to lock/unlock certain parts. A mouse click toggles between states.
- Tool post and saddle assembly control: Saddle assembly, top-slide and cross-slide are moved by hand-wheels. To rotate clockwise, position the mouse pointer over the specific hand-wheel, then press and hold down the right mouse button to start activities. Movement can be stopped by releasing the mouse button. The left mouse button was used to control the anti-clockwise rotation.
- Tail-Stock movement: The Tail-Stock can be moved backwards and forwards along the lathe bed by positioning the mouse pointer, over the main body of the Tail-Stock, and pressing and holding down the appropriate mouse button until the Tail-Stock has moved a required distance. The clamp lever must be set to its 'OFF' position to enable the Tail-Stock to be moved.
- Tail-Stock barrel: This is extended and retracted using the rotating hand-wheel located at the back of the Tail-Stock (through mouse operations). The barrel clamp lever must be set into the "OFF" position before the Tail-Stock barrel can be extended, a message reminder is designed to remind the user of the state of the barrel.
- Power lever: The power lever can be operated by the mouse pointer. Sound effects are explored to indicate the machine 'running' and 'idle' states. Figure 5.3 shows the constructed virtual lathe.

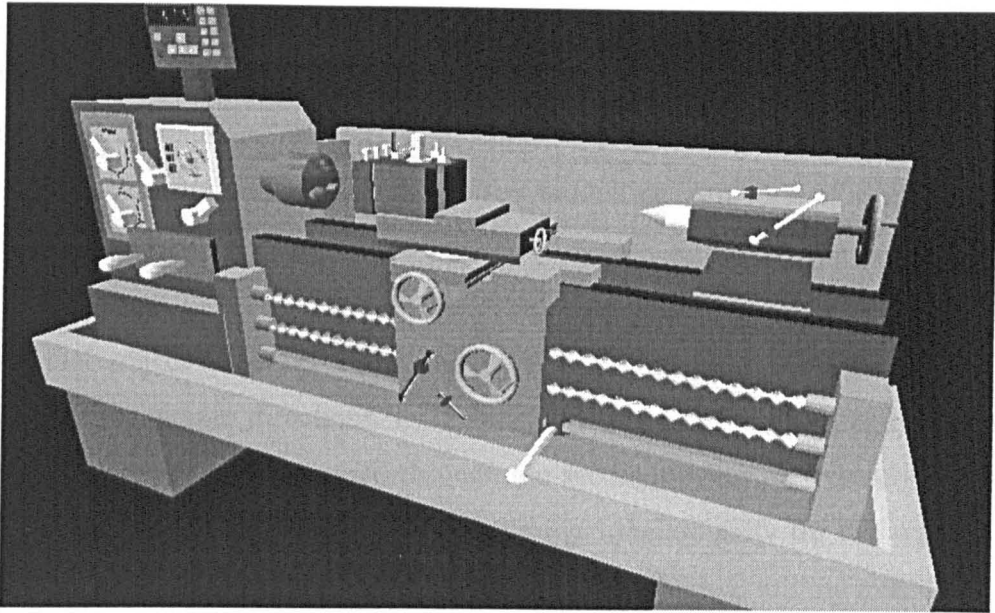


Figure 5.3 Snapshot of the virtual lathe

5.2.2 Virtual milling machine model

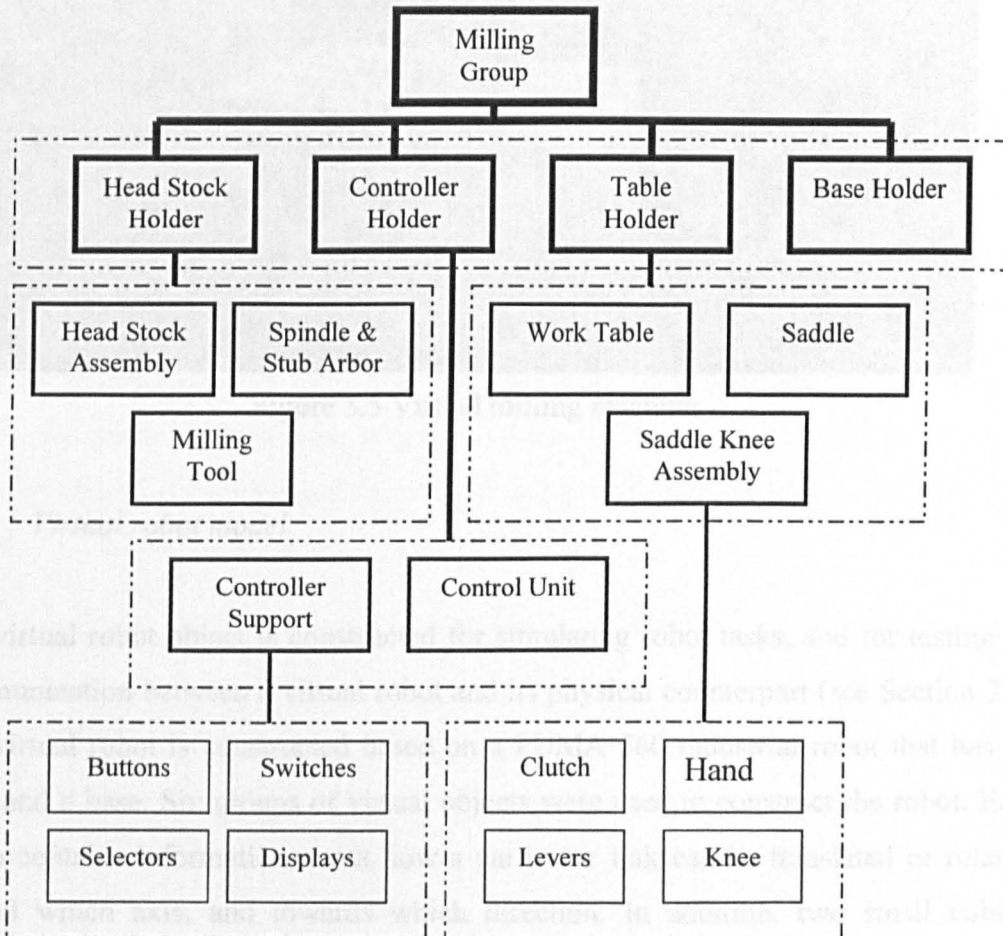


Figure 5.4 Modelling structure of a virtual milling machine

Similar to the processes involved in constructing the virtual lathe, a virtual milling machine was also developed as a virtual object. The development starts with the design of the structure of the milling machine, and then assigns dynamic features to relevant components. Figure 5.4 shows the modelling structure of a conventional column-and-knee-milling machine.

This virtual object provides controlled motion to the virtual worktable in three mutually perpendicular directions. (i) Through the knee moving vertically along the way at the front of the column, (ii) through the saddle moving transversely along the way on the knee, (iii) through the table moving longitudinally on the way on the saddle. Figure 5.5 shows the snapshot of the constructed virtual milling machine.

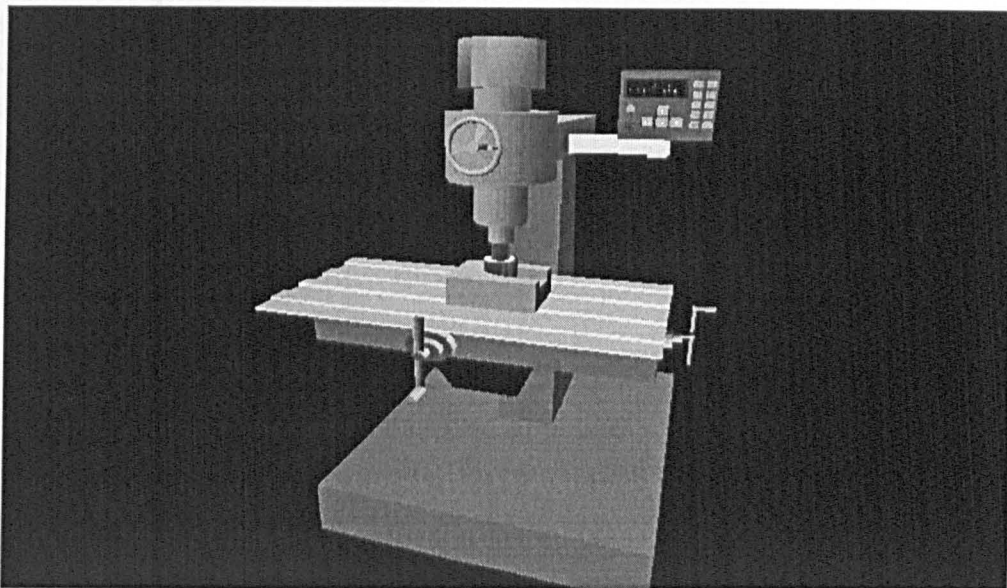


Figure 5.5 Virtual milling machine

5.2.3 Virtual robot model

The virtual robot object is constructed for simulating robot tasks, and for testing the communication between a virtual robot and its physical counterpart (see Section 7.6). The virtual robot is constructed based on a PUMA 560 industrial robot that has six links and a base. Six groups of virtual objects were used to construct the robot. Each group contains information about how a particular link can be translated or rotated, around which axis, and towards which direction. In addition, two small cubical objects are used to simulate the robot gripper, which is located at the top of the robot

upper arm. The gripper has two states - open and closed. Figure 5.6 shows the modelling structure of the virtual robot.

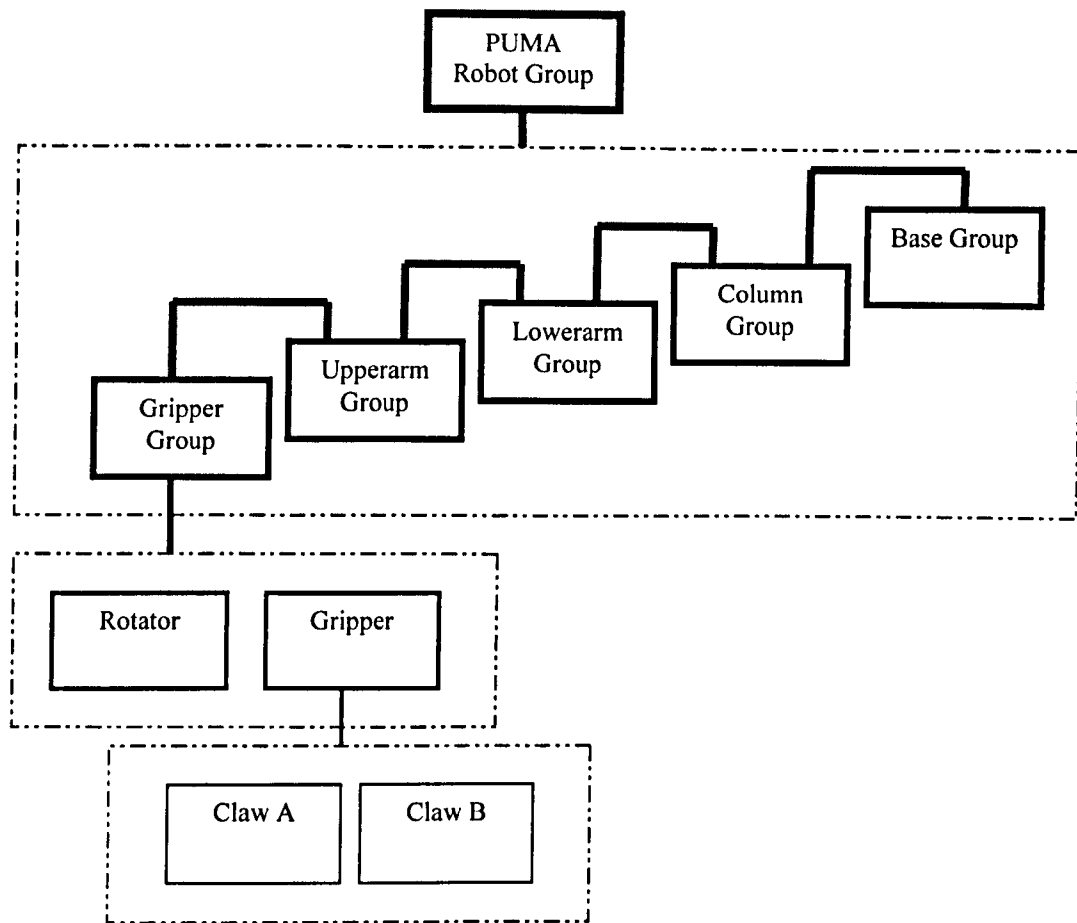


Figure 5.6 Modelling structure of the virtual robot

The robot can be controlled using a VRT dialog-based command interface. This interface allows the user to enter specific commands. The robot can also be controlled through an application program interface, which allows easy control of any of the six joints by selecting the joint and specifying an angle of rotation. The constructed robot model is showed in Figure 5.7.

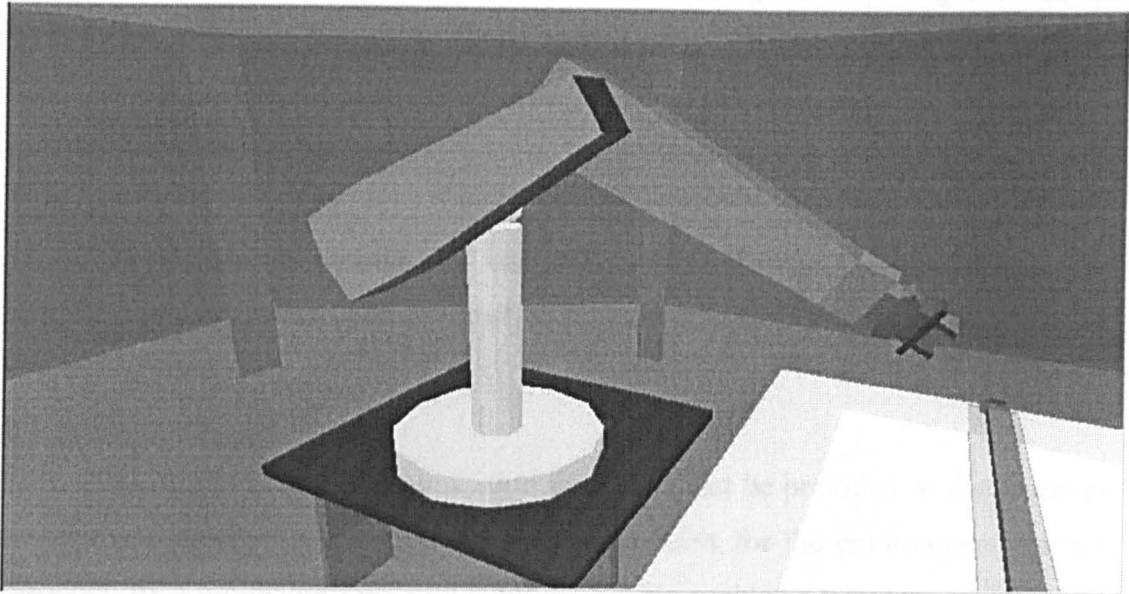


Figure 5.7 A snapshot of the virtual robot

5.3 MODELLING TEMPLATE ENVIRONMENTS

5.3.1 *The modelling criteria*

Template environments built in this research have adopted certain guidelines, (i) the virtual objects must be representative and informative, (ii) the virtual environments should be re-configurable by users, (iii) application information must be able to be visualised, (iv) external data sources can be connected to the environments and information exchange functions should be provided. For meeting these guidelines, the template environments were modelled with the following specified criteria.

- (1) A balance must be kept between the environment detail level and the overall environment size, this was found to be crucial. Individually, every virtual object inside a virtual environment will influence the realism of the virtual world. However, too much detail on to unimportant objects increases rendering time and reduces the rendering speed. Only essential foreground should be modelled in detail, other background objects should be defined as rough polygons only.

(2) Virtual object(s) must be prioritised according to their significance of environment representation in a virtual environment. The most significant objects should be used as landmarks to distinguish the virtual environments, and to be used as searching keys for the database.

(3) The simulation controls and interactions (described in Section 5.4 and 5.5) between users and virtual objects must be defined specifically before the modelling process is started.

(4) The run-time data communication interface must be provided so that users at run-time can modify an environment, and the controls for the environment are not restrained by a pre-determined simulation task. This enables a template environment to be used for a series of similar applications. Chapter 7 describes the run-time platform for environment implementation.

5.3.2 The construction of template environments

Five fully functional template environments were built in the research and saved in the database (see Chapter 6). Each of the templates can perform a series of similar manufacturing tasks. Table 5.1 shows the typical features of these template environments, described by their primary resources, secondly resources, and the tasks they are capable of performing.

VME Name	Primary Resources	Secondly Resources	Job Type	Simulation Control	Main Task
Robot Cell	Puma robot	Lansing robot	Robot control object handling	Rotation and object picking	Control simulation
Lathe Cell	Lathe	Lathe	Turning processes	Rotation and interaction	Rotational parts
Milling Cell	Miller	Rotator table	Facing	Cutter rotation and movement	Non-rot parts Facing
Job Cell	Lathe and miller	Robot and conveyor	Machining and transportation	Aggregate of individuals	Layout management
Large-scale Workshop	Transportation belt	Other machines	Layout design visualisation	Environment navigation	Layout and walk-through

Table 5.1 Features of the virtual template environments

Each template has distinctive features as follows:

(1) **Robot Cell:** This virtual template environment includes two robots and three control computers. By clicking on the virtual robot start button, each robot can adjust its arm position and return to the “Home” position. Each of the robot components can be controlled to perform certain actions and the “Stop” button on the control unit terminates all robot movement. Figure 5.8 shows the robot cell.

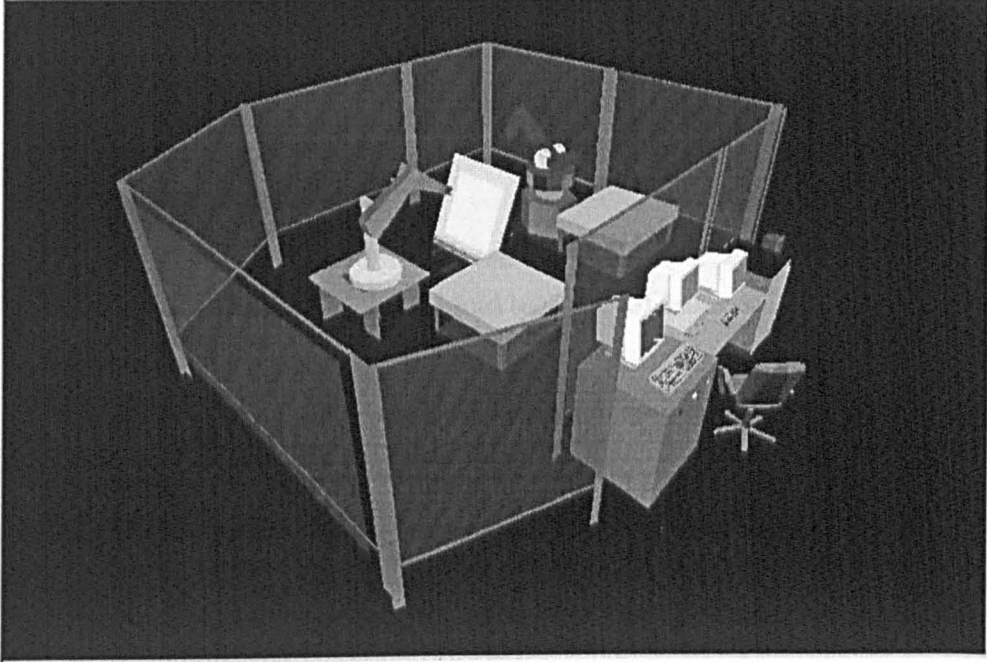


Figure 5.8 Virtual template of robot cells

(2) **Lathe Cell:** This template environment includes two virtual lathe objects. Depending on the application, they can be set with similar or different machining capability. As explained in Section 5.2.2, the virtual lathe object has functions similar to its physical counterpart and can be controlled using its virtual parts such as levers and hand-wheels or by using a virtual control panel. Figure 5.9 shows the lathe cell.

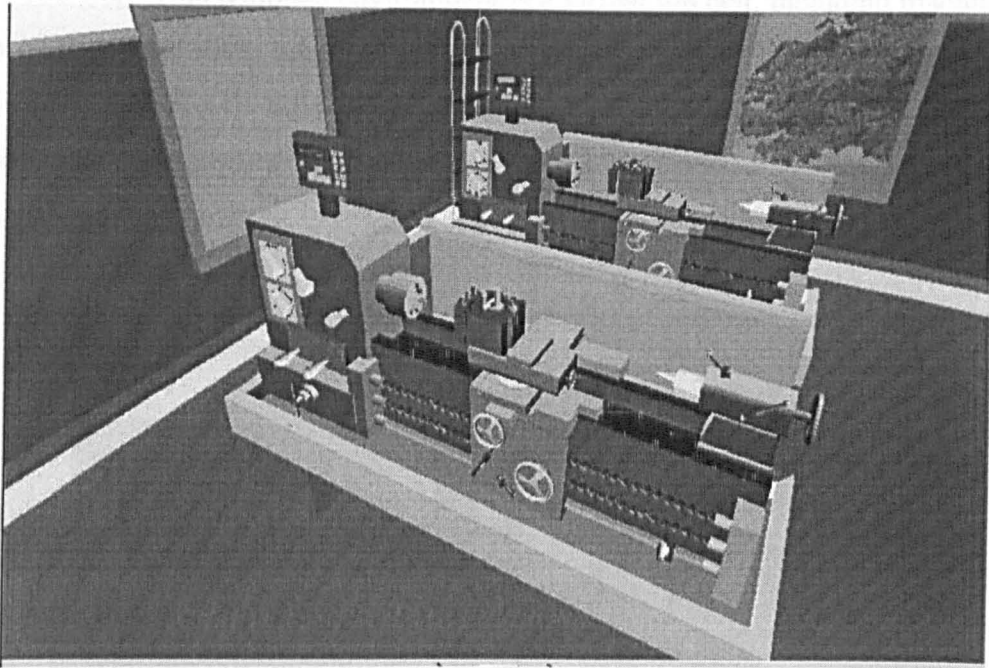


Figure 5.9 Virtual template of lathe cell

(3) **Milling Cell:** The milling cell template has a milling machine, a lathe, and a rotating table. The milling machine can perform true milling operations. The rotating table is used for transporting parts and is controlled by a mouse or keyboard. By clicking “Up”, “Down”, “Left”, and “Right” arrows keys, the table can adjust its height, turn to specific directions and transport a part to a pointed machine. Figure 5.10 shows the template milling cell.

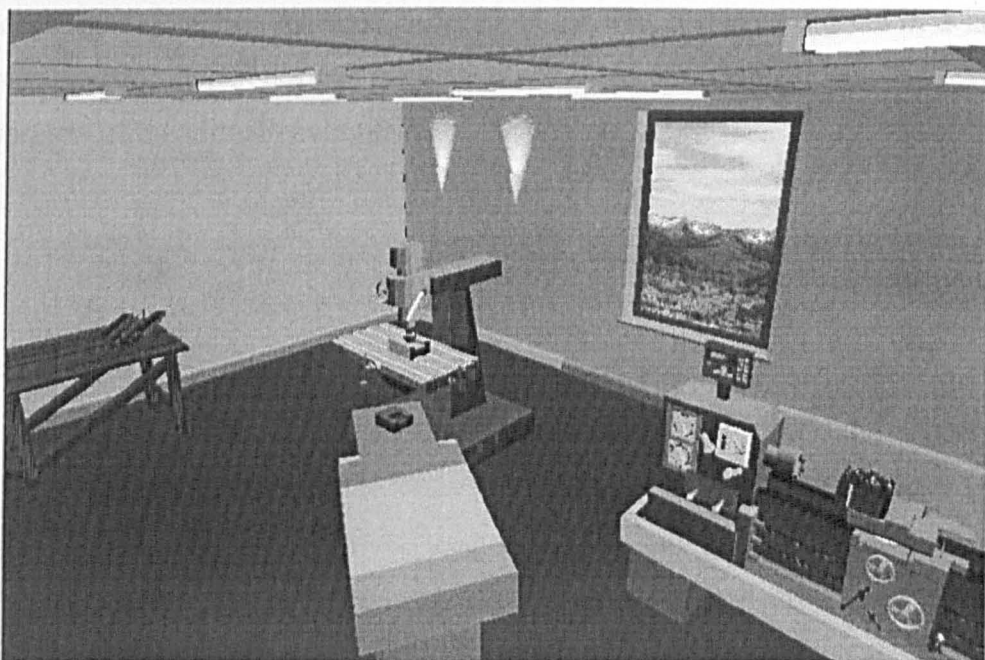


Figure 5.10 Virtual template of the milling cell

(4) Job Cell: This template environment is a virtual job cell, including machining, transportation and stock sites. These sites have their own individual simulation functionality that can be activated by different users and other environment events. They can also be controlled in a co-ordinated sequence. Figure 5.11 shows a snapshot of the template environment.



Figure 5.11 Virtual template of CIMs rooms

(5) Large-scale Workshop Environment: The virtual template environment of large-scale manufacturing systems is designed for user to locate objects in a large-scale virtual environment through navigation and exploration tools. Figure 5.12 shows the overview of the virtual environment.

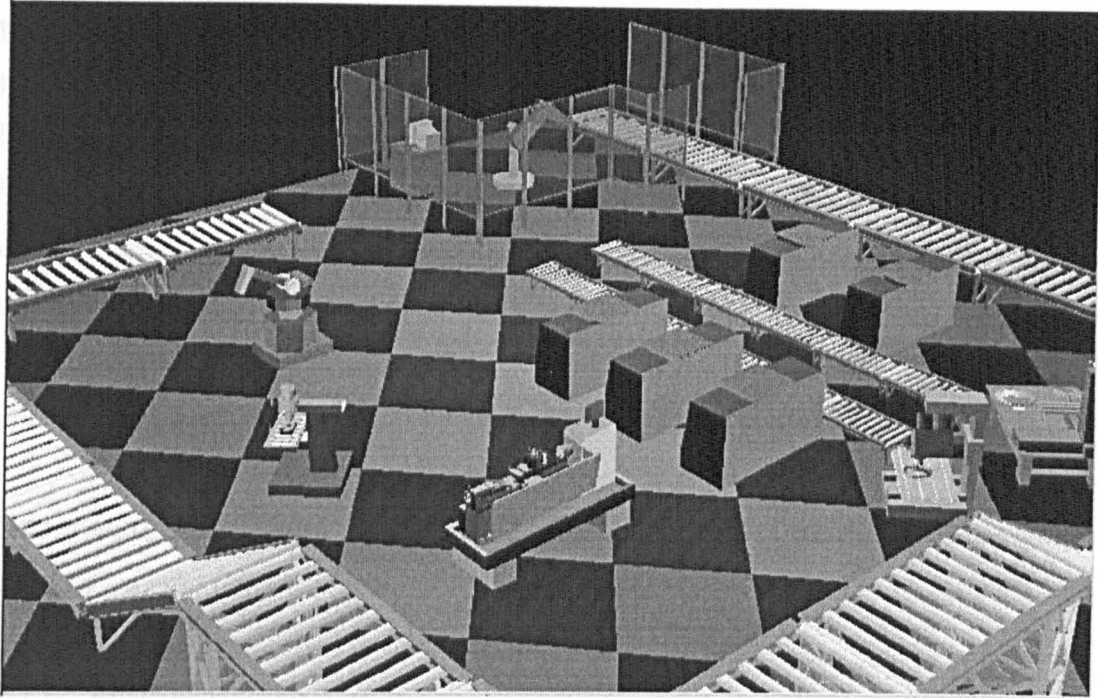


Figure 5.12 Virtual template of large scale virtual manufacturing environment

Five virtual template environments have so far been created in the research. It is apparent that the more template environments that exist in the system database, the less construction work is demanded from the users to construct a particular virtual environment for a given task.

5.4 TEMPLATE ENVIRONMENT SIMULATION

Simulation provided for the above template environments was classified as: (i) self-imposed simulation, such as gravity, restitution, and animation, (ii) functional actions, including movement, rotation, and collision, (iii) interaction actions, for instance, a machine is activated by pressing the “Start” button. An interaction always involves more than one virtual object. A simulation control language was (SCL) used in designing the simulation actions. In SCL, a simulation contains a user-defined function, a pointer to a virtual object with which the simulation task is associated, and a priority value that specifies the order in which the task is executed relative to other tasks as the simulation runs. The simulation coding is mainly enforced on the virtual objects rather than virtual environments. Figures 5.13 and 5.14 show the simulation actions for the lathe and milling machine objects. The simulation loops in both cases

(Figure 5.13 and Figure 5.14) follow a pre-defined sequence, starting from mounting a part on the worktable, and then setting machining parameters using the 3D control panel. The machines can be started by pressing the start button or by lifting the power lever, and then the machining simulation mimics the real machining activities. Operations can be recorded in a data file and stored in the database. When the operations are finished the part can be unloaded and the machines may be switched off or wait for reloading.

5.5 INTERACTING WITH A VIRTUAL TEMPLATE ENVIRONMENT

User-environment interactions were investigated for virtual environment navigation, exploration, and object control.

5.5.1 *Environment navigation*

Environment navigation is to dynamically view a virtual environment and its inside objects through multi-viewpoints or defined navigation routes. It enables users to view a virtual model from physically an “impractical” distance and angle. In this work, environment navigation is achieved through bringing user to the virtual objects, attaching a viewpoint on moving objects to check, say part alignment with tools and fixtures, and to eliminate potential collisions. The dynamic navigation routes are also used to give users an overview of the virtual environment and help them become familiar with the environment layouts. A virtual environment navigation path stores a series of positions and orientations in world co-ordinates. These routes can be used to guide the viewpoint or move to other entries in the environment. Routes can be recorded, edited, saved, loaded, and played back. For instance, when attaching a viewpoint on to a cutting tool, the recorded route of a simulation loop will represent the tool path. Navigation routes are made up of a set of discrete data elements, each element storing an absolute position and orientation. A file is created when a route is saved and this file can be edited as a simple ASCII file if users want to make a change within a text editor.

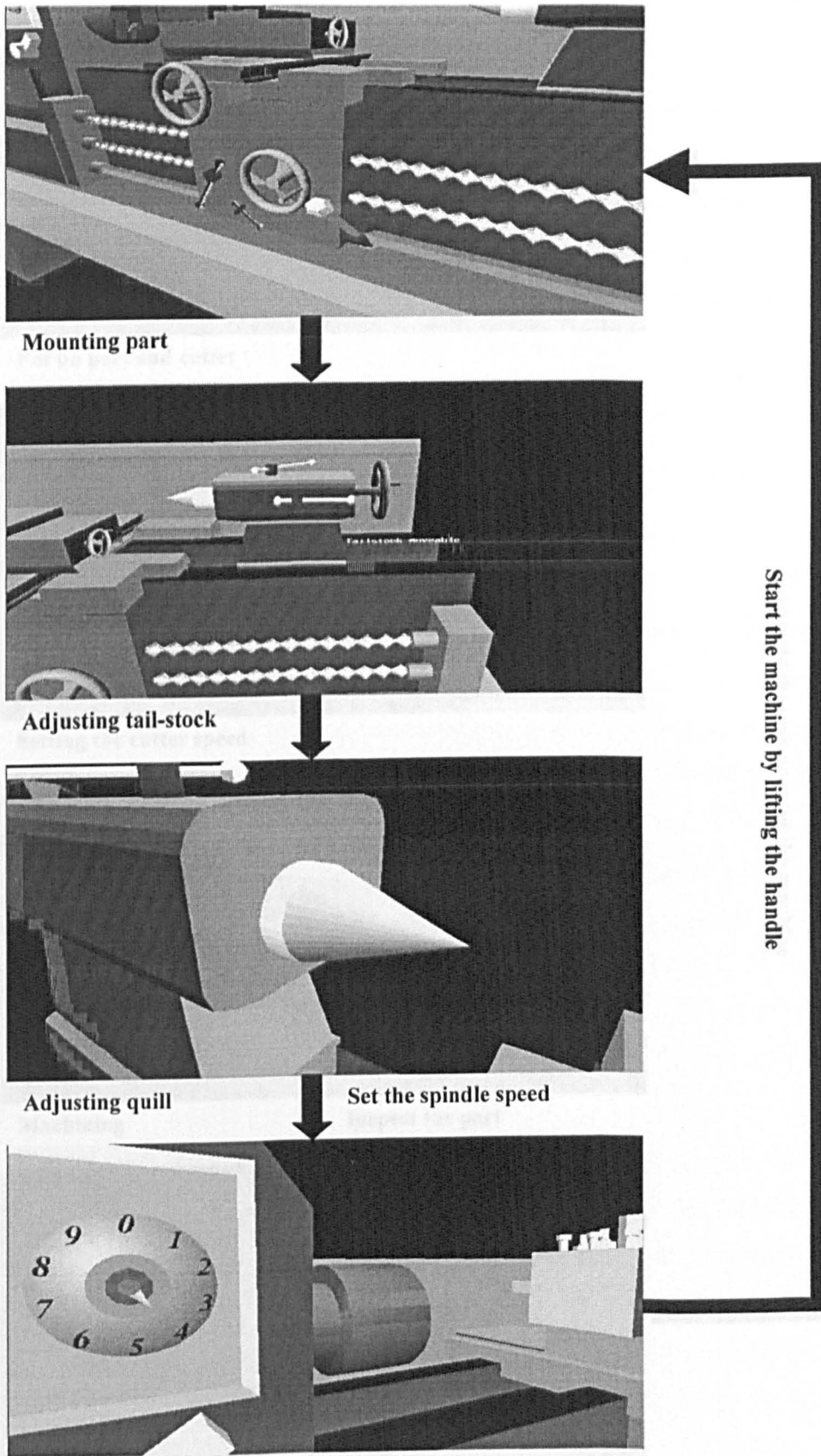


Figure 5.13 The simulation loop imposed on the virtual lathe objects

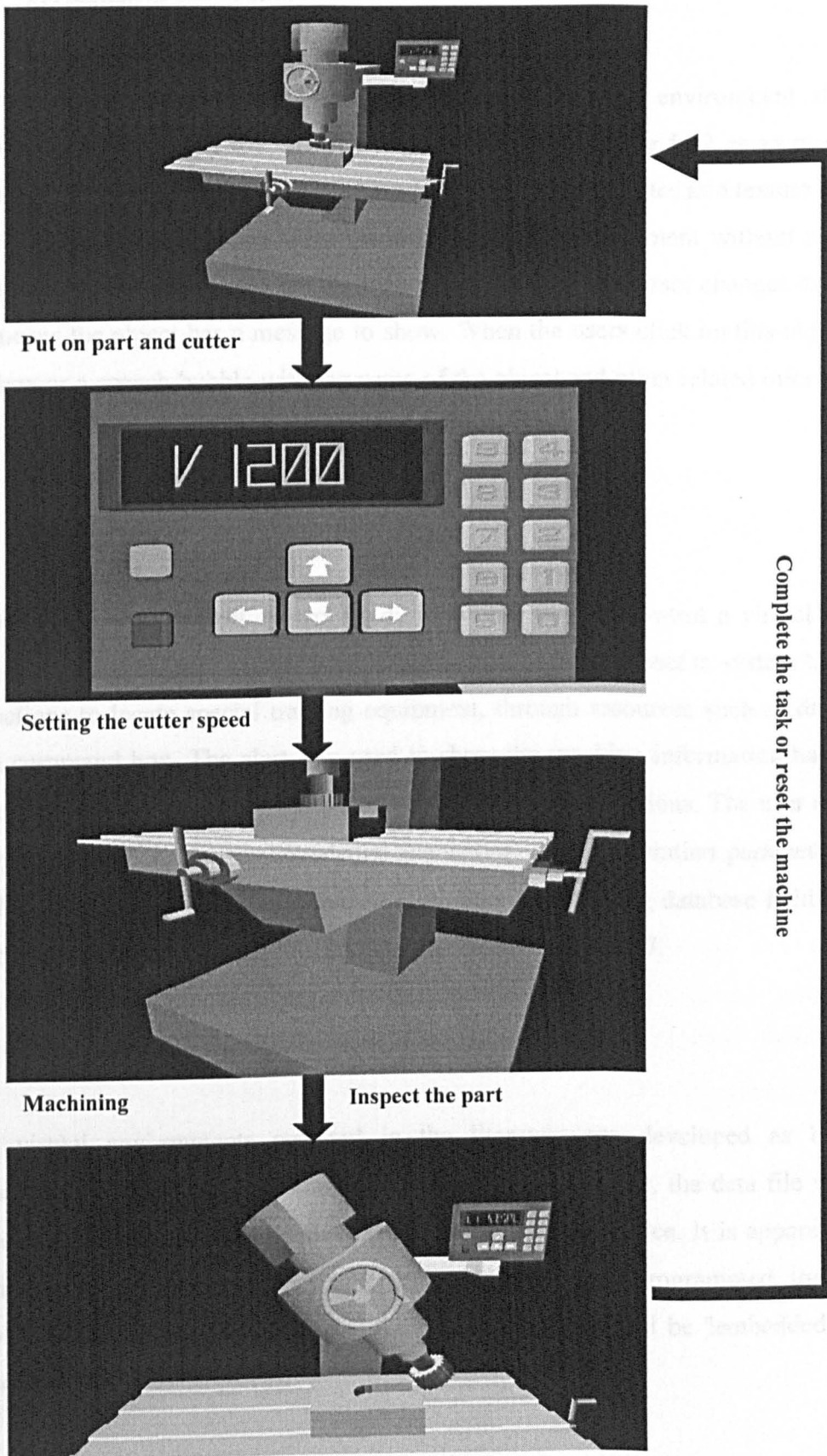


Figure5.14 The simulation loop imposed on the virtual miller objects

5.5.2 *Environment exploration*

Environment exploration helps users understand a virtual environment design, functions and use. Using the virtual workshop shown in Figure 5.12 as an example, the virtual machines in the workshop have their own identity plates as a texture image, which users can move closer to and see the plates. For equipment without such an identity plate, users can move the mouse cursor over it, if the cursor changes its shape that means the object has a message to show. When the users click on this object, an alert box or a speech bubble with the name of the object and other related information will pop up.

5.5.3 *Object control*

A user in the environment control mode may also want to control a virtual object using experience or 'trial and error'. For instance, a user may react to system training instructions to locate special training equipment, through resources such as dialogue and a command box. The alert box used to show the machine information has been linked to another detailed control dialog to support these operations. The user control information is important for knowledge acquisition, so the operation parameters are saved in a buffer and can be exchanged with the corresponding database fields. This type of control and data communication is described in Chapter 7.

5.6 KNOWLEDGE SOURCES AND CAPTURE

Most virtual environments reported in the literature are developed as built-in subroutines and stored as a geometry data file. At the run-time, the data file will be loaded into the memory and display on a computer output device. It is apparent that the more environment simulation actions are considered and programmed, the more complex the environment will be and the more knowledge will be "embedded" into this environment.

To use a simple virtual environment in representing a wider range of simulation situation or knowledge without using excessive hard-coded program routines and

complex data files, a virtual environment should be provided with two basic functions.

- (i) A data communication with the physical environment so that it can be logged on to knowledge sources (such as a normal manufacturing database, a spreadsheet and a text document) and extracts the knowledge from those sources and use it to reconfigure its own contents (see section 7).
- (ii) Defining the captured data into its internal format (SCL) and interpret the data into environment object attributes. This function is to represent the physical data and knowledge into a visualised format as follows.

5.6.1 Data interpretation

Data interpretation deals with the translation of data, from external sources, into a data format that the environment can understand, and then apply those data in updating the environment and object attributes. The environment data can also be converted back into external data compatible to the external data source.

This work has considered two types of external knowledge sources. One is the database that is designed in Microsoft Access format. The database stores the basic information about machine tools and robots. This information has also been used in creating the template manufacturing environment. The other external knowledge source is the physical robots from which operations and control data are transmitted directly into a virtual environment. The data from both types of knowledge sources is interpreted into primitive data blocks to define the elementary simulation action such as translation, rotation, scaling and shearing of a virtual object. The format of the interpreted data complies with the environment modelling data format (See Section 6.2 and 6.6).

5.6.2 Knowledge representation

After the data from an external data source has been interpreted into basic simulation elements, it needs to be synthesised into higher level actions (more meaningful information) such as starting or stopping a lathe, loading or unloading a component

and detecting a robot arm collision. This work regards this data synthesis as knowledge representation. Currently this work can only synthesise the data from the database sources. Due to the real-time limitation of data communication between a virtual environment and a physical one, the data from a physical source has not been able to be synthesised into useful knowledge. This remains a challenge for future investigation. At present, the work can interpret data from a physical robot and send it to the database.

There are two simple techniques that were devised for knowledge syntheses, (i) a SCL based data link designed using Dynamic Data Exchange (DDE) and Microsoft's Active X functions, (ii) a " data binding" program that assigns the external source data to the template environments as environment properties. The implementation of these two techniques aims to achieve two aspects of knowledge representation. One is for data completeness and the other is data synchronisation. Data completeness requires a full dual-communication between the virtual environment and the database. Data synchronisation requires that every change made in either the virtual environment or the database will be reflected and changed accordingly at the other end. These processes are described in more detail in Chapter 7.

5.7 CONCLUSIONS

The construction of template environments is a development process carried out by system developers, not by the users. Once a considerable number of templates are in place, they can be used by the users to rapidly build their own complex environments.

CHAPTER 6

**DATA MANAGEMENT WITHIN VIRTUAL
ENVIRONMENTS**

This chapter presents a data structure for the virtual environment database in the KAMVR system. Rather than saving each of the environments as an individual database file, a novel data structure was developed to enable the database to store only an index number of each environment using a 'General Reference' table. The retrieval of a virtual environment relies on the index to load related information from the General Reference and other environment property tables. The chapter starts with the description of the environment structure, and shows how it is used in constructing template environments. The database design is explained and its merits discussed. Finally, relationships between the database files and manufacturing information based on the structure are explored.

6.1 OVERVIEW OF THE ENVIRONMENT DATA

In KAMVR, data stored and managed by the database falls into two main categories - environment data that are used in constructing VEs, and physical manufacturing data that used for controlling and operating environments. For environment data, the database stores only the essential information of the template environments sufficient to enable users to construct their own environments. For physical manufacturing data, the database stores the information for helping users to establish controls about machines, tools, and simulation actions in their applications.

6.2 THE HIERARCHY OF ENVIRONMENT DATA

A template virtual environment in the KAMVR system is compressed into a basic tree structure - called a scene graph - before being saved in the database. All rendering information such as colour, texture, lighting, animation and simulation data are formatted and stored as data records in different data files. The amount of support for retrieving environment scene graphs, and assembling the rendering data around the graph depends on the robustness of the database.

In computing terms, the virtual environment is held in the database is done so by assembling virtual objects into a hierarchical scene graph and then controlling them according to the transformation nodes. The scene graph dictates how the environment is rendered on the screen. For instance, users can create a light source, and specify its location in the scene graph such that it only affects the objects under its branch of the scene graph. The operations being imposed on the scene graph can determine:

- Whether or not a polygon surface should be rendered in culling mode.
- Whether or not a Z-buffer should be used.
- How a virtual object is transformed in 3D co-ordinates.
- The level of detail that the data should be presented.
- How an object is grouped.
- And, how the environment is controlled.

For a better understanding of the concept of the scene graph and providing a foundation for work reported on data acquisition and management (introduced in later sections), imagine a virtual environment as a 3-D space filled with empty transparent cubes, each cube can be filled by geometric and property data. These cubes have unique ID numbers and any two of them are inter-related through a link chain forming the relationships such as parent, child and sibling.

Figure 6.1(a) shows one simple scene graph with seven objects, where Object 1 is the parent object of Object 2, 3 and 4, Object 5 and Object 6 are siblings and have the same parent – Object 3. In terms of meaningful virtual objects, this environment is composed by a single object group 'TailGroup' (extracted from the Lathe machine model described in Section 5.2.1). There are five objects in the 'TailGroup', two levers, the tailstock, the deadcenter, and the handwheel. In addition the environment has a default 'RootObject' (Object 0), which defines the virtual universe, and is the parent for all other objects in the environment. Figure 6.1(b) shows a snapshot of the virtual environment.

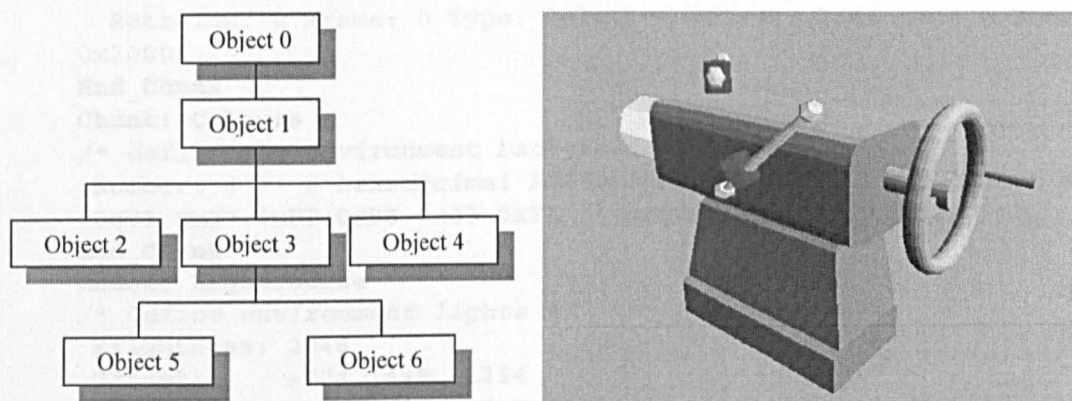


Figure 6.1 (a) Virtual environment scene graph (b) A snapshot of the environment

The scene graph of the virtual environment is then populated with other rendering data describing object appearances and dynamics. The complete virtual environment scene graph (shown in Figure 6.1(a)) with its related data to be loaded and displayed are listed in List 6.1(a) and List 6.1(b):

```

/* Object 0, Define the virtual universe */
Chunk: Standard
/* Mandatory data section for all objects */
Name:      "RootObject" /* Name of the object */
  
```



```

Number:    0 /*Automaticalluy assigned object number */
Size:      2147483647 2147483647 2147483647
Position:  0 0 0
Type:      65535
Layer:     0
DFlags:    E
End_Chunk /* End of a data section */
Chunk: ViewPoint
/* Define the observation viewpoints */
Number:    100 /* Maximum 100 viewpoints */
Subchunk /* Define each of the viewpoint */
Number:    1
Type:      35 /* define viewpoint type such as fly or walk */
View:      0
Point:     0
Frames:    1
Position:  0 Frame: 0 Type: StraightMove Pos: 2024377 26064
1995529
Rotation:  0 Frame: 0 Type: Relative Offset: 4000 6304 0
Zoom:0x2000
Subchunk /* more viewpoints definitions */
Number:    2
Type:      35
View:      0
Point:     0
Frames:    1
Position:  0 Frame: 0 Type: StraightMove Pos: 2028478 23760
1991656
Rotation:  0 Frame: 0 Type: Relative Offset: 2944 5808 0 Zoom:
0x2000
End_Chunk
Chunk: Colours
/* define the environment background colour */
Number:    6 /* 6 hexadecimal RGB values, three for one colour */
0xD5 0x07 0xFF 0xB8 0x03 0xFF /* ground and sky colour */
End_Chunk
Chunk: LightSource
/* Define environment lights */
Brightness: 2048
Offset:     -724 1448 -1254
Rotation:   0x0000 0x0000 0x0000
BeamWidth:  0
Dispersion: 200
Colour:     0xFF 0xFF 0xFF
BeamEdge:   0
Flags:      p0o
End_Chunk

```

List 6.1(a) Environment definition in the scene graph script

List 6.1(a) shows the 'RootObject' definition uses the format of a data chunk (marked by the token 'Chunk'), which is a data section that defines a specific type of property. Each object can have a single (mandatory Standard chunk) or multiple chunks. Inside

each of the chunks, there are number of detailed data fields for further defining the property of a virtual object. For example, in the 'Standard' data section, virtual object name (text data type), object number (integer data type), size and position (integer data type) are declared, and in the 'Colours' data section, the environment background colours are defined. For each individual colour used in the environment, three hexadecimal numbers (range from 0 to 255, or 0x00 to 0xFF) are used to represent the original colour – red, blue, and green (RGB).

The difference between the 'RootObject' and other virtual objects in an environment is that the 'RootObject' is actually the virtual environment, which defines the size of the VE boundary and sets other environmental parameters such as viewpoint number and positions. The VE background colour and lighting are also defined in the 'RootObject'.

A similar format has been used in defining other objects in the environment positioned under the 'RootObject' in the scene graph tree by using the token 'Children'. The 'Children' token is also used to define further parent-child relationships among these objects. List 6.1(b) shows other objects of the environment in the scene graph script.

```

Children:
/* Object 1, the TailGroup */
Chunk: Standard
Name:      "TailGroup"
Number:    1
Size:      9989 5739 3427
Position:  2013787 18794 2002813
Type:      65535
Layer:     0
DFlags:    rE
End_Chunk
Chunk: Rotations
Initial:   0x0000 0x0000 0x0000
Centre:    4787 2870 1691
End_Chunk
Chunk: InitPos
Position:  2012816 18794 2002813
End_Chunk

Children:
/* Object 2, the deadcenter */

Chunk: Standard

```

```

//Defintion for the deadcenter object
End_Chunk
Chunk: InitPos
//Initial position of the object
End_Chunk

/* Object 4, the handwheel */
Chunk: Standard
//definition for the handwheel object
End_Chunk
Chunk: Rotations
//Rotational data
End_Chunk
Chunk: SCL
short  mx, my;
fixed  rx, ry;

resume (1, 0);
if (activate (me, 0))
{
  mx=mousex;
  my=mousey;

  ry=yrot (parent (me));
  while (mouseb && xpos (#4)>0
        && xpos (#4)<1100 && yrot (#2)==135)
  {
    xrot (me)=ry+mousex-mx;
    waitf;

    xpos (#4)+=ry+mousex-mx;
    waitf;

    if (xpos (#4)<=0 || xpos (#4)>=1100)
      xpos (#4)=ixpos (#4);
  }
  clrtrig (me, 0);
}
end
End_Chunk

/* Object 3, the tailstock */
Chunk: Standard
//Definition of the tailstock object
End_Chunk
Chunk: SCL
short  mx, my;
fixed  rx, ry;

resume (1, 0);
if (activate (me, 0) && zrot (#3)==90)
{
  mx=mousex;
  my=mousey;

  ry=yrot (parent (me));
  while (mouseb)

```

```

    {
      xrot (me)=ry+mousex-mx;
      waitf;

      xpos (#1)+=ry+mousex-mx;
      waitf;
    }
    clrtrig (me, 0);
  }
end
End_Chunk
Chunk: Colours
  Number: 37
  0x2F 0x1D 0x1A 0x1D 0x1D 0x2F 0x2F 0x2F 0x2F 0x2F 0x1D 0x2F
0x26 0x1B 0x25 0x1C
  0x29 0x1D 0x28 0x1F 0x19 0x1A 0x21 0x24 0x29 0x2C 0x2F 0x2F
0x29 0x19 0x1E 0x21
  0x26 0x29 0x2F 0x2F 0x26
End_Chunk

Children:
/* Object 5, Lever 1 */
Chunk: Standard
//Definition of the lever 1
End_Chunk
Chunk: InitPos
//Initial position data
End_Chunk
Chunk: SCL
resume (0, 1);
if (activate (me, 13))
  yrot (#2)=135;
waitf;

if (activate (me, 0))
  yrot (#2)=180;
waitf;
end
End_Chunk
Chunk: Rotations
//Rotational data
End_Chunk

/* Object 6, Lever 2 */
Chunk: Standard
//Definition of Lever 2
End_Chunk
Chunk: Bubble
//Speech Bubble definition to display control message
End_Chunk
Chunk: SCL
resume (0, 1);
if (activate (me, 0) && zrot (#3)<85)
{
  while (mouseb)
  {
    zrot (#3)+=45;
  }
}

```

```

    waitf;

    sptext (me)="Tailstock moveable";
    waitf;
  }
}

if (activate (me, 13) && zrot (#3)>50)
{

  while (mouseb)
  {
    zrot (#3)--=45;
    waitf;

    sptext (me)="Tailstock locked";
    waitf;
  }
}
end
End_Chunk
Chunk: Colours
//define applied colour numbers and values
End_Chunk
Chunk: LitCols
//define initial colours
End_Chunk
Chunk: Rotations
//Rotational data
End_Chunk
End_Children
End_Children
End_Children
End_Children
End_File

```

List 6.1(b) Object definitions

List 6.1(b) shows that some objects are wrapped by a group object. Because a group object has its own distinctive co-ordinating system that differs to the virtual environment, so every component in this group will have two different data sets to identify its spatial state - one relative to the environment, another relative to the group. The purpose of a transformation is to place and rotate an object in the scene relative to various other co-ordinating systems. Transformations accumulate as you traverse down the scene graph tree. Any data section started with a 'Children' token uses the co-ordinating system of its parent to define the positional and orientation properties of itself. For example, the 'Lever 1' object (which is the immediate child of the 'TailGroup' object) uses the co-ordinating system of 'TailGroup' for its relative position and orientation. So the absolute spatial data of the 'Lever 1' object is its

relative value plus the offset of its parent co-ordinating system from the one in the virtual universe.

Another important data section in this structure is the 'SCL' chunk, which stores the coded simulation procedures in the environment. These simulation procedures need to be programmed by the VE developer at the environment design time, and attached to specific virtual objects with little or no flexibility of reconfiguration. Also, due to the fact that the simulation results acquired from running these coded procedures are only presented in the environment, so the information exchange between the environment-dedicated simulations and application data sources is difficult. A solution to these problems requires a virtual environment and its object data to be organised in a highly-formatted computing structure for easy management. The following sections will concentrate on reporting the creation of such a data management system. More detail on environment reconfiguration and information exchange will be described in Chapter 7.

6.3 THE DEVELOPMENT OF AN ENVIRONMENT DATA STRUCTURE

As shown in the script file, a virtual environment with only a few simple objects can be organised in a long list of code. A practical application for a large-scale and complex virtual manufacturing environment that will have considerably more objects and much more complex properties, which would culminate with thousands of lines of program code that would be difficult to read, query, modify and manage.

6.3.1 *The database of the environment*

A database for storing and managing a virtual manufacturing environment has been designed to overcome this problem. It would need the following attributes:

- All of the template virtual environment can be stored in the database accurately and completely
- Given a virtual environment index number, all objects belonging to that environment and their properties can be retrieved

- Given a virtual object number, the environment or environments to which it belongs to can be identified.
- A virtual environment can be modified by altering database records, adding or deleting objects.

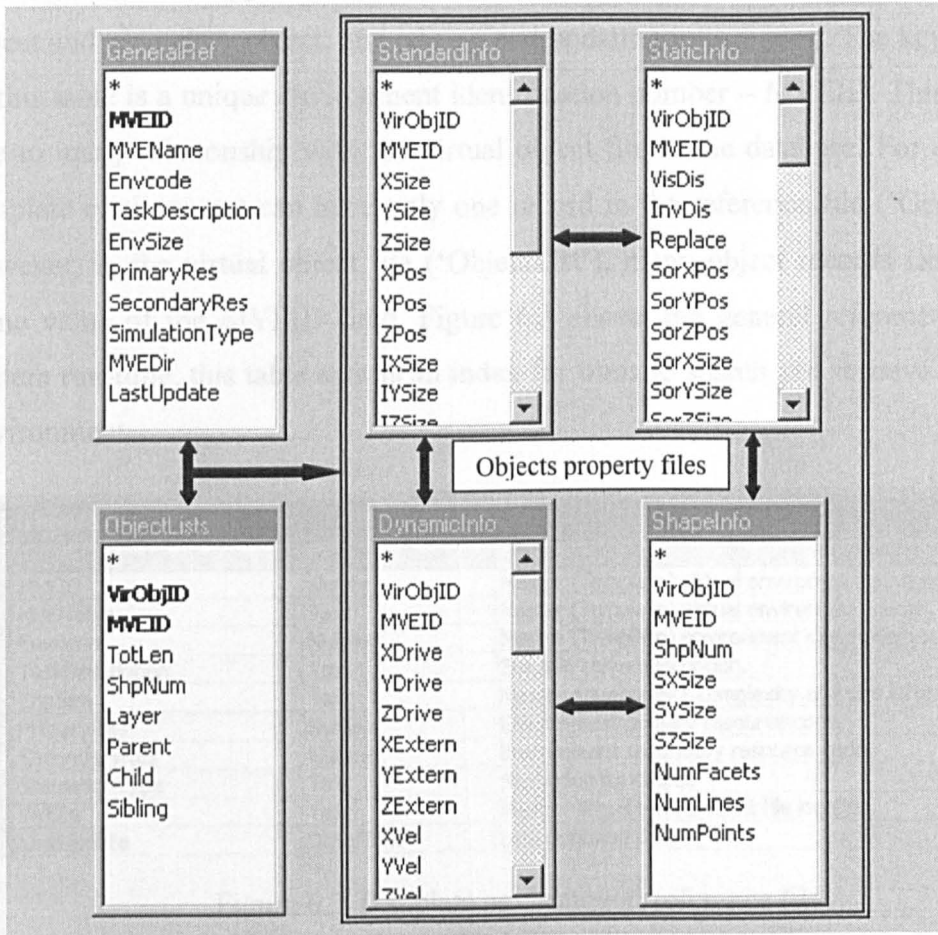
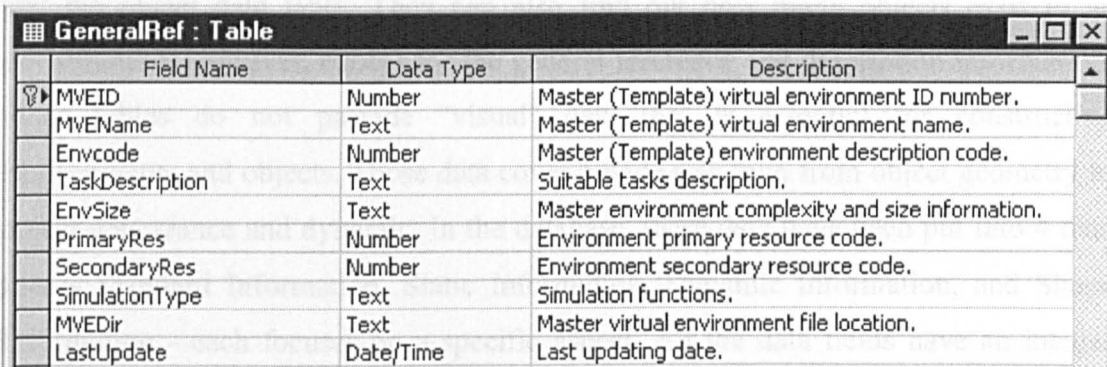


Figure 6.2 KAMVR system database structure

Figure 6.2 shows the structure of such a database. It has six data files - General Reference, Object List, Standard Information, Dynamic Information, Static Information, and Shape Information – and is organised in a relational database format. Data fields MVEID (which refers to template virtual environment identification number) and VirObjID (which refers to virtual object identification number) are used as key data fields for setting the data relationships in the database. Each data file is explained in the following.

6.3.2 General template environment reference file

This file stores descriptive information of all the constructed template environments, such as environment ID number, environment name, environment code (which refers to the environment application domain introduced in Section 3.4.1 and 4.3.1), primary object and secondary object, and editing and updating information. The key data field in this table is a unique environment identification number – MVEID. This file has a one-to-many relationship with the virtual object file in the database. For example, a template environment can have only one record in the reference file ('GeneralRef'), however, in the virtual object file ('ObjectList'), many object records can have the same value of the MVEID field. Figure 6.3 shows the general reference table. At system run-time, this table acts as an index for users to search and retrieve a template environment.



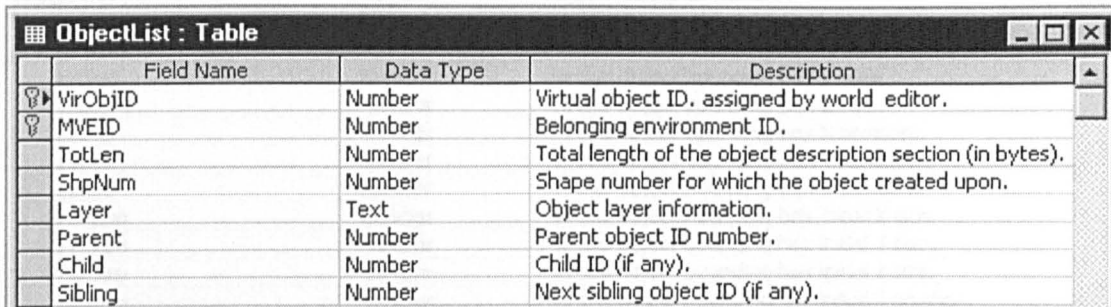
Field Name	Data Type	Description
MVEID	Number	Master (Template) virtual environment ID number.
MVENAME	Text	Master (Template) virtual environment name.
Envcode	Number	Master (Template) environment description code.
TaskDescription	Text	Suitable tasks description.
EnvSize	Text	Master environment complexity and size information.
PrimaryRes	Number	Environment primary resource code.
SecondaryRes	Number	Environment secondary resource code.
SimulationType	Text	Simulation functions.
MVEDir	Text	Master virtual environment file location.
LastUpdate	Date/Time	Last updating date.

Figure 6.3 Template environment reference file

6.3.3 Virtual object file

Many virtual objects are repeatedly used in environment construction. Therefore, it is uneconomic to duplicate an individual copy in the database. The only information vital for keeping a clear track of those objects is (i) in which environment an object properties are recorded, and (ii) the distinctive object ID number in that environment. For the first information, MVEID provides the solution, and for the second, the VirObjID data field does. The combination of these two data fields is sufficient to uniquely identify an object. Figure 6.4 shows the data file schema, where the code length of an object, its basis shape number, the object layer, its relative object

information in the environment scene graph tree are stored in corresponding data fields.



	Field Name	Data Type	Description
?	VirObjID	Number	Virtual object ID, assigned by world editor.
?	MVEID	Number	Belonging environment ID.
	TotLen	Number	Total length of the object description section (in bytes).
	ShpNum	Number	Shape number for which the object created upon.
	Layer	Text	Object layer information.
	Parent	Number	Parent object ID number.
	Child	Number	Child ID (if any).
	Sibling	Number	Next sibling object ID (if any).

Figure 6.4 Virtual object data file

6.3.4 Standard information

A user can identify a specific object from querying the environment reference table and the object data table. They can also find out how many objects exist in an environment. However, except for the general reference and description information, those tables do not provide "visual" data that is essential for constructing environments and objects. Those data cover a wide spectrum from object geometry to object appearance and dynamic. In the database, these data have been put into 4 data files - Standard Information, Static Information, Dynamic Information, and Shape Information - each focuses on a specific aspect. All the data fields have an integer data type in the four data files for the convenience of scanning and storing data from an environment into the database (see detail in Section 6.4).

Standard information refers to data that is essential for the computer to provide and display (render) an environment (scene graph). The standard data are used in defining the scene graph using place-holders (see Section 6.2.1). The standard information table stores the place-holders' spatial information, for instance, initial and current sizes and position, as well as their transformation nodes data (Section 6.2.2), such as rotation and rotation centres. Figure 6.5 shows the design of the standard information file.

StandardInfo : Table			
Field Name	Data Type	Description	
VirObjID	Number	Virtual object ID number.	
MVEID	Number	Master environment ID number.	
XSize	Number	Virtual object bounding box X size.	
YSize	Number	Virtual object bounding box Y size.	
ZSize	Number	Virtual object bounding box Z size.	
XPos	Number	Virtual object bounding box X position.	
YPos	Number	Virtual object bounding box Y position.	
ZPos	Number	Virtual object bounding box Z position.	
IXSize	Number	Virtual object bounding box initial X size.	
IYSize	Number	Virtual object bounding box initial Y size.	
IZSize	Number	Virtual object bounding box initial Z size.	
IXPos	Number	Virtual object bounding box initial X position.	
IYPos	Number	Virtual object bounding box initial Y position.	
IZPos	Number	Virtual object bounding box initial Z position.	
XCenter	Number	Object X center point, vital for object rotation.	
YCenter	Number	Object Y center point, vital for object rotation.	
ZCenter	Number	Object Z center point, vital for object rotation.	
VptNum	Number	Number of viewpoints.	
VptAttached	Number	Object to be attached by the viewpoint.	
VptConObj	Number	Viewpoint controlling object.	

Figure 6.5 Virtual object standard information file

6.3.5 Dynamic information

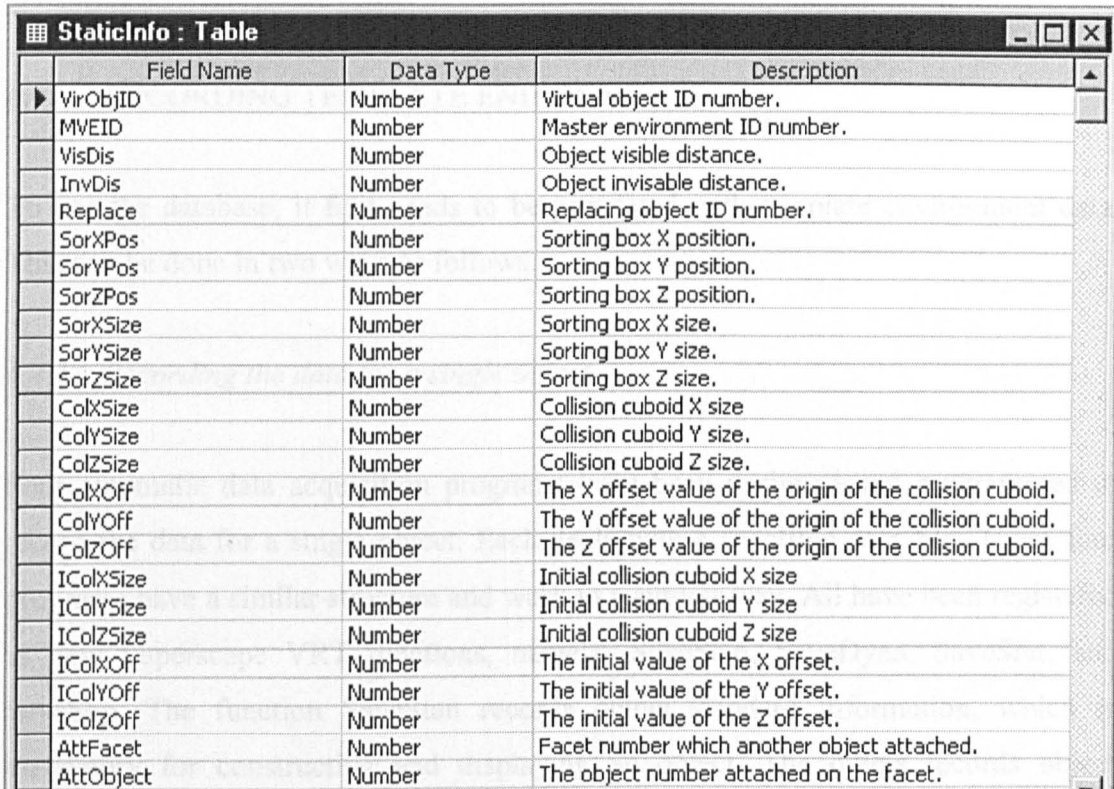
Once virtual object spatial information is defined, its dynamic data will decide its capacity for performing simulation tasks. An object's activities in an environment can be seen as one of three types, scaling, rotation, and translation. They will be affected by factors such as external driving force, gravity, friction, and restitution. The database file that tackles this part of the data is the dynamic data file (see Figure 6.6.)

DynamicInfo : Table			
Field Name	Data Type	Description	
VirObjID	Number	Virtual object ID number.	
MVEID	Number	Master environment ID number.	
XDrive	Number	Object driving force along X (abstracted from the vector).	
YDrive	Number	Object driving force along Y (abstracted from the vector).	
ZDrive	Number	Object driving force along Z (abstracted from the vector).	
XExtern	Number	Object external force along X (abstracted from the vector).	
YExtern	Number	Object external force along Y (abstracted from the vector).	
ZExtern	Number	Object external force along Z (abstracted from the vector).	
XVel	Number	Object velocity along X (abstracted from the vector).	
YVel	Number	Object velocity along Y (abstracted from the vector).	
ZVel	Number	Object velocity along Z (abstracted from the vector).	
XFriction	Number	Object friction along X (abstracted from the vector).	
YFriction	Number	Object friction along Y (abstracted from the vector).	
ZFriction	Number	Object friction along Z (abstracted from the vector).	
XAngv	Number	Object angular velocity along X.	
YAngv	Number	Object angular velocity along Y.	
ZAngv	Number	Object angular velocity along Z.	
XRot	Number	Object rotation along X.	

Figure 6.6 Virtual object dynamic data file

6.3.6 Static information

Static data deal with environment properties such as colour, lighting, texture, and distance, which defines the appearance of an object. The design of the static information table includes fields defining object colour and texture, as well as data fields defining how an object is displayed, such as the sorting box for surface hiding and the distancing replacement. The data file design is shown in Figure 6.7.



Field Name	Data Type	Description
VirObjID	Number	Virtual object ID number.
MVEID	Number	Master environment ID number.
VisDis	Number	Object visible distance.
InvDis	Number	Object invisible distance.
Replace	Number	Replacing object ID number.
SorXPos	Number	Sorting box X position.
SorYPos	Number	Sorting box Y position.
SorZPos	Number	Sorting box Z position.
SorXSize	Number	Sorting box X size.
SorYSize	Number	Sorting box Y size.
SorZSize	Number	Sorting box Z size.
ColXSize	Number	Collision cuboid X size
ColYSize	Number	Collision cuboid Y size.
ColZSize	Number	Collision cuboid Z size.
ColXOff	Number	The X offset value of the origin of the collision cuboid.
ColYOff	Number	The Y offset value of the origin of the collision cuboid.
ColZOff	Number	The Z offset value of the origin of the collision cuboid.
IColXSize	Number	Initial collision cuboid X size
IColYSize	Number	Initial collision cuboid Y size
IColZSize	Number	Initial collision cuboid Z size
IColXOff	Number	The initial value of the X offset.
IColYOff	Number	The initial value of the Y offset.
IColZOff	Number	The initial value of the Z offset.
AttFacet	Number	Facet number which another object attached.
AttObject	Number	The object number attached on the facet.

Figure 6.7 Virtual object static data file

6.3.7 Shape data

So far all the data fields defined from the aforementioned data files act on the place-holding cubical volumes, it is the shape or so-called 3D model data that will eventually fill in those volumes. This research did not undertake a detailed exploration of object geometric data. However, the shape reference data are registered in the KAMVR for object retrieval purposes. It includes base shape number, construction points list, number of lines and facets. Figure 6.8 shows the shape data file.

Field Name	Data Type	Description
VirObjID	Number	Virtual object ID number.
MVEID	Number	Master environment ID number.
ShpNum	Number	Shape reference number.
SXSize	Number	Shape X size.
SYSize	Number	Shape Y size.
SZSize	Number	Shape Z size.
NumFacets	Number	Number of facets constructing the shape.
NumLines	Number	Number of lines of the shape.
NumPoints	Number	Number of the points of the shape.

Figure 6.8 Object shape data file

6.4 RECORDING TEMPLATE ENVIRONMENTS

To use the database, it first needs to be populated with template environment data. This can be done in two ways as follows.

6.4.1 Recording the data for a single object

Four automatic data acquisition programs have been designed and programmed to record the data for a single object. Each deals with a specified data file. These four programs have a similar structure and work in a similar way. All have been registered as new Superscape VRT functions, namely, SaveStan, SaveDyna, SaveStat, and SaveShp. The function SaveStan records object standard information, which is mandatory for constructing and displaying an object. The others records object dynamic, static, and geometric information.

The method of implementing those four programs within Superscape VRT can be described in the following steps:

- (1) Declaring the function in a compiler to register the SaveStan function in the structure. The function record is defined as,

```
static T_COMPILEREC NewSCL=
{
    "SaveStan", //New command name
    0,         //Function code
```

```

0x20,          //2 input, 0 output
0,            //Compiler code
E_PROCEDURE,  //Procedure type
E_SSNONE,     //None return value
E_SSOBJNUM,   //Input object number
E_SSINTEGER, //Input environment ID
};

```

(2) Taking an object ID number from the environment data stack and using it as a function parameter. The user input function parameters are retrieved from a program stack and assigned to data variables. For instance,

```

MVEID=PopN(E_SSINTEGER);    //Get template environment ID
ObjNum=PopN(E_SSOBJNUM);    //Get object number

```

(3) Locating the relevant object property data section and retrieving the memory address. This is to locate the position of each property data section in memory. The following code gives examples of how this is achieved.

```

Object=ChunkAdd(ObjNum,E_CTSTANDARD);    //Standard data section
InitSize=ChunkAdd(ObjNum,E_CTINITSIZE); //Initial object size
InitPos=ChunkAdd(ObjNum,E_CTINITPOS);    //Initial rotation data
Rotation=ChunkAdd(ObjNum,E_CTROTATIONS); //Rotation data section
Viewpoint=ChunkAdd(ObjNum,E_CTVIEWPOINT); //Viewpoint data

```

(4) Accessing the data sections and recording the data into the database. After the individual object property data sections are located and the data addresses are returned, the next step is to access the data stored in the memory sections and record them into a text file. The following is an example.

```

fprintf(f,"%ld\t", Object->Std.XSize);
fprintf(f,"%ld\t", Object->Std.YSize);
fprintf(f,"%ld\t", Object->Std.ZSize);
fprintf(f,"%ld\t", Object->Std.XPos);
fprintf(f,"%ld\t", Object->Std.YPos);
fprintf(f,"%ld\t", Object->Std.ZPos);
fprintf(f,"%ld\t", InitSize->Isz.IXSize);
fprintf(f,"%ld\t", InitSize->Isz.IYSize);

```

```

fprintf(f,"%ld\t", InitSize->Isz.IZSize);
fprintf(f,"%ld\t", InitPos->Ips.IXPos);
fprintf(f,"%ld\t", InitPos->Ips.IYPos);
fprintf(f,"%ld\t", InitPos->Ips.IZPos);
fprintf(f,"%ld\t", Rotation->Rot.XCentre);
fprintf(f,"%ld\t", Rotation->Rot.YCentre);
fprintf(f,"%ld\t", Rotation->Rot.ZCentre);
fprintf(f,"%ld\t", Viewpoint->Vpt.NumVPs);
fprintf(f,"%ld\t", (Viewpoint->Vpt.View)->ObjView);
fprintf(f,"%ld\t\n", (Viewpoint->Vpt.View)->ObjCon);

```

The four programs have been developed using C++ and Superscape's API (application program interface) Developing Kit (SDK). In Chapter 7, the environment data acquisition is also implemented using a similar method, but the final recorded data are not to a text file. The data are saved to the database directly through the ODBC (Open Database Connectivity) mechanism (Section 7.5). The detailed implementation of these four data recording programs is included in Appendix B.

6.4.2 *Scanning all objects in an environment*

As opposed to recording individual objects, the task of scanning all the objects in a complex environment can be tedious. However, the problem is alleviated by using Superscape's simulation control language (SCL) and the scene graph tree data structure. The following code illustrates the method used in this research to save all the environment information in a single file.

```

obnum current, top;          //Object type variables
if(activate(me,0))
{
top = 'RootObject';        //Sets the scanning entrance
while(child(current)!=top) //Exhausts all objects on tree
{
current = child (current);
}
while(current != top)
{
if(sibling (current)!=top)

```

```

{
current = sibling(current);
while(child(current) != 'RootObject')
{      //Records object data
current = child(current);
SaveGen('current');
SaveDyn('current');
SaveSta('current');
SaveSha('current');
}
}
else
current = parent(current);
}
}

```

In this example, the "top" variable was assigned a "RootObject" which is the top of the scene graph tree. When acquiring an object information, this variable is assigned the number of the referred object. The first "while" loop keeps assigning the variable "current" to its child object's until the loop reaches the bottom of the tree. It then lets data acquisition functions save an object information. The second "while" loop traverses the tree backwards until it reaches the starting point. Inside of the second "while" loop, the program continually checks to see whether the current object has a sibling whilst performing the task. When it reaches the end of the sibling list, the program moves one level up. When finally the current variable is equal to the top object in the tree, the process is finished.

The information recorded in the intermediate file can be converted into a database file using a text file parser. This project used Microsoft Access text parser. Figure 6.9 shows the generated environment standard information table.

VirObjID	MVEID	XSize	YSize	ZSize	XPos	YPos	ZPos
438	5	248253	160583	92711	0	0	706456
313	5	248253	160583	92711	0	0	1049115
578	5	39999	39999	39999	418286	0	827264
575	5	39999	39999	39999	421294	0	453376
580	5	79999	126479	87647	537876	0	257298
569	5	133894	135070	120742	566586	0	1275401
552	5	137494	154102	176242	557372	0	633910
540	5	167863	135463	96431	602417	0	356676
535	5	95999	154399	103999	710388	0	1074274
530	5	95999	154399	103999	714011	0	878253
517	5	134791	147759	79191	43472	0	58465
224	5	159997	79997	63997	25652	0	239522
135	5	159997	79997	63997	29044	0	358754
46	5	159997	79997	63997	20534	0	483492
43	5	39999	39999	39999	2271560	300	2213655
40	5	71415	1000	295620	893061	0	225276
37	5	89757	1000	473072	683072	0	329240
36	5	84425	1000	277306	637777	0	574773
33	5	69918	1000	171202	542049	0	827230
30	5	69918	1000	171202	542049	0	1017087
27	5	69918	1000	171202	267338	0	1249884
24	5	69918	1000	171202	267338	0	1110546
23	5	69918	1000	171202	227441	0	1167669
20	5	84425	1000	277306	207034	0	834078
17	5	84425	1000	277306	207034	0	638126
14	5	69918	1000	171202	267338	0	446543

Figure 6.9 Environment standard information table

6.5 ENVIRONMENT CONSTRUCTION FACILITATED BY THE DATABASE

To construct an environment using the data recorded in the database, only four steps are necessary. (i) Retrieval of a template environment and its scene graph, (ii) retrieve and render the base-shape geometry, (iii) search for static information and assign it to the virtual objects, and (iv) retrieve dynamic data to form simulation tasks and to set up objects' interactions.

6.5.1 Constructing the scene graph of an environment

A scene graph can be built from scratch, or by loading an existing one directly from a virtual environment description file, and modified. The first method is applied in the bottom-up approach presented in Chapter 3, and adopted in the template environment construction. Basically it is a CAD modelling process and demands the skill of using environment-authoring tools. The second method requires file format conversion and graph reconstruction. In this work, the second method is used. When constructing an environment, users do not need to build an environment from scratch, but to

customise the closest scene graph of a template environment by insertion, deletion and data modification.

	Bit E0 Env-Type	Bit E1 Func-Type	Bit E2 Primary-Obj	Bit E3 1st&2nd Obj-Num	Bit E4 Detail-Level
Environment Coding Scheme	Production Cell (0)	Milling Unit (0)	Vertical Lathe (0)		
		Robot Cell (1)	Horizontal Lathe (1)	Single & Single (0)	Geometric Shapes (0)
		Welding Unit (2)	CNC Lathe (2)	Single & Multi (1)	Functional Object (1)
		Turning Unit (3)	Lathe and Robot (3)	Multi & Single (2)	
	Assembly Line (1)	Inspection Unit (4)		Multi & Multi (3)	
		Electrical Goods (0)			
		Mechanical Parts (1)			
	Stock Room (2)	Automobile (2)			
		Miscellaneous (3)			
	Others (3)			Monitor Wall (0)	Shapes (0)
			Control Panel (1)	Keyboard Based (1)	Full Simulation (3)
Control Room (2)			Others (2)		

Figure 6.10 Template environment-coding scheme

6.5.2 Retrieving a scene graph of template environments

The retrieval of a template environment is achieved by using an environment code which indexes to the suitable environment for a specific simulation task. Every constructed template environment in the database has a unique index code based on a coding scheme shown in Figure 6.10.

The template environment code has five digits, each represents a specific characteristic of the environment and has a value range to define its strength level. The characteristics relate to the environment type, primary object, and simulation ability. For example, for a manufacturing cell, the environment type code will be 0. Otherwise, the digit value is chosen from 1 and 3. The primary and secondary object digit (1st&2nd Obj-Num) decides the process capacity of an environment. The simulation level digit decides the available simulation functions in an environment, the lower the digit value the less simulation activities are available. It is important to point out that the presented hybrid environment-coding scheme is an experimental

prototype. It can be expanded and refined to fit with other more comprehensive environment categorising schemes.

6.5.3 *Modify scene graph*

Since the database stores only a limited number of templates, usually modification on a retrieved template environment is needed when a specific environment is to be constructed. The fewer the templates in the database, the more modifications will be needed. However, after the system has been used for a period of time, more template environments can be created and added into the database and the amount of modification will be reduced.

Scene graph modifications can be simplified into adding, deleting and moving objects around the graph tree. For example, the object insertion is a process of filling the data structure and locating the position in the graph tree where the object will be placed. This includes three steps.

Step 1: Retrieving data

This is to retrieve the data from the database to construct a place-holder (described in Chapter 7). The retrieval process is in fact a data query process. The query code is as follows:

```
SELECT Object_properties //Essential place-holder's data
FROM table_reference //Involved table reference
WHERE search_condition //Retrieval conditions
```

For example, if a user wants to create a cubic object in the environment scene graph, the query for searching for a stored cube object, its size and position would be programmed as below:

```
SELECT GeneralRef.MVEID, GeneralRef.MVENAME,
ObjectLists.VirObjID, StandardInfo.XSize, StandardInfo.YSize,
StandardInfo.ZSize,
StandardInfo.XPos, StandardInfo.YPos, StandardInfo.ZPos,
```

```

FROM ((GeneralRef INNER JOIN ObjectLists ON GeneralRef.MVEID =
ObjectLists.MVEID) INNER JOIN StandardInfo ON
ObjectLists.VirObjID = StandardInfo.VirObjID)
WHERE (("MVEID"=1) AND ("XSize" == "YSize" == "ZSize"));

```

The returned values of the query result are:

```

VirObjID = 1; MVEID = 1;
Xsize = 1000; Ysize = 1000; Zsize =1000;
Xpos = 2000500; Ypos = 1000; Zpos = 2000500;

```

Step 2: Construction of the main objects

The following step is to apply the returned data to an environment rendering process. This process requires a direct access to the internal data structure of the rendering buffer. It is programmed as dynamic-link-library functions similar to those developed for recording environment information (see Section 6.4). The format of the buffer can be represented as a structural code as follow:

```

static struct          //The buffer structure
{
    T_STANDARD Std;    //Declare basic object data
    short Term;       //End mark for the data
} Buffer=
{
    {
        E_CTSTANDARD, sizeof(T_STANDARD),
        sizeof(T_STANDARD)+sizeof(short), //Total size
        0,
        0,0,
        NULL,NULL,
        0, //Compiler requirements
        XSize,YSize,ZSize, //Place-holder size
        XPos,YPos,ZPos, //Place-holder position
        4000,
        0,0,
        0,0,
        0 //Fills to complete the structure
    },

```

```

-1 //Data section end mark
};

```

These functions are embedded as default procedures to enable users modifying an environment by entering function parameters at run-time.

Step 3: Traversing the scene graph tree

One reason to have a hierarchical structure to hold the template scene data is to help define the position and orientation relationships between objects in the environments. For instance, to build a composite object in the scene, the object is treated as a single object in relation to the rest of the scene. Its individual parts are considered to be a collection of distinct objects in relation to this composite object. In KAMVR, the object can be moved around the scene graph tree by directly calling the VRT functions. The VRT functions check the relative information of the object, such as child, parent and sibling. Then the object is moved around in the tree through calling other functions, for instance, adopting and re-linking to put the object in a different position.

6.5.4 *Assigning object properties*

After the object cubical bounding volume (or place-holder) is formatted and inserted in the scene graph tree, the next step is to assign static and dynamic properties' data to the object. Although strictly speaking, object shape properties also belong to static information and can be imported from the database. However, it would require much more memory and computing time to accomplish such a complex process.

To avoid this difficulty, the function for assigning object properties takes two parameters, shape number and an integer-type array that holds the property ID index. Since the placeholder is already being created and attached to the scene graph, this function only inserts the allocated spatial space with the shape geometry and the properties. Table 6.1 shows the object properties that can be added to the object placeholders.

Property ID	Attribute	Property ID	Attribute
0	Standard	16	Bending
1	Colours	17	Null
2	Initial Colours	18	Speech bubble
3	Rotations	19	Null
4	Distancing	20	Collisions
5	Angular velocities	21	Initial position
6	Null	22	Dynamics
7	Animations	23	Lit colors
8	Null	24	Initial lit colours
9	Null	25	Null
10	Animated colours	26	Textures
11	Null	27	Sorting cuboid
12	Null	28	Textures used
13	Null	29	Sounds used
14	Light source	30	Null
15	Initial size	-1	End of list

Table 6.1 Object static and dynamic properties

Figure 6.11 shows a constructed environment based on a template environment that has been modified using property data retrieved from database. Ten transportation belt sections have been inserted, and the machines were duplicated with different locations to fit in the layout.

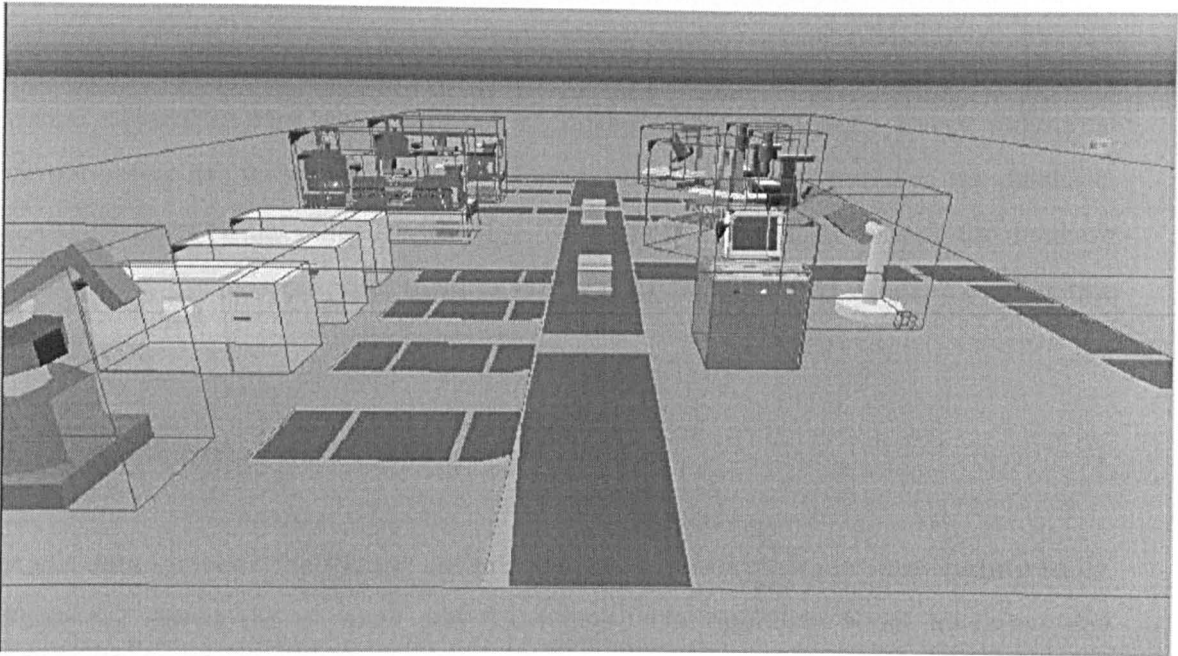


Figure 6.11 Environment constructed from a template VE

6.6 CONJUGATING MANUFACTURING DATA AND VE

Manufacturing knowledge and data are not essential for environment construction and is certainly too big an operation to be managed in a single database. Manufacturing knowledge can come from many different formats such as drawings, spreadsheets, rules, experiences and even expert systems. In this research, methods have been attempted to provide a mechanism to conjugate VE with manufacturing knowledge.

Generally speaking, manufacturing information can be classified as facility information, machining activity information and process knowledge as shown in Figure 6.12.

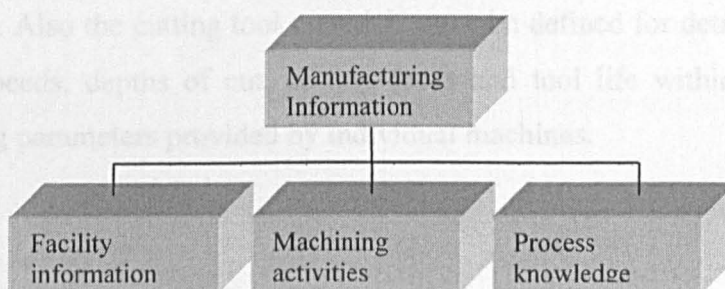


Figure 6.12 Manufacturing information composition

As introduced in section 6.3, a virtual environment is composed of virtual objects, virtual objects in turn are composed of further sub-components. Every individual virtual object has its static and dynamic properties, which are stored in the database. KAMVR links virtual objects with classified manufacturing data through the database as a bridge. For this purpose, manufacturing data are classified as static or dynamic data.

6.6.1 Static manufacturing data

Static data includes the facility information concerning machines and cutting tools. Table 6.2 shows the physical and functional facts, and data about machines and cutting tools, which influence the machining conditions, optimising processes and the selection of machines and cutting tools.

General attributes of machines and cutting tools include their names, the category number of machines and cutting tools, the maximum power of the machines and the cutting tool materials.

The machine names and numbers are used only as index information to identify individual machines and cutting tools. When the final process plan is produced, for instance, the name and the number of machines and cutting tools should be referenced to indicate what type of machines and cutting tools are used for a machining process.

The power of a machine process serves as boundaries for the maximum and minimum machining parameters used to control the metal removing rate in the machining processes. Also the cutting tool materials must be defined for determining the correct cutting speeds, depths of cut, cutting feeds and tool life within the limitations of machining parameters provided by individual machines.

Category	Information
General Attributes	Machine names and numbers Machine powers Cutting tool name and number Cutting tool material
Machine Movements	Machining movements Auxiliary movements Motion primitives Coordinate frames and datum
Cutting Tool Geometry	Cutting tool types Cutting tool forms Cutting primitives
Capacities of machines and tools (Machining activities and process knowledge)	Dimension capabilities Attainable accuracy Machining parameters Capacity primitives

Table 6.2 Machine tool knowledge

6.6.2 *Dynamic machining activities*

The machine tool dynamic activities are simplified as translation, rotation, scaling, and changing appearance. In this way, all the activities that occur in a machining process can be seen as a composite activity assembled from primitive motions. Machine movements can be described by so-called fundamental movements, each having its own moving and rotating direction along a coordinate axis with a defined step unit. Any other machine movements can be assembled from the algebraic calculation and matrix transformation of multiple primitive motion units.

6.7 CONCLUSIONS

The KAMVR database system developed in this work stores and manages various environment information in an explicit manner that enables the user to query object

information and modify the environment scene graph. Manufacturing data captured within an environment is categorized according to its VR features (static and dynamic). The database currently only has a limited manufacturing information data file and the link to the environment table is through a one-to-one mode.

CHAPTER 7

**ENVIRONMENT CONFIGURATION
AND COMMUNICATION**

This chapter starts with the concept of configurable virtual environments and then describes the methods and techniques for its implementation using the layered environment structure presented in Section 3.3.2 to define multi-level object properties. Object properties are essential for application programs or user defined systems to access and configure the environment at system run-time. To verify those methods and techniques, and to achieve a demonstrable environment, this chapter also presents a real-time database access mechanism where records can be bound with environment properties to allow the automatic update of the environment database. A case study of connecting physical robots with a virtual environment is used to explain the communication processes.

7.1 CONFIGURABLE VIRTUAL ENVIRONMENTS

A virtual environment designed for a specific application is difficult to reuse due to its fixed object properties, pre-programmed simulation controls and human-environment interactions [Mironov 1998]. A configurable virtual environment has the potential to overcome this problem through a run-time and application-based configuration interface to rearrange and reset its contents and controls to meet different application requirements.

In Section 3.3.1, it was stated that a virtual environment can be constructed in a hierarchical layered structure, where each layer has its explicit functions that are exposed to application programmes. In this way, communications and controls can be established between the virtual environment and the application programmes. The following sections describe in detail how this was achieved.

7.2 ACCESSING ENVIRONMENT PROPERTIES

From a user point of view, an ordinary VR environment is in fact a "canned data block" that can only allow users to view it from various pre-defined perspectives. An essential feature of a configurable environment is that it should allow users to directly access its data, or at least its properties, to achieve the following outcomes:

7.2.1 *Configuring environment data structure*

In fact, an environment is a data file that is loaded into memory at run-time. For example, in Superscape VRT, it is a data buffer referenced by pointers.

The first part (256 bytes) of the file contains a standard header that describes general information about the environment (e.g. environment name, see Figure 7.1). Following the header there is an object list defined in C "struct" format as shown in list 7.1.

```
typedef struct
{
```

```

Object data member variables
} T_OBJDATA_CHUNK

```

List 7.1 Virtual environment object data definition

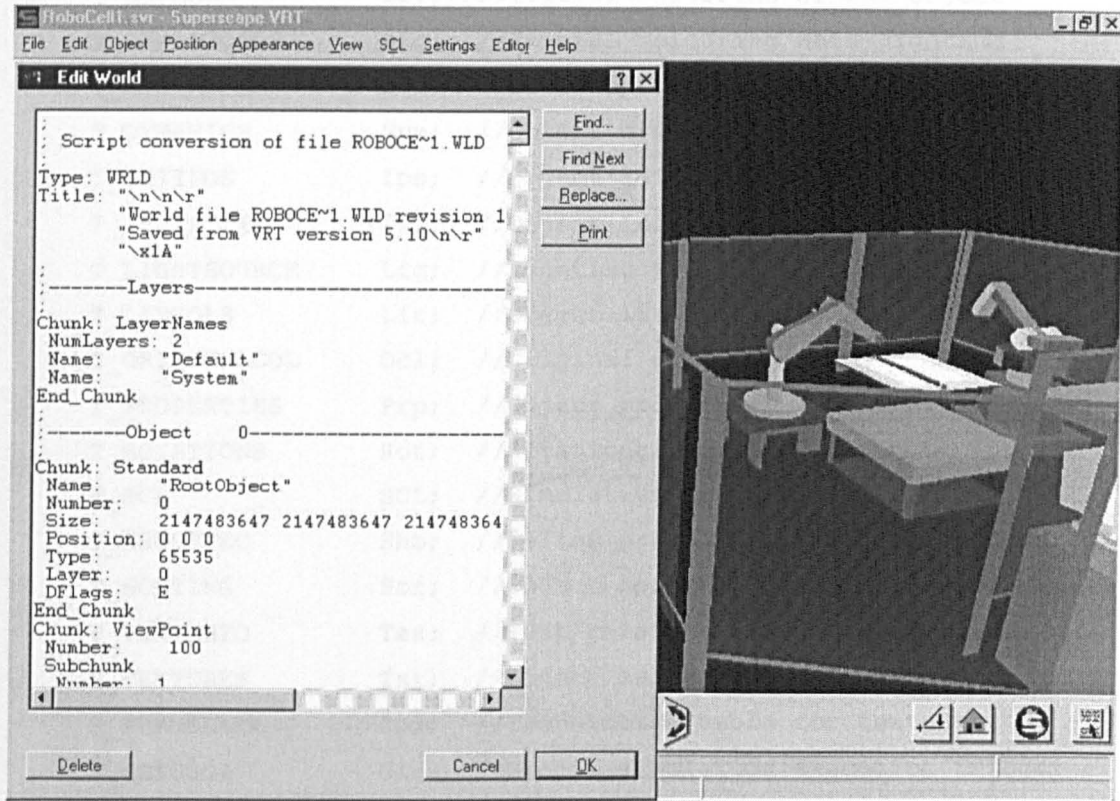


Figure 7.1 Virtual environment script header

Following the object list definition is the data section defined as C “union” and called `T_WORLDCHUNK`, see List 7.2. Therefore, a pointer can be used to access the entire data structure and its data members. For example, the size of a virtual object can be located by a pointer p , which points to the `STANDARD` data section, $XSize = p \rightarrow Std.XSize$.

```

typedef union
{
    T_STANDARD      Std; //The standard data section
    T_ANGVELS      Ang; //Object angular velocity section
    T_ANIMATIONS   Ani; //Animation movement definition
    T_ANIMCOLS     Acl; //Animation colour data
    T_ATTACHMENTS  Att; //Specify objects attachment

```

```

T_AUTOSOUND      Asn; //Sound definition
T_BENDING        Ben; //Shape bending control
T_BUBBLE         Bub; //Speech bubble definition
T_COLLISION      Cln; //Collision detection setting
T_COLOURS        Col; //Object colour data
T_DEFCOLS        Def; //Initial colouring of the object
T_DEFLITCOLS     Dlc; //Initial colouring when lighting
T_DISTANCE       Dis; //Distance replacement setting
T_DYNAMICS       Dyn; //Object dynamic information
T_INITPOS        Ips; //Object initial position
T_INITSIZE       Isz; //Object initial size
T_LIGHTSOURCE     Lig; //Lighting source definition
T_LITCOLS        Lit; //Object lit colour
T_ORIGINALCOL    Ocl; //Original colour setting
T_PROPERTIES     Prp; //Object properties
T_ROTATIONS      Rot; //Rotational definition
T_SCL            SCL; //Simulation control program
T_SHOOTVEC       Sho; //Define projectiles path
T_SORTING        Sor; //Object spatial position sorting
T_TEXTINFO       Tex; //Text relate to the Object
T_TEXTURES       Txr; //Object surface texturing
T_TRANSLATE      Spx; //Translation table for textures
T_TRIGSCL        Glo; //Global simulation execution trigger
T_TRIGSCL        Loc; //Local simulation execution trigger
T_VIEWPOINT      Vpt; //Viewpoint setting data
T_MATERIAL       Mat; //Object material type information
} T_WORLDCHUNK;

```

List 7.2 Virtual environment data type

T_WORLDCHUNK is organised in four blocks according to its accessing methods: (i) shape properties, (ii) static properties, (iii) dynamic features, and (iv) general information.

7.2.2 *Configuring object shape properties*

Object shape property data are the lowest level of information in the data hierarchy. These define object geometry in terms of points, lines and facets. By directly

accessing and altering these data, users or application programs can change the geometric appearance of individual objects through a sub-data section derived from T_WORLDCHUNK. This derived sub-data section is listed in List 7.3.

```
typedef union
{
    T_ANIMCOLS      Acl; //Shape animation colour
    T_COLOURS       Col; //Shape colour
    T_FACETCHK      Fac; //Defines the facets make up the shape
    T_LINECHK       Lin; //Define the lines of facet
    T_LITCOLS       Lit; //Colouring when lighting
    T_POINTSCHK     Pnt; //Define points make up the shape
    T_SCL           SCL; //Shape construing program
    T_SHAPESIZE     Siz; //Shape size
    T_TEXTINFO      Tex; //Shape related text
    T_TEXTURES      Txr; //Texture of the shape
    T_TRANSLATE     Spx; //Translation table for textures
    T_NORMALS       Nor; //Normalise shape angles
} T_SHAPECHUNK;
```

List 7.3 Object shape information

Accordingly, a set of operation functions were designed to access the shape properties, they are:

- ChPoPos - accessing the absolute position of a specific point
- ChSpSize – accessing the size of a specific object and its geometric properties
- ChLitCol - accessing the lit colour properties
- ChColor - accessing the colour information and related texture properties.

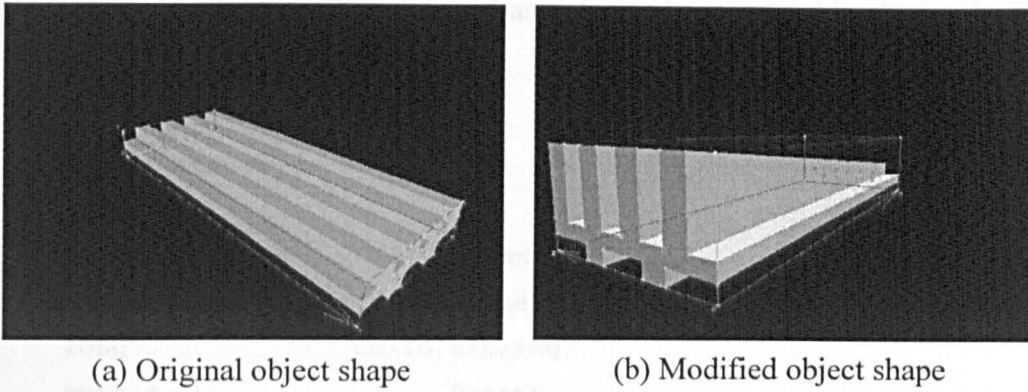


Figure 7.2 Example of accessing shape property data

Figure 7.2 shows a snapshot of a milling machine work table. The first is the original design and the second shows how it can be changed using functions `ChSpSize` and `ChLitCol`.

7.2.3 Configuring object static properties

Object static properties are parameters defining a virtual object's appearance, position and orientation in an environment. Figure 7.3 shows a flow chart of how these properties are accessed.

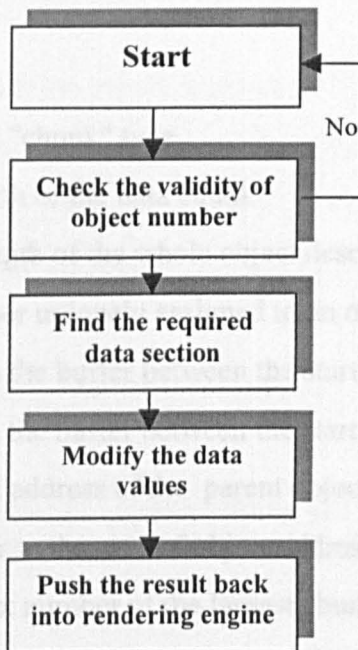


Figure 7.3 Accessing object static properties

The object static properties are accessed and changed from an object's standard data which is defined in List 7.4.

```
typedef struct
{
    unsigned short    ChkType, Length,
                    TotLen, Number;
    long             Child, Sibling;
    void *           Parent;
    void **          List;
    unsigned short    MaxChunk;
    long             XSize, YSize, ZSize,
                    XPos, YPos, ZPos,
                    DiagDis;
    unsigned short    Type, Layer;
    long             DFlags, OFlags;
    unsigned short    Trigger;
} T_STANDARD;
```

List 7.4 Object standard data section

The above data structure contains the standard information that is common to all objects, which include:

- ChkType The data "chunk" type
- Length The length of the data chunk
- TotLen Total length of the whole object description
- Number ID number uniquely assigned to an object
- Child Offset in the buffer between the start of this object to its first child
- Sibling Offset in the buffer between the start of this object to its first sibling
- Parent Absolute address of the parent object
- List A pointer to the start of object address list
- MaxChunk The index number of the largest chunk in the object
- XSize, YSize, ZSize The size of the object bounding cube
- XPos, YPos, Zpos The position of the object relative to its parent
- DiagDis Not used

- `Type` The index of the shape that this object based on
- `Layer` Layer number this object belongs to
- `DFlags` Flags reflecting the data structure of the object
- `OFlags` Flags reflecting the object status
- `Trigger` Flags indicating the object event types

The implementation of accessing the object static properties is by dynamic-linked-library (DLLs), which:

- Defines a pointer to each data section;
- Pops operation parameters from the environment data stack;
- Finds data address for the specific object properties;
- Pushes the address back to the return stack as a writable pointer;
- Sends the property to render engine as a variable.

The designed functions for accessing the object static data are:

- `ChObjSz` - accessing the environment data section which contains object size to configure its geometry
- `ChObInSz` - accessing the object initial size to change its dimension
- `ChObjPos` - accessing the object positional information to alter its 3D position
- `ChObInPo` - accessing the object initial position to set up its starting point
- `ChObjCol` - accessing the object colouring information to configure its colouring property
- `ChObLiSo` - accessing the object lighting source information to change its lighting property
- `ChObjTex` - accessing the object texture information to configure its surface property

Figure 7.4 shows the result of the configuration attained through these functions. It is an environment of a virtual lathe that is fully configurable on the static object properties.

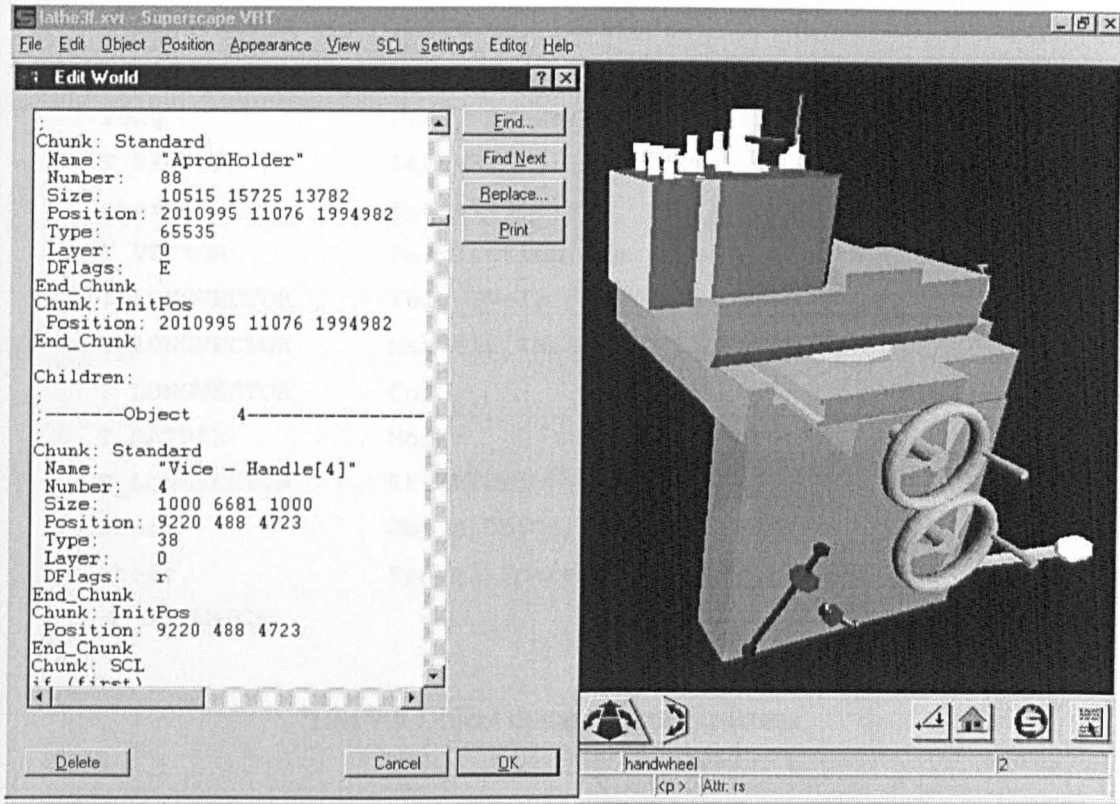


Figure 7.4 Virtual object static data

7.2.4 Configuring dynamic properties

Accessing and changing the dynamic properties of an object follows a similar approach, but different handles are assigned to different data sections that contain the dynamic properties including collision, rotation and translation. List 7.5 to 7.7 shows the data sections defined for accessing these dynamic properties.

```

typedef struct
{
    unsigned short    ChkType, Length; //Data type and length
    long              MType, IMType;
    long              Fuel, IFuel;
    short             Collided, CollCube;
    short             Flags;
    unsigned short    Coupled;
    short             Grav, IGrav;
    short             Climb, IClimb;

```

```

T_LONGVECTOR    Drive, IDrive;
T_LONGVECTOR    External, IExternal;
T_LONGVECTOR    MaxForce, IMaxForce;
T_VECTOR        GroundFric, IGroundFric;
long            Mass, IMass;
T_VECTOR        DeltaR;
char            Spare1[10];
T_VECTOR        Restitution, IRestitution;
T_LONGVECTOR    Vel, IVel;
T_LONGVECTOR    MaxVel, IMaxVel;
T_LONGVECTOR    CofG;
T_MATRIX        MofI;
T_LONGVECTOR    Stiction;
short           ObjIn, ObjOn, GVel;
short           Spare3, Spare4, Spare5;
} T_DYNAMICS;

```

List 7.5 Object dynamic data structure

```

typedef struct
{
    unsigned short    ChkType, Length;
    short             XRot, YRot, ZRot,
                    IXRot, IYRot, IZRot;
    unsigned short    Spare;
    long              XCentre, YCentre, ZCentre;
} T_ROTATIONS;

```

List 7.6 Object rotational data properties definition

```

typedef struct
{
    unsigned short    ChkType, Length;
    unsigned short    NumColls;
    T_COLLSPEC        Collision[1];
} T_COLLISION;

```

List 7.7 Collision data properties definition

The functions designed for accessing and configuring these data sections are:

- SetObjMo - accessing movement information including rotation, translation and scaling.
- SetExFor - accessing object external force information
- SetObVel - accessing object velocity information
- SetInVel - accessing object initial velocity information
- SetObRot - accessing object rotational information
- SetInRot - accessing object initial rotational information

7.2.5 *Configuring general environment properties*

List 7.8 and 7.9 show the definitions of global environment properties including the environment name, type, general data description, scene graph definition and navigation control path.

```
typedef struct
{
    long Value;
    long Min;
    long Max;
} T_PROPERTY_LONG;
typedef struct
{
    long Offset;
    long Length;
    long Strings;
} T_PROPERTY_STRING;
typedef struct
{
    float Value;
    float Min;
    float Max;
} T_PROPERTY_FLOAT;
typedef struct
{
```

```

    short Value;
} T_PROPERTY_BOOL;
typedef struct
{
    unsigned short    Type;
    union {
        T_PROPERTY_LONG    Long;
        T_PROPERTY_STRING String;
        T_PROPERTY_FLOAT   Float;
        T_PROPERTY_BOOL    Bool;
    } Value;
} T_PROPERTYDEF;
typedef struct
{
    unsigned short    ChkType, Length;
    unsigned short    NumProperties;
    unsigned short    Changed;
    T_PROPERTYDEF     Property[1];
} T_PROPERTIES;

```

List 7.8 Configurable environment properties

```

typedef struct
{
    unsigned short
    Length, ObjView, ObjCon, ObjMis, ObjFir, ShootVec, Point;
    unsigned short
    CurFrame, TotFrame, NumPosCont, NumRotCont, OldZoom;
    unsigned char    VPLock, Type;
    T_POSCONT       PosCont[1];
} T_VIEW;

typedef struct
{
    unsigned short    ChkType, Length;
    unsigned short    NumVPs;
    T_VIEW            View[1];
} T_VIEWPOINT;

```

List 7.9 Data structure for accessing and configuring environment viewpoints

The functions for accessing and configuring the global environment data include:

- ChgVwPnt - accessing viewpoint setting information to configure the viewpoint, e.g. viewpoint position and orientation
- SetLong - accessing object “Long” property flag
- SetFloat - accessing object “Float” property flag
- SetString - accessing object “String” information
- SetBool - accessing object “Boolean” information

7.3 MIGRATING ENVIRONMENT PROPERTIES

With the configurable data structures and corresponding accessing functions introduced in Section 7.2, the next step is to migrate object properties from an environment’s internal file or buffer to an external application in a way that can allow users to make any change and control on them.

7.3.1 *Extracting and migrating shape properties*

Figure 7.5 shows a snapshot of an interface that can extract shape data of a virtual object. This interface (and other interfaces shown later in this section) was specially designed as an add-on module to most applications with DDE functions so that once the object properties are extracted from an environment’s repository data pool, they are automatically streamlined to the application database or data buffers.

The image shows a dialog box with the following fields and controls:

- ShpName: Cox
- ShpID: 13
- ShpCol: 12
- ShpXSize: 10000
- ShpYSize: 10000
- ShpZSize: 4000
- ShpLight
- OK button
- Cancel button

Figure 7.5 Interface for setting object shape properties

This mode also allows users to change the data between such data transfer. Similarly, when the application's data are to be pipe-lined back to the environment, the interface functions operate in the same way.

7.3.2 Extracting and migrating object static properties

Static properties are exposed in a similar style as the shape properties. For example, when object name, ID number, position, size and colour properties are extracted, users can set them into a virtual environment according to a given rendering requirement. Figure 7.6 shows the static property interface. It works in a similar style to that of the object shape property shown in Figure 7.5.

ObjName	Handwheel
ObjID	2
XPos	2000000
YPos	0
ZPos	2016424
XSize	4277
YSize	3982
ZSize	3576
IXPos	2000000
IYPos	300
IZPos	2000000
IXSize	4000
IYSize	4000
IZSize	4000
ObjColor	12
<input type="checkbox"/> ObjLight	
<input checked="" type="checkbox"/> Vis/Invis	
<input type="button" value="OK"/> <input type="button" value="Cancel"/>	

Figure 7.6 Interface of object static property

7.3.3 Extracting and migrating object dynamic properties

The dynamic properties including translation, rotation and external force are extracted and migrated using an object dynamic property interface as shown in Figure 7.7.

XMove	1000
YMove	0
ZMove	0
XRot	30.0
YRot	0
ZRot	0
XAngV	0
YAngV	127.5
ZAngV	0
XVel	0
YVel	0
ZVel	0
<input checked="" type="checkbox"/> Collision	
<input type="checkbox"/> Trigger	
OK	Cancel

Figure 7.7 Interface of object dynamic property

Using this interface, users can set-up object dynamic functions such as movement and rotation speed. In Figure 7.7, the example object has been given an angular velocity of 127.5 degree along Y axis. The “Collision” and “Trigger” check boxes allow object collision detection and object triggering flags (default functions) to be setup.

7.3.4 Extracting and migrating global environment properties

Finally, for the global environment data and user navigation control data, a global property interface was designed to extract and convey data about the environment name, indexing code, general task and viewpoint control. Figure 7.8 shows a snapshot of this interface. It works in a similar way to those described above.

WldName

WldID

Description

Viewpoint

LongPrp

StringPrp

FloatPrp

BooleanPrp

Figure 7.8 Environment property interface

7.4 SETTING AND UTILISING PROPERTIES

If the configurable environment is to send or to receive from an application, the data properties have to be in a format that can be readable by both ends. Also, the data must be flexible for users to change and update during the transition between the environment and the application. For this reason, an application run-time shell platform was designed to accomplish the task (see Section 8.2). Before describing this platform, it is necessary to explain the techniques and methods used in designing it and how to set and utilise the object properties.

7.4.1 *Setting the simulation triggers for an environment*

The property types, setting methods, and application events of the system are generalised in Table 7.1.

A virtual object in the virtual environment can take the input of event signals, change its own properties and output new events to other objects. This is achieved by pre-defined control programs written in an environment simulation language (SCL in this research). The input events can be of three types: time-based events, action-based events, and condition-based events. For starting the pre-programmed simulation program, triggering functions such as "SetTrigger" has been used as control signals.

An application program can control the environment simulation loop by calling a function at system run-time.

Properties Type	Methods Type	Events Type
Shape properties	Trigger methods	Initialising event
Static properties	Counter methods	Clearing event
Dynamic properties	Marker methods	Environment event
World properties	Viewpoint methods	Device event
	Property setting methods	State event
	Pointing device methods	

Table 7.1 Application platform utilities

7.4.2 *Setting environment counters*

During a conditional environment simulation, by calling a condition-based function, an application program can have a multi-entry-point to the environment, to gain flexible control over the environment. A virtual robot cell control simulation presented in Chapter 8 is set-up in this way. The run-time functions supporting this setting are "SetMarker", "GetMarker", "SetCounter", and "GetCounter".

7.4.3 *Setting object properties*

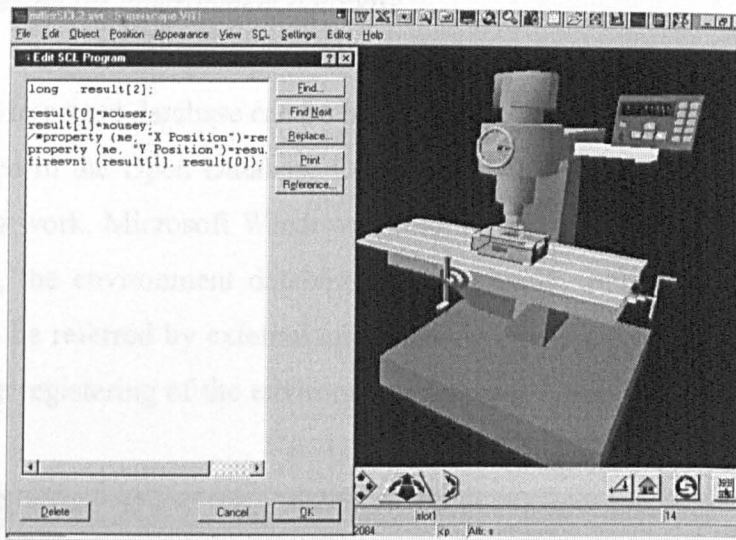
Perhaps the most useful environment setting methods derived in this research are those that directly manipulate individual object properties. The system enables eight such methods.

- GetLongProperty - taking an object number, and its property name as parameters, returning the object long type property value
- SetLongProperty - taking the object number, its property name, and a long type value as parameters, setting the value to the pointed object property
- GetFloatProperty - taking an object number, and its property name as parameters, returning the object float type property value

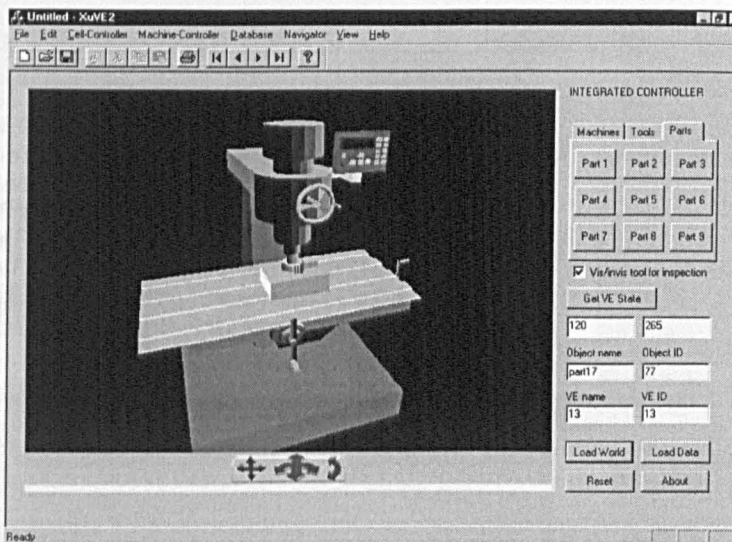
- SetFloatProperty - - taking the object number, its property name, and a float type value as parameters, setting the value to the pointed object property
- GetStringProperty - taking an object number, and its property name as parameters, returning the object string type property value
- SetStringProperty - - taking the object number, its property name, and a string type value as parameters, setting the value to the pointed object property
- GetBoolProperty - taking an object number, and its property name as parameters, returning the object boolean type property value
- SetBoolProperty - - taking the object number, its property name, and a boolean type value as parameters, setting the value to the pointed object property

7.4.4 *Receiving environment events*

When a control or simulation event occurs in a virtual environment - such as an object collision, time limit reached or a certain condition is fulfilled in a simulation loop - an environment signal is sent to a container program. It is then passed to the application program. Suppose a user wants to move the cursor from point A to point B, and acquire the mouse position change during the operation. By declaring two integer variables in the environment simulation program to hold the mouse position, the real-time mouse movements can be passed to the application through firing an event carrying these two variables to the container program. Figure 7.9(a) and (b) has shown the firing event simulation program and the real-time mouse position display in the run-time platform.



(a) SCL firing event carrying two arguments



(b) An application receiving the two arguments

Figure 7.9 Firing event from a virtual environment

7.5 PLATFORM-BASED DATABASE ACCESS

The implementation of a virtual environment and database interaction at system runtime was achieved using Microsoft's open-database-connectivity (ODBC). It is a database application program interface, which enables a container program to access data from various databases and connecting with application operations. The procedure of this accessing method process is described in the following:

7.5.1 Registering the environment database

Before the environment database can be linked to an application environment, it needs to be registered in the Open Database Connectivity (ODBC) utility in the operating system (in this work, Microsoft WindowsNT 4). By registering it in the Data Source Administrator, the environment database address, name, and other information are saved and can be referred by external application programs as a Data Source. Figure 7.10 shows the registering of the environment database VME1 that was introduced in Section 6.3.

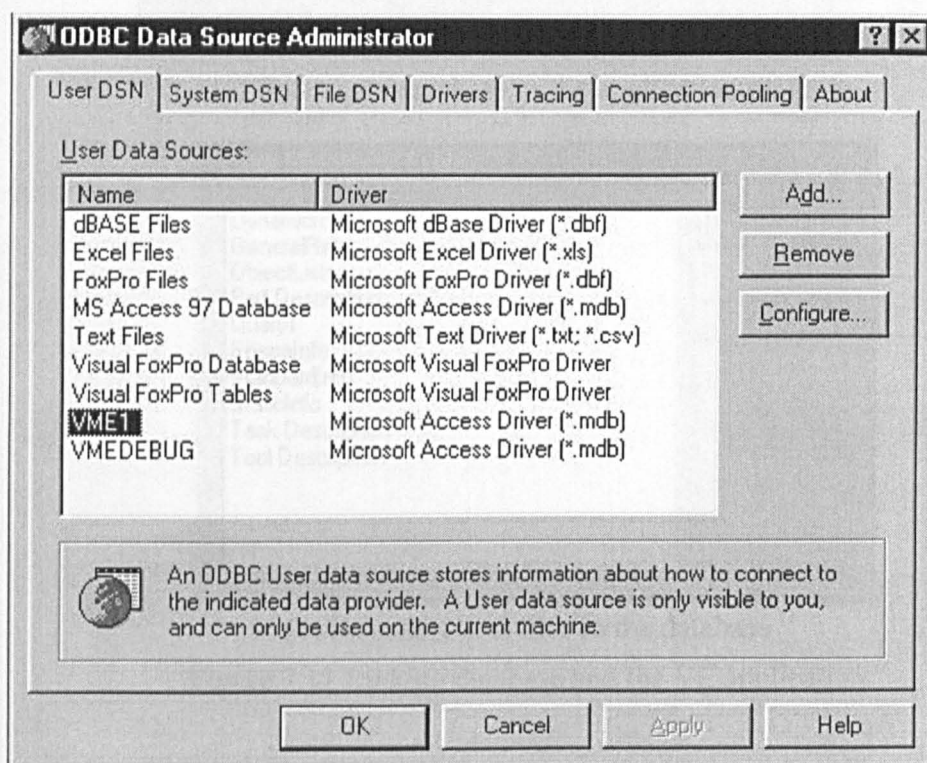
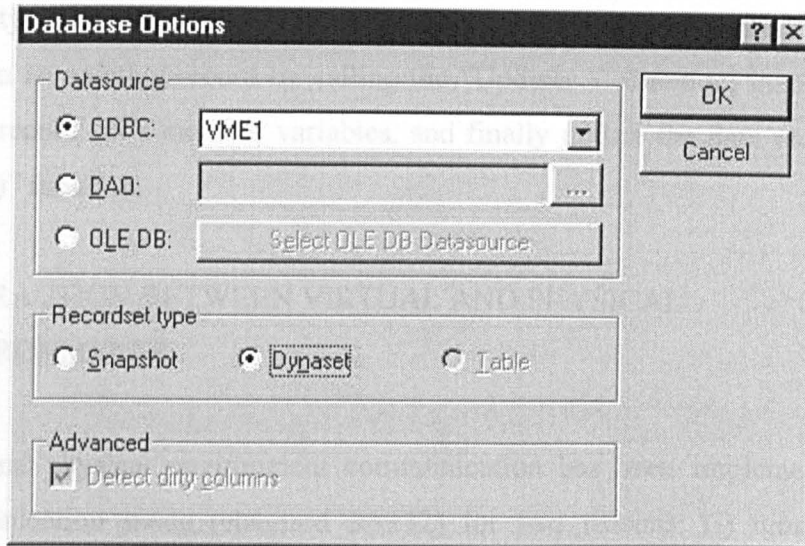


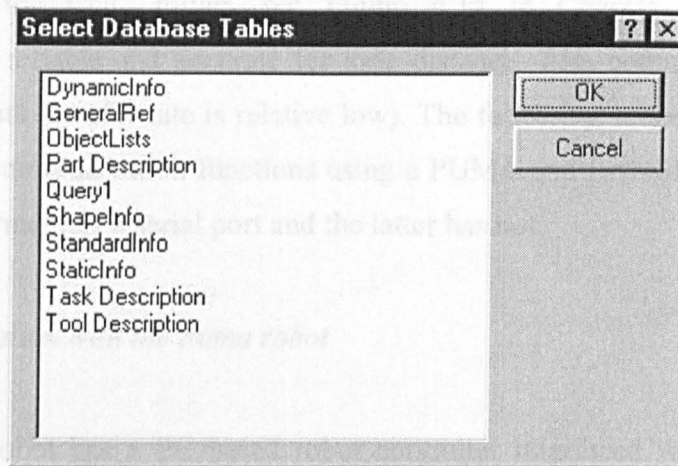
Figure 7.10 Registering the environment database

7.5.2 Connecting the database

After the environment database is registered, it needs to be connected to the application program. Figure 7.11 (a) and (b) shows the connection of the registered database and its data files.



(a) Referring a database in the application program



(b) Selecting data files in the database

Figure 7.11 Linking database and the VE application

7.5.3 Binding environment properties with database records

The data binding of the environment and object properties with corresponding data fields enables environment monitoring, updating and recording. There are two possibilities in this process, retrieving data from the database for environment modification, and recording data or events from the environment to the database. As shown in Figure 7.11, a “dynaset” type data record is applied in the program that keeps synchronisation between the database data and the environment data. The access to the environment and properties of an object has been implemented by the “getProperty” and “setProperty” functions introduced in Section 7.4. Depending on a user’s requirement, a modified object can be saved in a new record by calling the

database utility function (provided by C++ function library) “CRecordset::AddNew()”, or saved in an existing record by calling the “Update()” function, then move the data into the record set’s member variables, and finally update the data record using the “Update()” function.

7.6 INTERACTION BETWEEN VIRTUAL AND PHYSICAL ENVIRONMENTS

The virtual and physical environment communication has been implemented in a serial communication mode (standard RS232) for two reasons: (i) most modern manufacturing equipment has a built-in serial port (in this work, the PUMA robot, the CNC lathe and the CNC miller, see Figure 4.14 in Chapter 4). (ii) serial communication is reliable and accurate for long distance data communication (the drawback is the data transfer rate is relative low). The following sections explain the KAMVR system communication functions using a PUMA and LANSING robot as a case study. The former has a serial port and the latter has not.

7.6.1 *Communication with the Puma robot*

The PUMA 560 robot has a PC based robot controller interfaced with an ageing, inflexible Unimation Mark II controller. The simple hardware servo control was replaced with a software based control strategy. To communicate with multiple peripheral devices, an eight port intelligent RS-232 interface board (PCL-844) was installed in the host computer. All the physical equipment was linked with the host computer through this interface. The PUMA robot has a built-in 10-pin DIN RS-232 port, only the data transmit (Tx), receive (Rx) and the ground (GND) pins were used. The data communication is controlled by the software (see Appendix D) with the computer internal clock monitoring the time intervals, so that handshaking pins are not necessary. The communication signal format was set at 9600b/s with no parity bit. Each data pack has 8 data bits and 1 stop bit. The communication software was programmed in C code. The working procedures are shown in Figure 7.12.

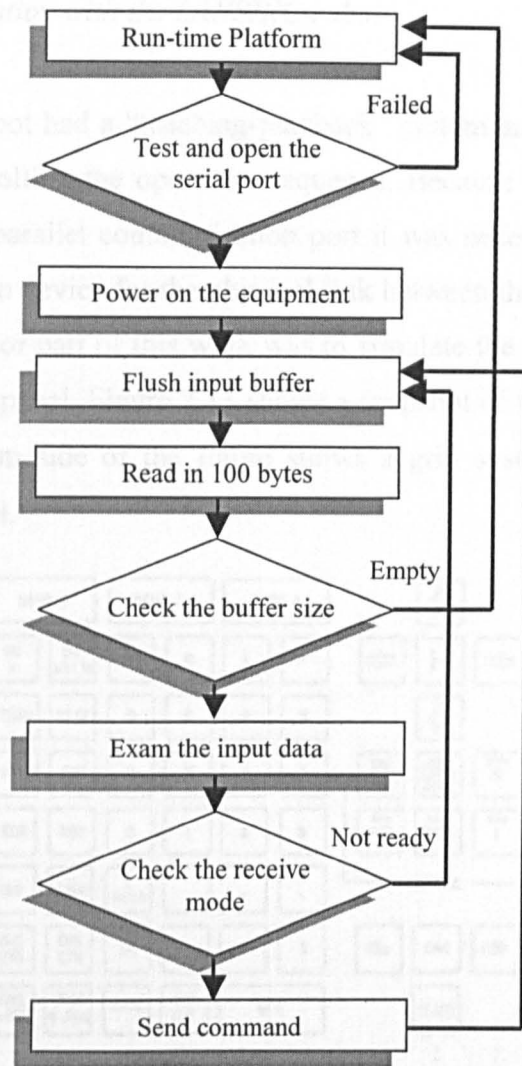


Figure 7.12 Connecting the PUMA robot

It first checks if the multi-port PCL-844 board has been installed and the referred port has been opened, task-specific functions that come with the communication board are used in the program to check the return values. If the state is correct, then the program tries to read the signals from the robot control unit. If they are available, the control program reads and saves them into a pre-defined host computer buffer. Next, the users can read the message from the buffer and decide which command is to be sent to the robot. The program is designed to support both command and file level communication with the ability to transfer space point records and arm joint information.

7.6.2 Communication with the LANSING robot

The LANSING robot had a “teaching-playback” system and a sequential descriptive language for controlling the operation sequence. Because there was no ready-made standard serial or parallel communication port it was necessary to develop a single-way communication device for the physical link between the host computer and robot control unit. A major part of this work was to simulate the control instructions of the LANSING control panel. Figure 7.13 shows a snapshot of the physical control panel. The left and bottom side of the figure shows a grid system to locate each of the buttons on the panel.

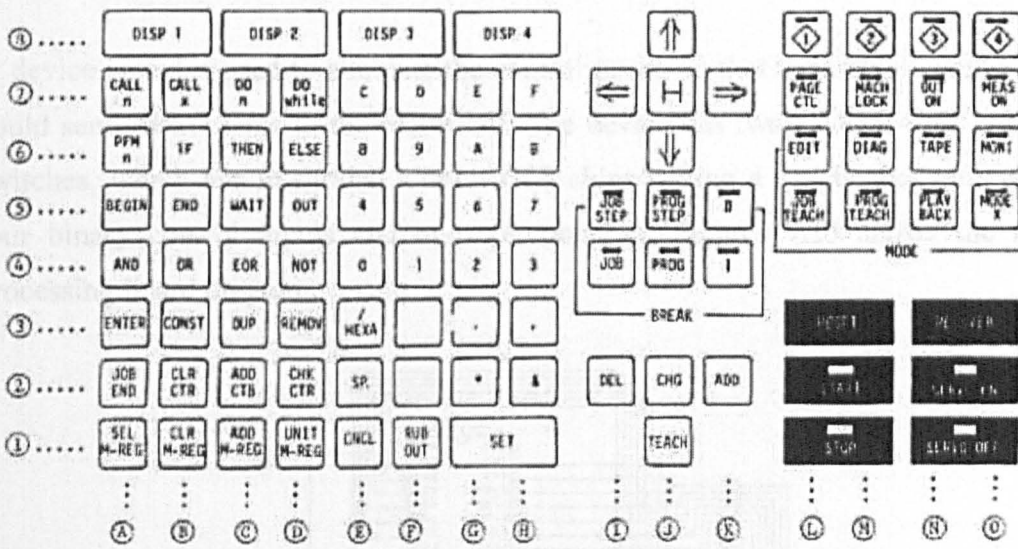


Figure 7.13 Lansing robot control panel

There is a 50-way connector in the LANSING controller to transfer the user instructions input through the control panel to a central processor. It was found to have a similar electrical working mechanism to a conventional keyboard with 16 high-voltage (5v) ways and 14 zero-voltage ways to give a maximum 224 combinations. Figure 7.14 shows the combinations for each button on the control panel. The same grid as shown in Figure 7.13 was used to identify the keys. For example, The “DISP 1” button (8A) is controlled by pin 10 (0v) and 17(5v) on the connector.

8	-	+	-	+	-	+	-	+	/	-	+	/	-	+	-	+	-	+												
	10		17	10		34	10		16	10		35	/	40	34	/	12	17	12	34	12	16	12	35						
7	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+						
	18	25	18	26	18	24	18	27	18	23	18	28	18	22	18	29	40	16	40	17	40	15	12	15	12	36	12	14	12	37
6	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+				
	32	25	32	26	32	24	32	27	32	23	32	28	32	22	32	29	/	40	35	/	38	17	38	34	38	16	38	35		
5	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+				
	19	25	19	26	19	24	19	27	19	23	19	28	19	22	19	29	39	17	39	34	39	16	38	15	38	36	38	14	38	37
4	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+				
	31	25	31	26	31	24	31	27	31	23	31	28	31	22	31	29	39	35	39	35	39	15	39	36	/	/	/	/		
3	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+				
	20	25	20	26	20	24	20	27	20	23	20	28	20	22	20	29	/	/	/	/	/	/	/	/	/	/	/	/	/	
2	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+				
	30	25	30	26	30	24	30	27	30	23	30	28	30	22	30	29	11	17	11	34	11	16	13							
1	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+				
	21	25	21	26	21	24	21	27	21	23	21	28	21	22	22	/	-	+	/	-	+	-	+	-	+	-	+			
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O															

Figure 7.14 The keyboard control combinations

A device was designed to simulate the control panel, so that the virtual control panel could send instructions to the real robot. The device has two CMOS 4067 analogue switches, which are Integrated Circuit (IC) chips having a 16-channel multiplexer, four binary control inputs and one common pin. Figures 7.15 shows the signal processing board diagram.

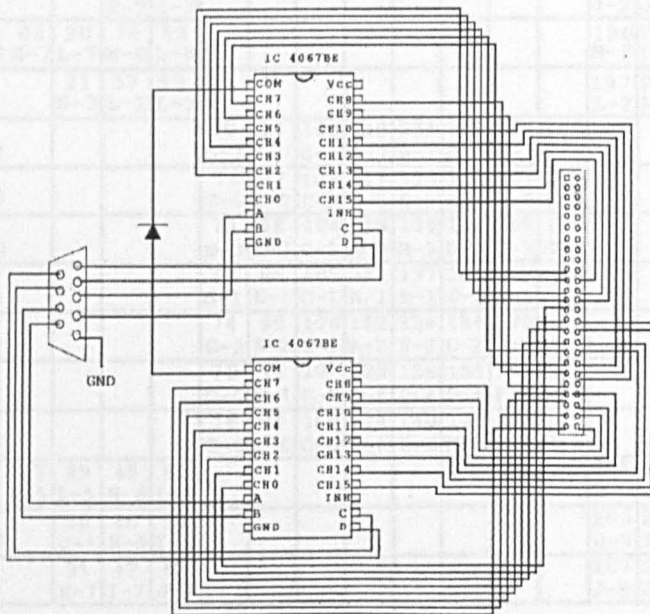


Figure 7.15 Virtual control panel signal processing board

Every combination of input signal can determine an output channel that is connected with the common pin. The circuit board was designed to use a PC generated serial signal (8 data bits) to select two output pins (one from each chip) and a dual-way

7.7 CONCLUSIONS

This chapter described the method of exporting the environment and object properties to a database and external programs. Using the this method, virtually all the data in an environment can be accessed, controlled and modified at environment run-time without the need for off-line editing. This enables an environment to be customised by the user and enhance the usability and flexibility of the environment. Furthermore, this method also enables template environments to be accessed and the database to be updated.

CHAPTER 8

THE RUN-TIME IMPLEMENTATION

This chapter reports how the modules developed in this research were integrated into a system. It also shows the results of testing the KAMVR system for user-environment interaction, environment-database interaction, environment simulation, and virtual environment and real world interactions.

8.1 INTRODUCTION

To verify the domain-analysis based top-down virtual environment construction approach and the KAMVR system development based on it, an integrated “module-container” program was coded using C++. In this program, the template environments, the database, and the application program interface are all embedded entities, which enable the template environments to be developed using various environment authorisers, and allow users to customise a specific environment for given simulation tasks.

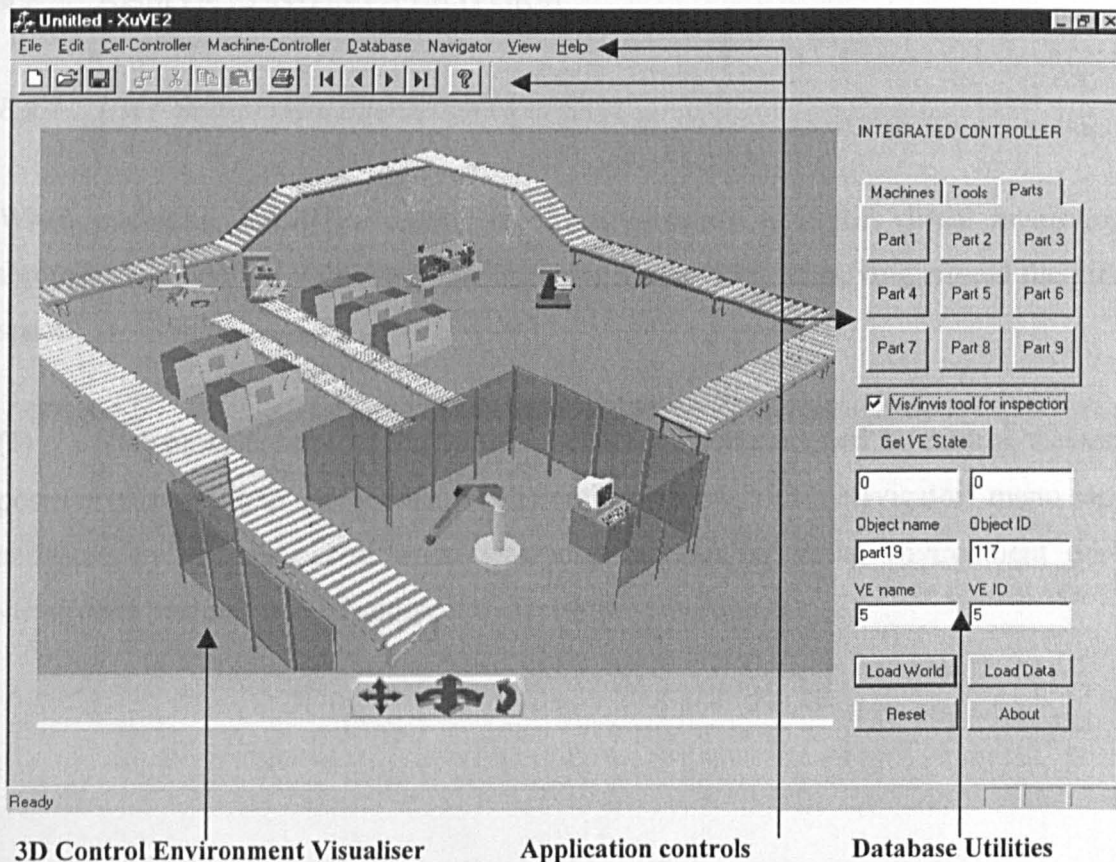


Figure 8.1 KAMVR run-time platform

Figure 8.1 shows a snapshot of the program interface. It consists of three parts: 3D control environment visualiser, application controls and database utilities. For simplicity the following sections, refer to this program as the run-time platform. The run-time platform provides control or monitoring of the VE and applications activities. There are six “textfield” components, two used to monitor the VE or object state. Information, such

as the machining state (running or idle), simulation timing (in milli-second), and mouse cursor position (to identify which object is activated), can be displayed. Other text fields are used to display information retrieved from the database such as object name, identification number, its environment name, and the environment number. The toggle buttons on the run-time platform allow the retrieving of environment from the database and the loading of specific objects according to the user input from the aforementioned textfields.

8.2 KAMVR RUN-TIME PLATFORM

8.2.1 *User-environment interaction controls*

When operating KAMVR, users can directly interact with the virtual environment through manipulating virtual objects, changing viewpoints, or moving around the virtual space.

(1) **Viewpoint control:** A pull-down menu has been designed to control viewpoint positions according to user inputs. As shown in Figure 8.2, the “Navigator” menu can be activated by choosing a different viewpoint number, a virtual environment pre-set viewpoints position will be adopted to display the environment.

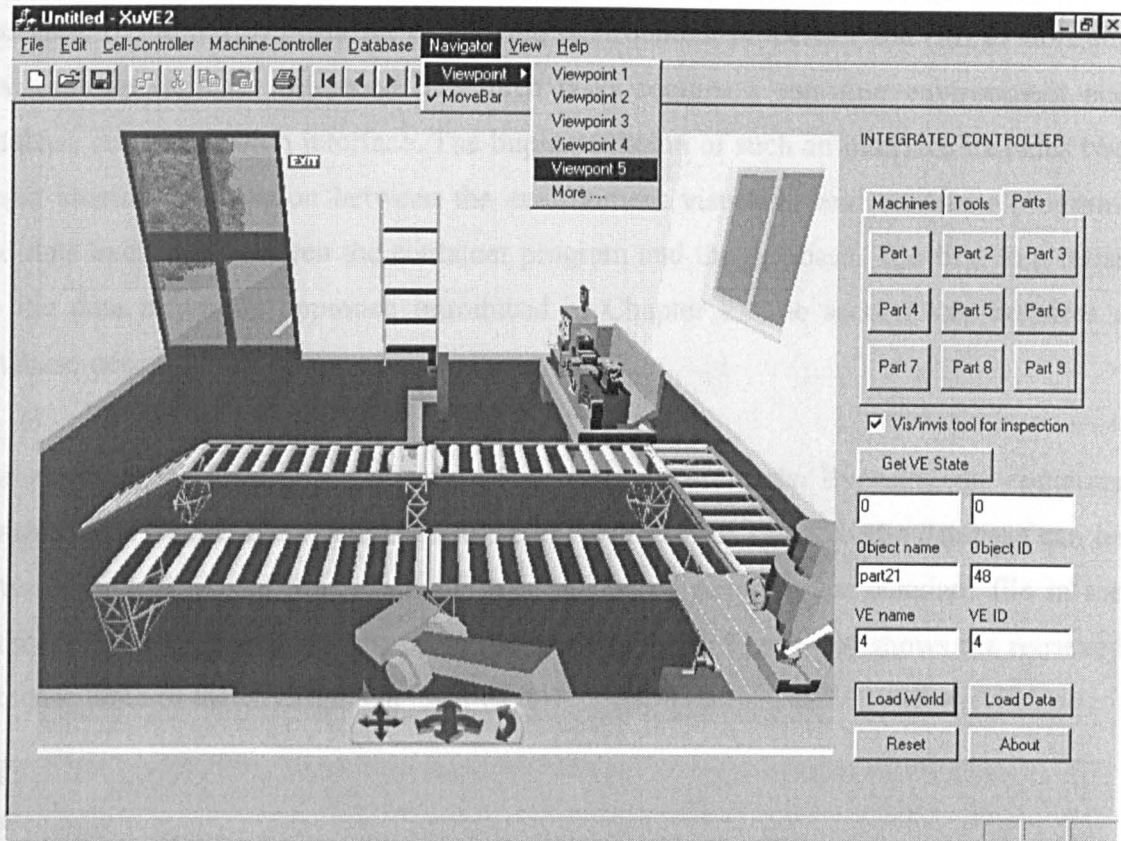


Figure 8.2 Viewpoint control menu

(2) Environment navigation: Except using the default viewpoints to explore an environment, navigation is also possible by using the three icons on the environment navigation bar as shown in Figure 8.3. By toggle and drag the left icon, a dynamic viewpoint will move up, down, left and right. The middle icon moves the viewpoint forward and backward, rotating it left and right, while the right most icon supports tilting the viewpoint up and down.



Figure 8.3 The VE navigation bar

8.2.2 Control environment and database interactions

The purposes of the run-time environment and the database interaction are: (i) to update a database record when a certain condition in the environment is fulfilled or special events

occurred, (ii) to retrieve data for controlling environment properties and (iii) to store the environment for later reference. All three tasks require a run-time environment and database communication interface. The implementation of such an interface includes two steps: sharing information between the environment visualiser and container program, and data exchange between the container program and the database. The first step relies on the data migrating approach introduced in Chapter 7. The second step requires a database-access tool.

The run-time platform provides a set of database access tools. By using the container control icons, menu items, and the text fields, data records saved in the database can be browsed and displayed. For example, the data record saved in the standard file in the database can be retrieved and displayed in the text fields. Figure 8.4 shows the retrieved database table of the environment “workcell4”.

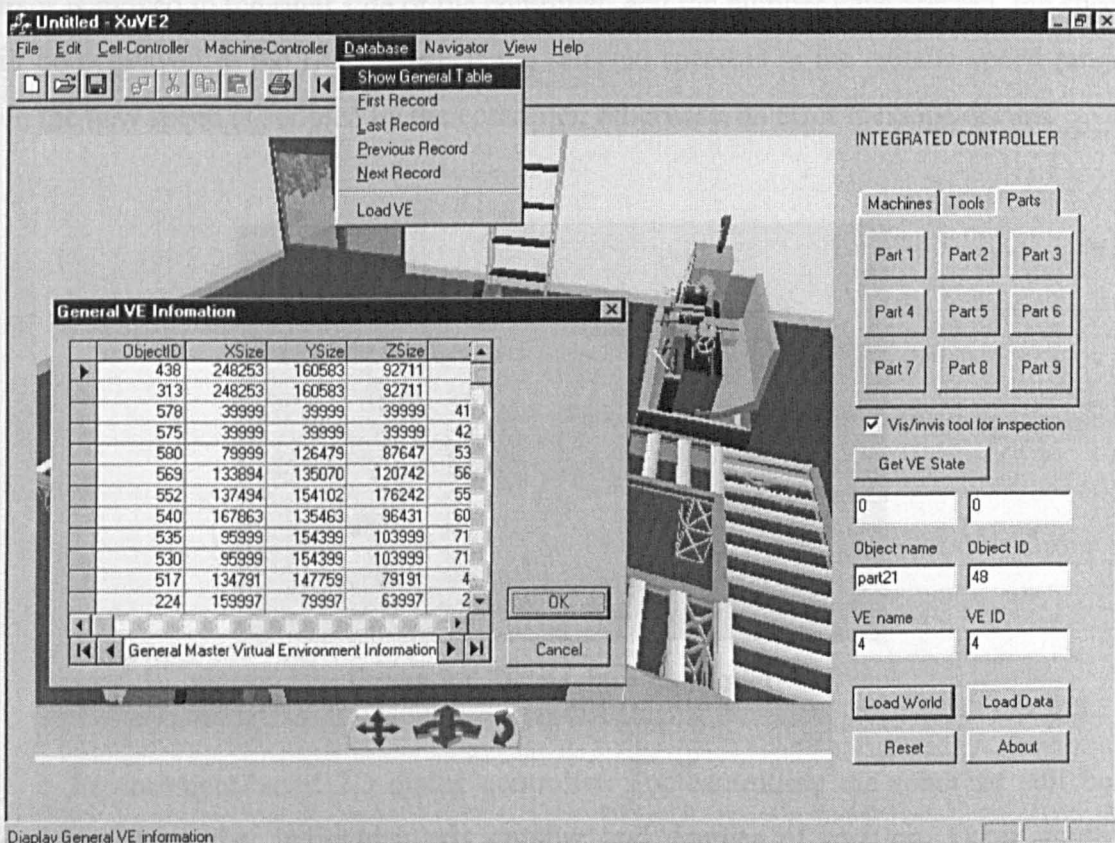


Figure 8.4 Updated general object information

Except for the general VE and database controls, the simulation control functions in the run-time platform have been classified into two levels, functions concerning standalone machine controls and functions concerning cell activities.

8.2.3 Stand alone machine controls

Stand alone machine controls are mainly achieved through Machine Controllers attached on the primary and secondary objects in the environment. There are three different types of machine controller available on the run-time platform, object based 3D machine controller, 2D control dialog box, and a container menu-based controller.

(1) 3D machine controller: as shown in Figure 8.5. It is usually attached on a machine. It controls machine parameters such as spindle speed. In this case, if the mouse cursor is moved to the right side of the controller, and the number keys pressed, the speed will be displayed in the LCD panel. If the selected speed is in the spindle speed range, then the new speed is adopted by the controller, otherwise, an error message occurs.

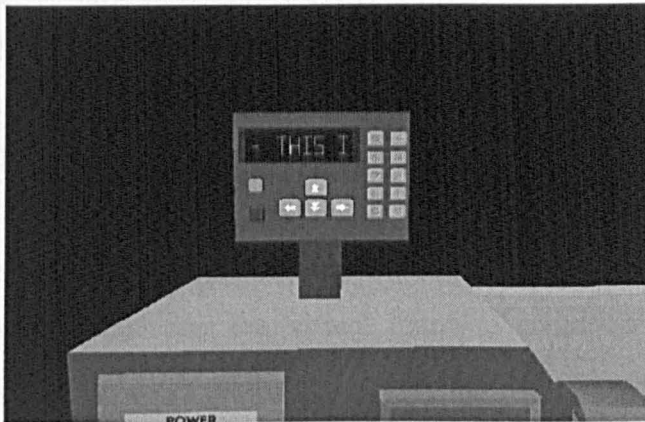


Figure 8.5 3D machine controller

(2) Environment-based 2D dialog controller: For controlling the robot an edit box takes user input, i.e. individual axis number and degrees of rotation. Other control components such as radio boxes, buttons, and slide bar are used to set up the move step size and control arm activities. Figure 8.6 shows the robot control unit dialog box.

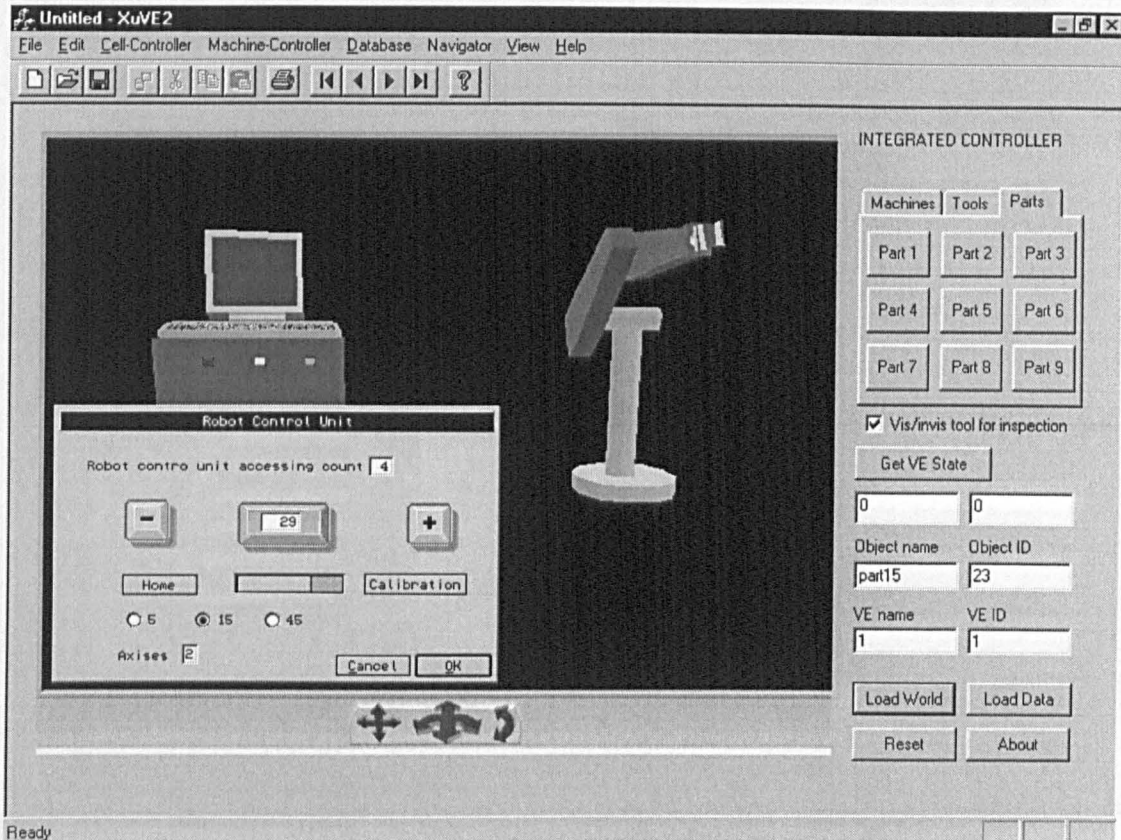


Figure 8.6 Virtual robot control dialog

(3) Run-time platform menu-based control: various machine control dialog boxes have been provided for controlling common manufacturing equipment. Figure 8.7 shows a conveyor controller that can be used to start or stop the conveyor, control rotation moving speed and direction. The control dialog is started by clicking on the platform “Machine-Controller” item, then selecting the “Conveyor”. The menu items also provide the basic functions of connecting the virtual machine controller with the physical machine control unit by assigning each machine an individual serial port on the serial card (see Section 4.3.3 and Section 7.6).

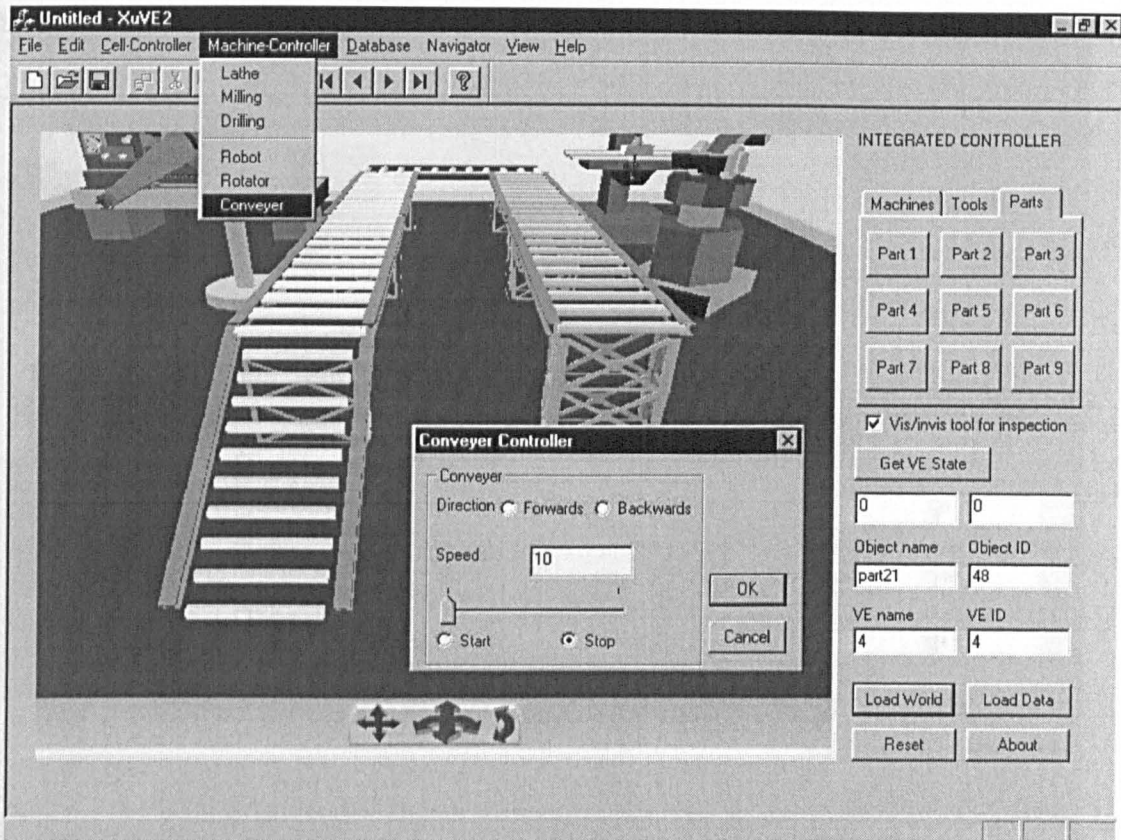


Figure 8.7 System platform device controller

8.2.4 Cell controller CHANISM

The cell controller interface is designed mainly for testing the time-sequenced control of a manufacturing cell. Those individual control commands form the sequence file, which link a series of pre-programmed actions performed by devices in the virtual robot world. As the sequence file plays, line by line, the actions will occur in the virtual world. Figure 8.8 shows the cell controller operation sequence editor.

When the system is started, the actual processing and management of information is dealt with by different application modules. For instance, a database system or a "31-then" knowledge base. There are two ways to get information from the environment and simulate in computer terms: passive mode and active mode. The operation code is provided in Appendix C.

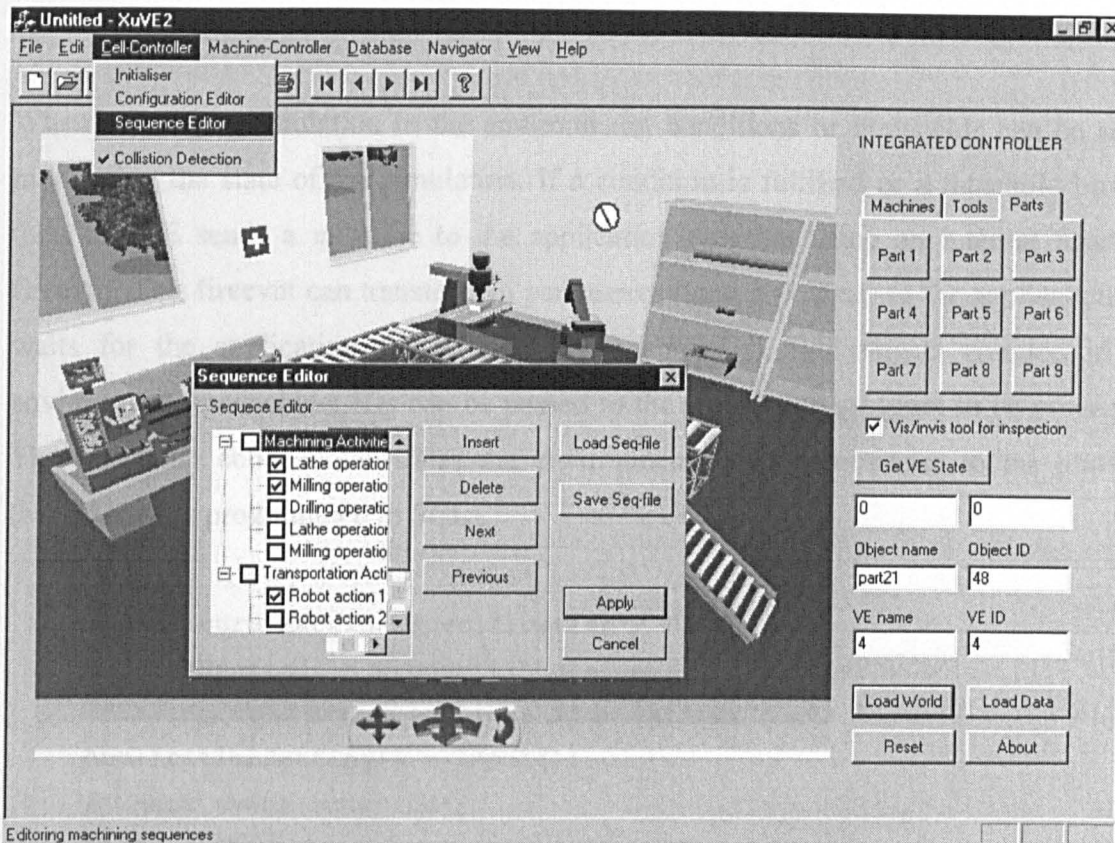


Figure 8.8 Cell controller operation sequence editor

8.3 WORKING MECHANISM

At system run-time, the run-time platform program must be aware what is occurring in the environment for two reasons. Firstly, the container program is a holder for various function modules in the system, which co-ordinates other modules' requirements for data and instructions. Secondly, the container program dispatches event messages in the KAMVR system. In other word, the role of the container in the system is merely a message collector, the actual processing and management of information is dealt with by different application modules, for instance, a database system or a "if-then" knowledge base. There are two ways to get information from the environment and simulation in computing terms: passive mode and active mode (the operation code is provided in Appendix C).

8.3.1 *Monitoring and controlling an environment in active mode*

When running a simulation in the environment, conditions or thresholds can be set to monitoring the state of the simulation. If a condition is fulfilled or a threshold broken, then the VE sends a message to the application program using an internal function: `fireevnt`. This `fireevnt` can transfer two parameters (long type data) to the application and waits for the application to respond. For example, if two objects collided in the environment, their object IDs can be passed to the application program to be processed. The following code demonstrates the environment sending messages to the platform (actual code is programmed in SCL).

```

/*Configure machine specification*/
MachineConfig[] = property("MachineName", "property[]");
/*Taking user input settings from various controllers*/
ApplicationSetting[] = input[];
/*Check event occurred*/
If(EventOccurs)
/*Passing object ID to container*/
Fireevnt(ObjID);
/*Object data retrieved by container and processed*/
wait;
/*Decide the route for the simulation and interaction*/
Marker = property ("Machine", "Marker");
/*Running simulation routes*/
If (marker){simulation instructions}

```

8.3.2 *Monitoring and controlling an environment in passive mode*

The passive mode is used to configure an environment or send instructions to the environment during a simulation loop. For example, if a user wants to set a move distance of an object, the run-time platform calls the functions "setLong" to send the parameter to the SCL program in the environment, the received value gives the boundary of the simulation program (see Section 7.4). Two collided objects can activate an application process, which results in a command that forces the two objects to be

separated by a specific distance. The following code shows the process to configure a virtual lathe with simulation parameters (in contrast with active mode VE and application communication, the control processes for passive mode are programmed in the run-time platform using a C++ program).

```
m_3dcontrol.SetLongProperty("Lathe", "MaxSpinSpeed",
m_strElement[0]);
m_3dcontrol.SetLongProperty("Lathe", "MinSpinSpeed",
m_strElement[1]);
m_3dcontrol.SetLongProperty("Lathe", "MaxTailMove",
m_strElement[2]);
m_3dcontrol.SetLongProperty("Lathe", "MaxSpinSpeed",
m_strElement[3]);
m_3dcontrol.SetLongProperty("Lathe", "MaxSpinSpeed",
m_strElement[4]);
m_3dcontrol.SetStringProperty("Lathe", "Cutter", m_strElement[5]);
```

where, “m_3dcontrol” is the current VE visualiser object name. “SetLongProperty” is the default visualiser class method, which takes three arguments, virtual object name, property name, and the property value. The above VE configuration program sets the lathe object in the environment with a user defined object property values.

In the environment’s simulation program section (SCL), the user-defined parameters are received to control a machine's dynamic behaviour. For example, the user (or application programme) defined lathe properties are transferred into an environment and being assigned to variables in the lathe simulation program.

```
/*Declare long type variables*/
long MaxSpinSpeed, MinSpinSpeed;
/*Assign them with retrieved property values*/
MaxSpinSpeed = property ("Lathe", "MaxSpinSpeed");
MaxSpinSpeed = property ("Lathe", "MaxSpinSpeed");
/* Define the object dynamic activities*/
if(currentspinspeed < MinSpinSpeed)
```

```

exit(1);
else if(currentspinspeed > MaxSpinSpeed)
exit(2);
else {zrot(me)= currentspeed}

```

8.4 RUNNING THE SYSTEM

To demonstrate the run-time platform, consider a user wants to create a machining environment where parts features will be produced in certain quantities. The process starts from recording information about the machined features, the number of parts and the type of operations required to form a task code. This code will be used as an index to search a suitable template environment in the database. The suitable environment name is used to retrieve the template environment and load it into a rendering buffer for display. Finally, the template environment is modified and configured to perform the required simulation tasks. The three sub-sections below explain in more details how the run-time platform performs these activities.

8.4.1 *Forming the task code*

Task coding is carried out by the user through a 3D coding panel. For the first time user, the coding instructions can be displayed in an information dialog (pressing “I” on the keyboard). This shows a definition of each task digit and its options (described in Section 4.3.1). For example, a non-metal shape with a hole and pocket would have a code 21631 as shown in Figure 8.9. The production size (small scale production) and the application type (training) are coded as 10 and 121 individually. The entire task for machining the part is then given as 2163110121.

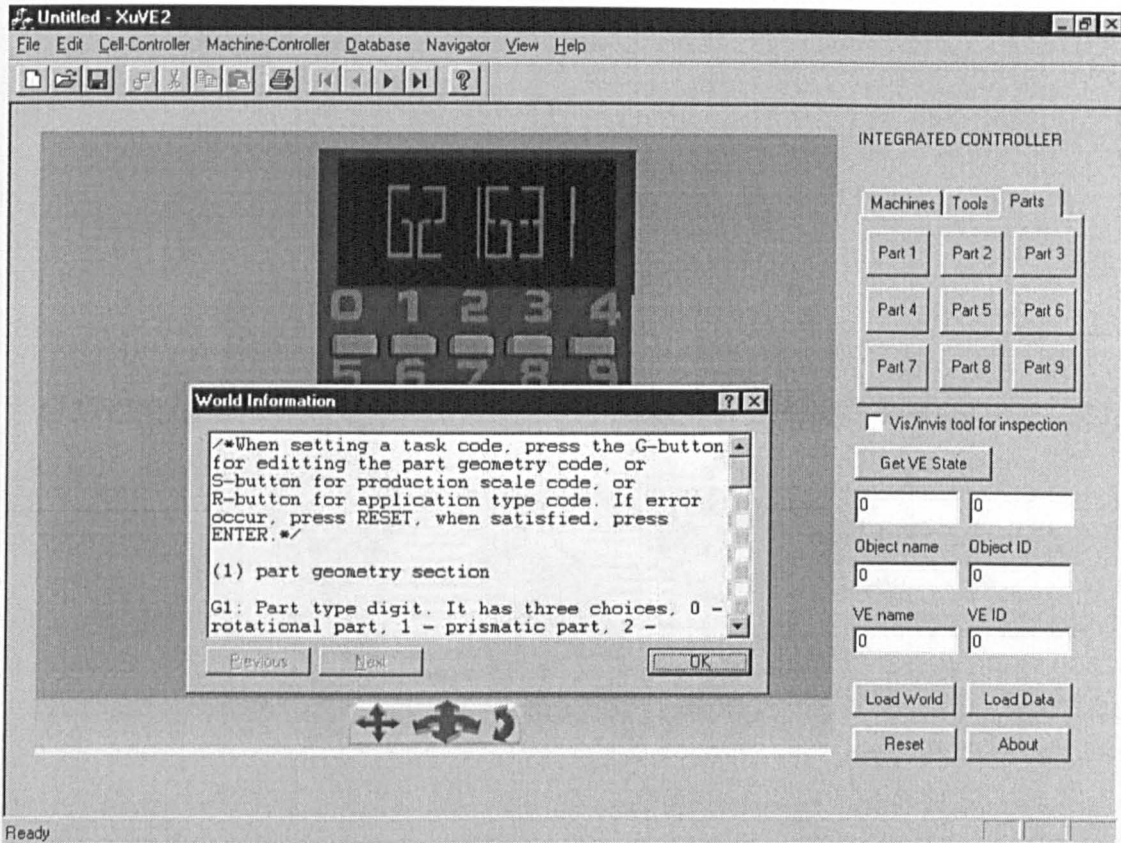


Figure 8.9 KAMVR run-time platform task coding environment

Based on the meta-code generating program in Section 4.3.1.2 and Task Classification method developed by Zhao [1998], the meta-code for retrieving an environment of the application requirements would be 03111. The closest template environment code in the database is template environment 4 with an environment code (Envcode) as 03311. The name of this environment (“workcell4”) is used to set the argument of an internal function “SetSrc(Filename)”. This function loads the environment into a visualiser. Figure 8.10 shows this environment to include a lathe, a milling machine, transportation rollers and two robots. All of those manufacturing machines and transportation devices have default simulation functions (see section 5.2 to section 5.4).

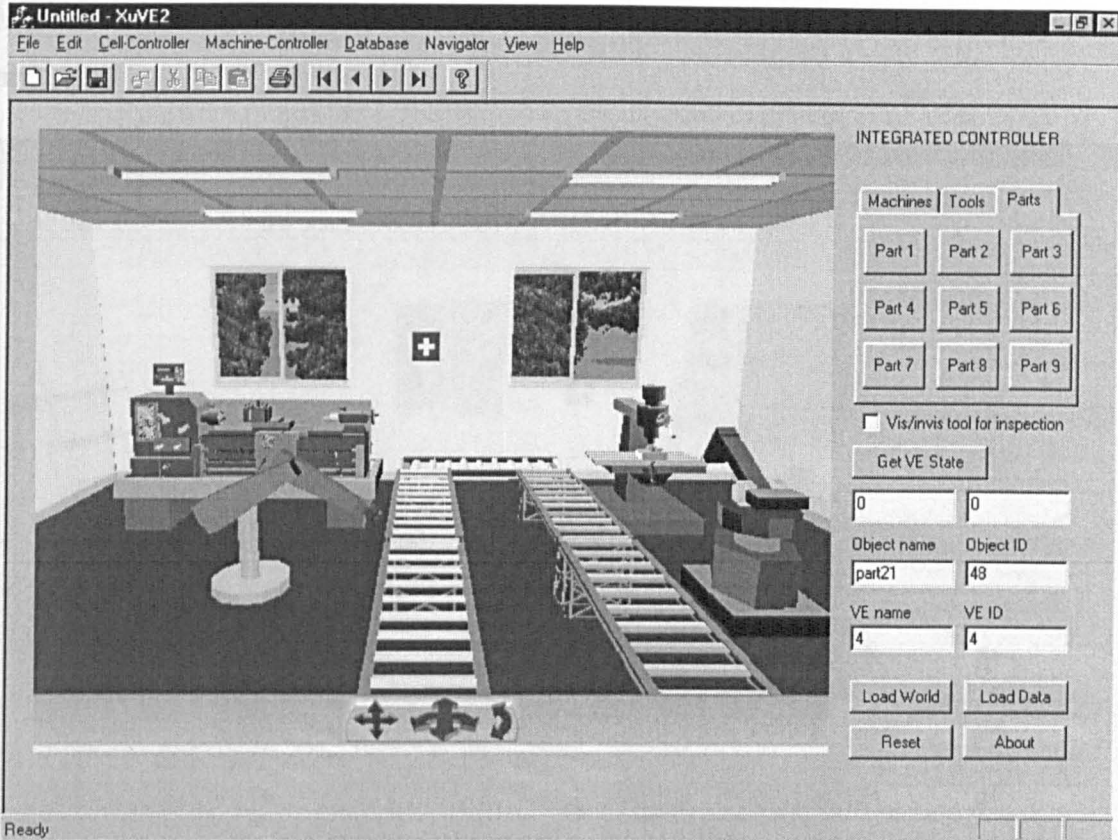


Figure 8.10 Loaded environment

8.4.2 Environment modification and initialisation

After the environment is loaded, users can explore it using the navigation bar, or observe the environment from various default viewpoints (see Section 8.2.1), and activate the default animation and simulation functions. In this way the user can quickly become familiar with the environment and its control and behaviour.

To demonstrate the data exchange function between the virtual environment and the database, the virtual lathe was changed along the X-axis 5000 steps (units) in the database. The environment simultaneously changes its layout and its overall size will be reduced by 10 percent. Figure 8.11 shows a snapshot of the modified environment. In addition to changing the environment layout and object size, the modification functions in the run-time platform also include: add, delete or duplicate objects (in this application,

the robot and the transportation roller might be removed to enhance the display frame rate).

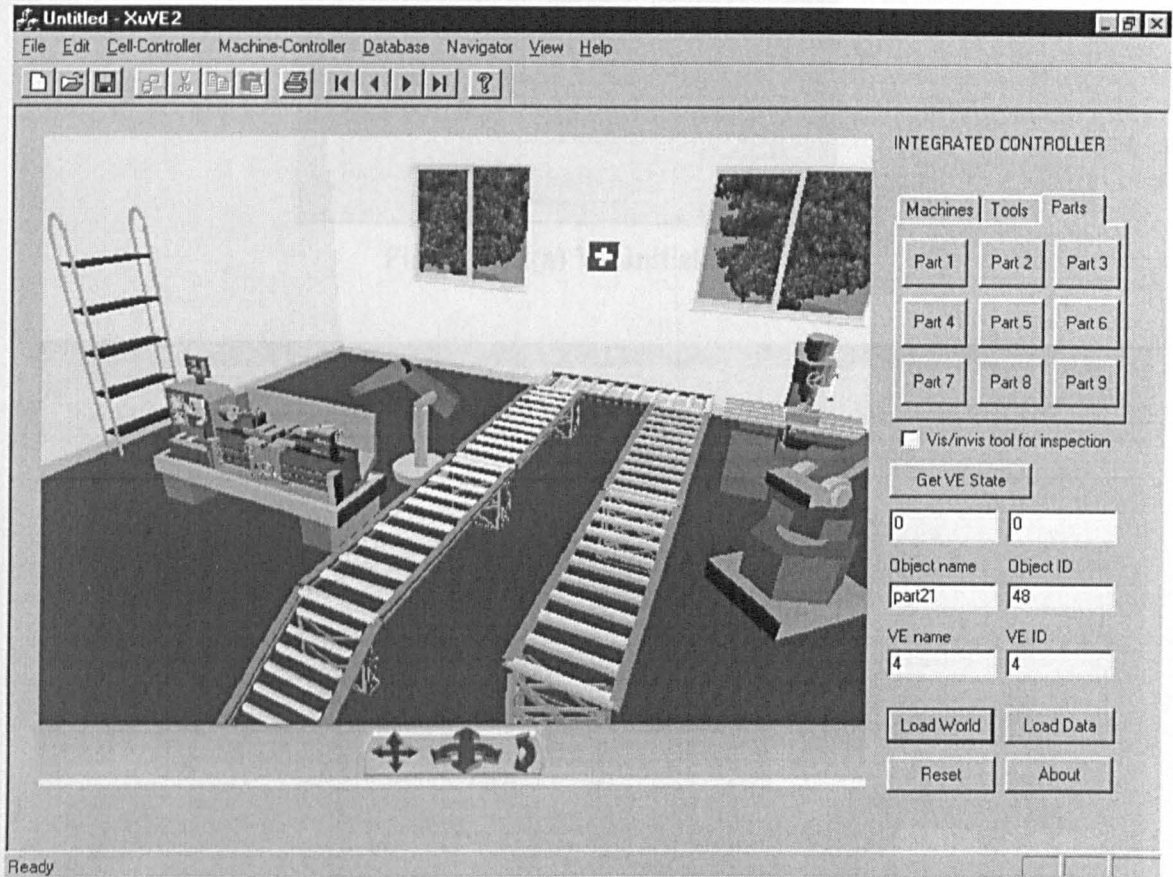


Figure 8.11 Modified virtual manufacturing environment

Before the system can be configured for a specific application and simulation, it needs to be initialised. This is done through the initialisation dialog box (activated by clicking the Initialiser menu item in the Cell-Controller menu, see Figure 8.8) shown in Figure 8.12(a). Four option buttons are available. The “Init-Pos” and the “Init-Size” buttons set objects that have initial size and initial position properties to their original values. The “Init-Trigger” and “Init-Flags” buttons clear remaining user configuration settings (from the last environment application) by resetting the animation flags, returning the dynamic objects to their home position, and cancelling all the randomised interaction triggers. To initialise the entire retrieved environment in this demonstration, all four buttons have been clicked. Figure 8.12(b) shows a snapshot of the resulting initialised environment.

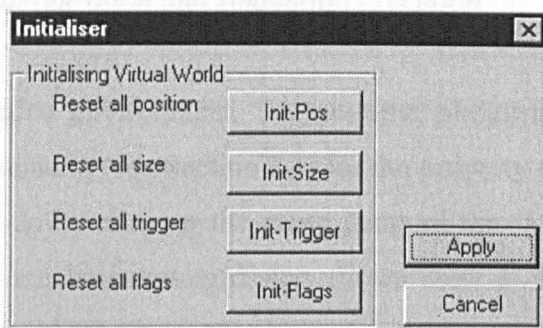


Figure 8.12(a) VE Initialiser

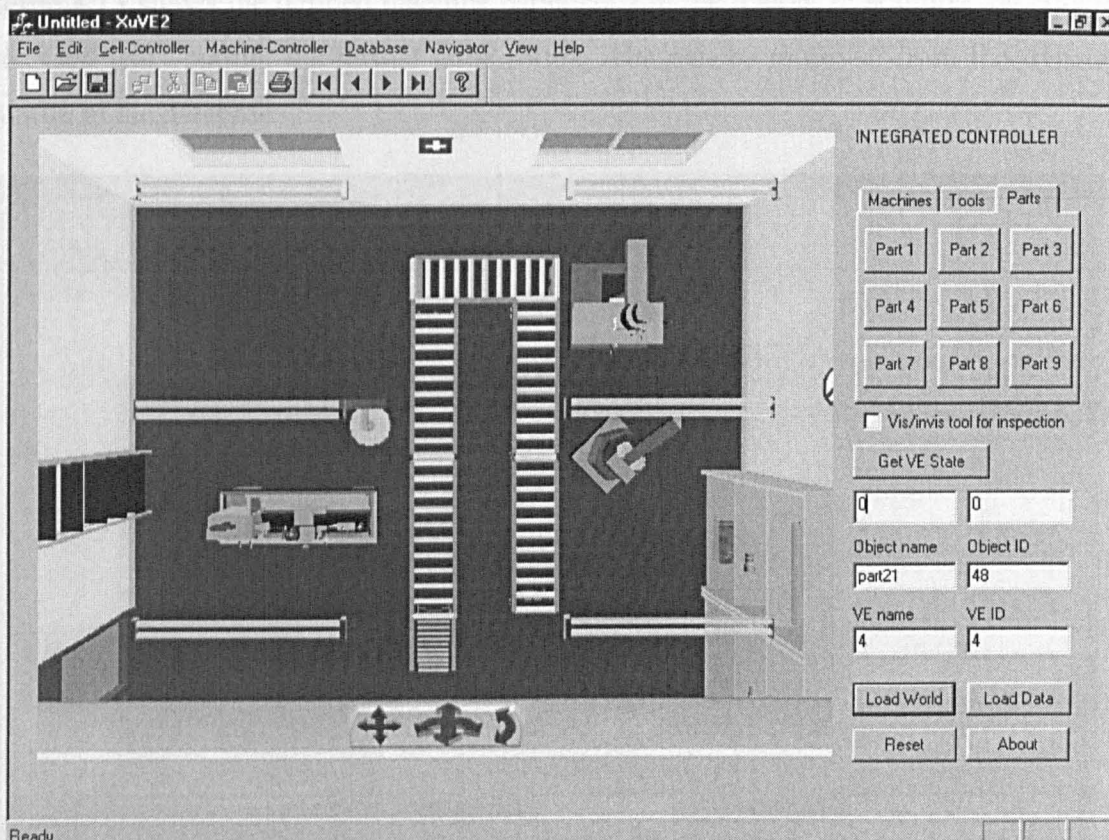


Figure 8.12(b) Initialised application environment

8.4.3 Application configuration and simulation execution

Figure 8.14 shows snapshots of changing different components in the environment. To customise the loaded environment, the run-time platform accesses the simulation functions in the environment (see Section 7.4) for the property setting. A setting dialog is activated and popped-up by clicking the menu items of the “Machine-Controller” menu (see Figure 8.7). The machine setting dialog allows users to set the machine operation parameters step by step, or load a stored set-up file in one step. In this example, the machine setting only concerns the virtual lathe, and is through a step-by-step procedure. Figure 8.13 shows the defined machine parameters in the dialog. In addition, by clicking the “GetField” button, the virtual lathe setting data can be retrieved from the dynamic datafile in the database.

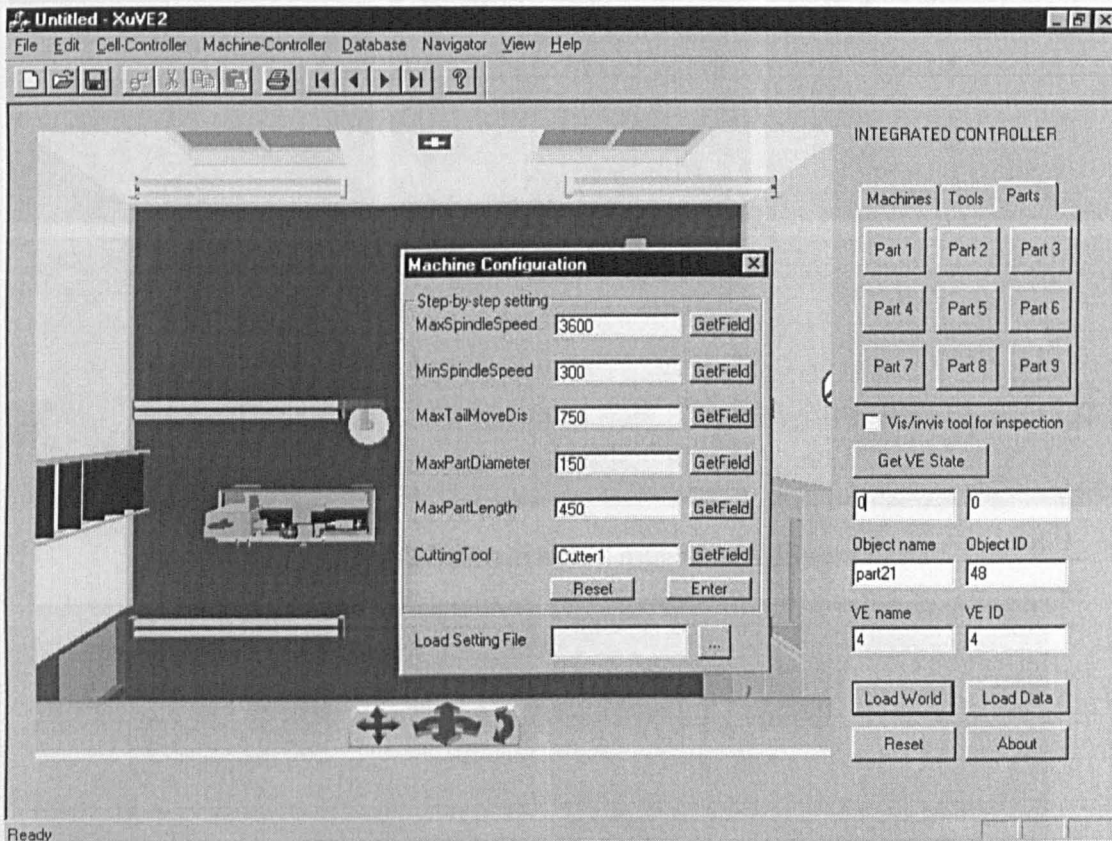


Figure 8.13 Application environment configuration

Finally, the configured environment can be used to simulate the machining activities. Figure 8.14 shows snapshots of changing different components on the lathe (by clicking the buttons on the platform named from “Part1” to “Part 9”). Figure 8.15 show snapshots of toggling the tool on a milling machine to view the processed component (by selecting the “Vis/invis” selection box).

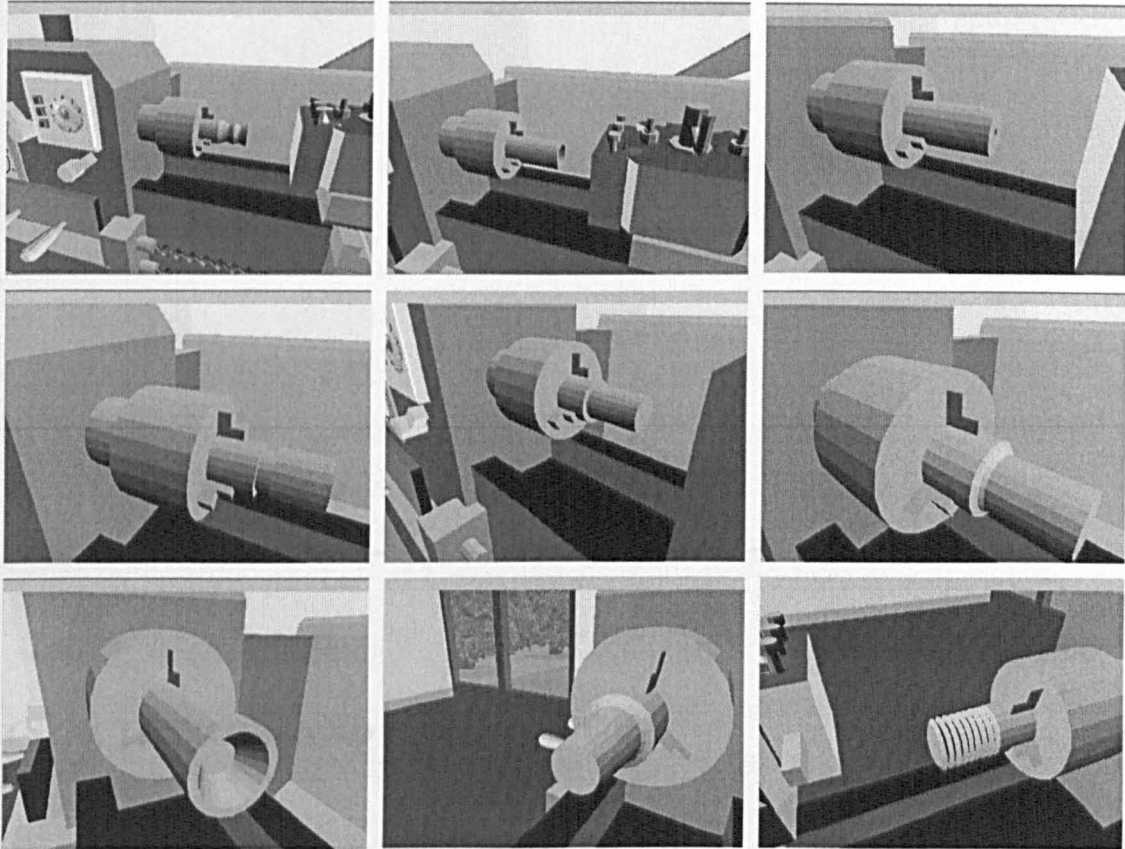


Figure 8.14 Simulation action – mounting different parts

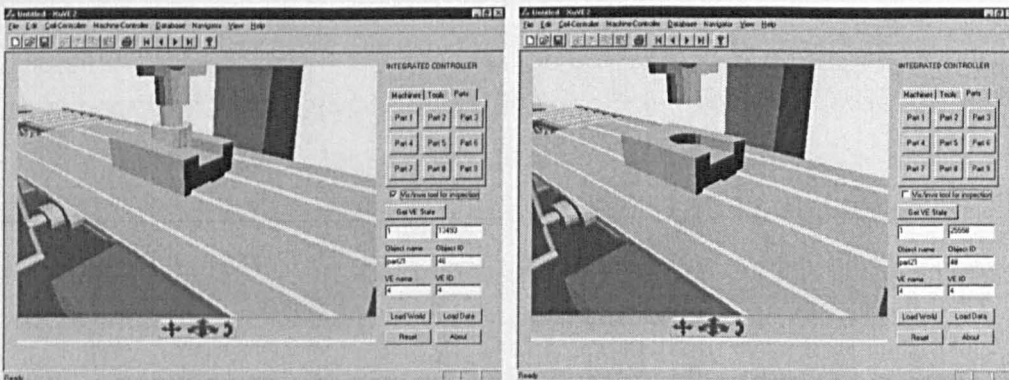


Figure 8.15 Toggle the tool for part inspection

CHAPTER 9

CONCLUSIONS AND FUTURE WORK

This chapter concludes the thesis and recommends the future research.

9.1 CONCLUSIONS

This research started with a survey of existing VR research and VR projects that are mainly related to manufacturing applications. Problems were specified for this research and thereafter a research strategy was established to deal with the difficulties that had been involved in constructing virtual environments, acquiring and representing environment knowledge and interfacing the environments with real applications.

The survey showed that the difficulties in the construction of virtual environments are most due to the so-called “bottom-up” geometric modelling process that often demands a great amount of labour-intensive computing operations and specialised programming techniques. The focuses of the end-users are restricted to this modelling process rather than the application purposes of the built environments. Hence the final built environments are inherently 3D graphical models that are lack of vital and useful application information or knowledge.

The survey also revealed that the difficulties in constructing virtual environments are closely related to the complex and indirect relationships between the geometric and behavioural attributes of the individual virtual objects within the virtual environments and the application databases. There had been a lack of a clear mapping between the information encapsulated within the virtual environments and the real knowledge defined in an application database (for example, a manufacturing database). A straightforward solution to this problem was to enforce the virtual environments by using pre-defined simulation functions or simulation programmes. This is however based on the assumption that all the attributes of the virtual objects within the virtual environments do not necessarily to be altered by the users even if the real application requirements are to be changed. Various commercial VR software and research prototype VR systems have adopted different mechanisms to provide users with reasonably less restricted access to the virtual environments, but offer the users little flexibility to configure and modify the constructed environments.

The survey also indicated that the interface between the data modelled within a virtual environment (for instant, a virtual machining cell) and the real data of an application

(for instant, a robot) is not only a challenging VR modelling problem, but also a bottleneck task for VR-based real-time simulation and control.

Based on the findings from the survey, the research had objectives that were set for fundamental solutions to the aforementioned problems. To achieve those objectives, a so-called “domain-analysis based top-down” virtual environment construction approach were established in this research. To implement and validate this approach, a system named KAMVR was designed and developed. With this system, users can construct a virtual manufacturing environment rapidly without the lengthy process of building one from scratch. The system also enables the user to reconfigure an environment and to interface its internal data with external physical systems through specific interface protocols.

The research has its novelty and contributions toward virtual environments and their applications in following aspects:

- It has provided generic solutions for the fundamental problems that currently restrict the rapid construction of complex and large scale virtual environments. This resulted in the establishment of a (application) domain-analysis based top-down construction approach. This novel approach relies on the concept of application-oriented template environments that are built in a layered structure. Such a structure allows the users to reconfigure the environment according to classified (or object-oriented) data blocks like scene graph (3D cubical volumes or so-called place-holders) and environs attributes (object static and dynamic properties classified according to their natures). All virtual environment data is managed in an environment database system and can be used to construct different virtual environments and to control specific application tasks at run-time. An application task coding-scheme was developed to establish the relationships between the application requirements and the virtual environment modelling data and to index the template environments onto manufacturing data in the database.
- The research laid a basis for integrated data management (for managing both virtual environment data and manufacturing information) in VR based

application systems. This could be further explored to establish new methods and to design new computing programmes for acquiring real manufacturing knowledge from physical manufacturing systems and representing the knowledge in virtual environments. With conventional virtual environments, representing real application data is a one-off process in which all the knowledge such as object appearance, simulation and operation interactions has to be implemented as pre-defined and pre-programmed modules and then embedded in the environments. The more knowledge an environment has, the more the computing overhead becomes and the poorer the run-time performance. The development of the environment data management in this research has enabled a precise one-to-one relationship between a virtual object model and its real world counterpart, which allows real world knowledge to be stored separately from the virtual environment data blocks (such as simulation programs and operation parameters). Currently, the research has achieved a flexible data management method, but physical information or external data sources (like user input) communicate with the virtual environments only at run-time or if necessary, through the database system, when the users are modifying or reconfiguring the virtual environments. This needs to be further researched to enable users to reconfigure the virtual environments in real-time, that is, the virtual environments and the physical systems could be communicate with each other in a close-loop in real-time.

- The research also provided an integrated software system. The functional modules of this system have enabled various phases of the domain-analysis based top-down environment construction. These modules were unified in a single run-time platform, where each module can perform its own tasks as well as communicating with other modules through a message collecting and distributing mechanism.
- Through building and testing a virtual robot and a real robot communication interface, this research has provided a new method of using VE to achieve realistic 3D simulation and real-time control for manufacturing operations.

The research in its present form has its limitations where further research is required and more development work should be provided:

- The concept of the domain-analysis top-down environment construction approach and the derived KAMVR system are based on the desk-top virtual reality systems, the application of which to an immersive virtual reality system may not be straightforward due to the structural difference of the two type of VR programmes. It could be possible to use the work provided in this research only if adequate VR peripherals devices drivers are designed as integrated parts of the data base protocols of the KAMVR system. This mostly requires a great deal of development and VR programming work rather than new research efforts.
- The coding scheme in the research has brought a method for classifying virtual environments by focusing on both the users' application requirements and the physical system characteristics. However the current scheme system only has limited digits for representing the crude but general application information. It should be ideal use specific manufacturing coding methods for specific simulation scenarios. Whether or not this is practical and achievable to apply different coding systems onto different manufacturing tasks is an issue that need further investigation.
- The KAMVR system is developed as a research prototype to verify the proposed VE construction concepts and approaches. The application of the system is generally for manufacturing simulation and training. Different users such as lectures, researchers, post- and under-graduate students have tested it in different occasions. It was also presented in seminars, national and international conferences, and journal publication, including demonstrations to industrial visitors. A general feedback from those occasions is that the future implementation of the specific system tools and functions for specific application areas is a most important part of the work to bring the system onto the users desktop.

9.2 FUTURE RESEARCH

In the KAMVR system, the environment construction data are stored in an external commercial database that in its present form communicates with the environment visualiser through the "object-linking" mode. Because of this, the data transfer are very much limited in its efficiency and capacity by the connection tools, for example, the ODBC utility. If a user wants to change the database structure during the environment run-time, the current database is inflexible for this task. An internal database system that forms part of the rapid environment construction and knowledge acquisition system would be an ideal solution. It could provide a self-defined data definition language (DDL) and data manipulation language (DML) specifically designed to handle environment data with minimum environment construction and visualisation time.

The environment task-coding scheme is fundamental part for developing an useful virtual environment for a specific application. The future research for this part is to take the advantage of the parallelism between the GT code for simulation models and the GT code for manufactured parts. Given the GT code of a part, the simulation environment should be able to suggest alternative manufacturing methods and a complete simulation scenario (based on various manufacturing criteria such as cost and shop floor load). That will provide a guideline for system analysis and conceptual model development by systematically identifying user goals and physical system characteristics.

Further research is also necessary on the development of converting tools for translating various external data and knowledge sources (for example, the "if – then" rule base) into a unified environment data format. This could overcome the difficulties in exchanging the VRML data from, say, ProE CAD model to the Superscape VRT Visualiser and to prevent the data loss and data distortion. Media exchange utilities could be developed to support importing multimedia data such as sound, image and animation into the database.

The current real world communication provided by the KAMVR system is mainly serial port based, even it has the highest data transfer the exchange rate is low. The

multiple physical devices that are connected with the VE to the hosting computer rely on a software program to control the port allocation and file operations like open, close, read, and write. The synchronisation between the VE and the physical devices operations was found to be a difficult problem. A more sophisticated approach could be to use ATM and ETHERNET via TCP/IP format communication to extend the ability of the KAMVR system.

The on-line KAMVR system could also allow several users to interact in the same environment while at different locations. The research for this purpose needs to be extended to the use of distributed database systems and other methods of retrieving data for environment construction and configuration.

Finally KAMVR has been implemented with a data glove based interaction and stereo sound effects. It is anticipated that using full-immersive VR techniques would provide more realistic effects. For example, with HMD and haptic devices, the environment operation interface could be improved by avoiding any programming based object control skills. Also with 3D sound effects, it could provide multimedia information to aid users explore unfamiliar sound-based environments. These functions, however, demands extensive low-level programming work to establish the peripheral drivers that in turn need to be integrated with the KAMVR system.

References

- 3D Web Consortium Incorporated, 1997. The Virtual Reality Modelling Language (VRML'97). *International Standard ISO/IEC 14772-1:1997*.
- Adam, J., Zikan, L., Karel, M., and Banner, D., 1995. Videometric Head Tracker for Augmented Reality Applications. *Proceedings of SPIE – The International Society for Optical Engineering*, Bellingham, WA, USA.
- Adiga, S., and Glassey, C., R., 1991. Object-oriented simulation to support research in manufacturing systems. *International Journal of Computer Integrated Manufacturing*, 29(12), pp2529-2542.
- Angster, S. R., Gowda, S., Jayaram, S., 1994. Feasibility Study for Ergonomic Design. *IFIP 5.0 Workshop on virtual prototyping*.
- Angster, S. R., 1996. VEDAM: Virtual Environments for Design and Manufacturing. *Ph.D. Dissertation*, Washington State University, USA.
- Aouad, G., Child, T., Marir, F., Brandon, P., 1997. Developing a Virtual Reality Interface for an Integrated Project Database Environment. *Proceedings of the 1997 IEEE Conference on Information Visualisation*, Ch.50, pp.192-197, ISBN 0-81-868076-8.
- Banerjee, P., Montreuil, B., Moodie, C. L. and Kashyap, R. L., 1992. A Modelling of Interactive Facilities Layout Designer Reasoning Qualitative Patterns. *International Journal of Production Research*, Vol. 30, No. 3, pp433-453.
- Banerjee, A., Banerjee, P., and Ye, N., 1999. Assembly Planning Effectiveness Using Virtual Reality. *Presence-teleoperators and Virtual Environments*, Vol.8, No.2, pp.204-217 ISBN 1054-7460.
- Barnes, M., 1996. Virtual Reality and Simulation. *Proceedings of the 1996 Winter Simulation Conference*, Ch.213, pp101-110, ISBN 0-78-033383-7.
- Barrus, J. W., 1994. The Virtual Workshop: A Simulated Environment for Mechanical Design. *Ph.D. Dissertation*, Massachusetts Institute of Technology.
- Bejczy, A.K., 1997. Virtual Reality in Manufacturing. *Re-Engineering for Sustainable Industrial Production*, Ch.46, pp48-60, ISBN 0-41-279950-2.
- Bennett, G. R., 1997. The Application of Virtual Prototyping in the Development of Complex Aerospace Products. *Aircraft Engineering and Aerospace Technology*,

Vol.69, No.1, pp19-25, ISSN 0002-2667.

- Blanchard, C., Burgess, S., Harvill, Y., Lanier, J., Lasko, A., Oberman, M., 1990. Reality built for two: A virtual reality tool. *Proceedings of the 1990 ACM Symposium on Interactive 3D Graphics*, pp35-36.
- Boman, D. K., 1995. International survey: virtual-environment research. *Computer*, IEEE Computer Society, pp57-65.
- Bricken, W., CoCo, G., 1994. The Veos Project. *Presence: Teleoperators and Virtual Environments*, MIT Press, Boston, 3(2), 111-129.
- Bullinger, H. J., and Roessler, A., 1998. Advances in Bridging the Gap: Using Virtual Reality to Enhance Productivity. *Virtual Environments '98 – Proceedings of the Eurographics Workshop*, pp1-7, ISBN 3-211-83233-5.
- Chapman, D. A. and Coddington, R. C., 1994. Using a Virtual Reality System For Machine Design. *Proceedings of The 5th International Conference of Computers in Agriculture*, Ch.150, pp87-92, ISBN 0-92-935546-6.
- Charitos, D. and Rutherford, P., 1996. Guidelines for the Design of Virtual Environments. *The proceeding of UK-VRSIG'3*.
- Chuter, J. C., Ramaswamy, S., and Barber, K. S., 1995. A Virtual Environment For Construction and Analysis of Manufacturing Prototypes. *Proceedings of the Computers in Engineering Conference*.
- Codella, C., Jalili, R., Koved, L. and B. J., 1993. A Toolkit for Developing Multi-User, Distributed Virtual Environments. *Proceedings of the IEEE Virtual Reality Annual International Symposium (VRAIS'93)*, Seattle, Washinton, USA.
- Connacher, H. I., Jayaram, S., 1995. Virtual Assembly Design Environment. *Proceedings of the Computers in Engineering Conference and the Engineering Database Symposium*, ASME 1995, pp875-885.
- Cook, J., Hubbard, R., and Keates, M., 1998. Virtual Reality for Large-Scale Industrial Applications. *Future Generation Computer Systems*, Vol.14, pp157-166.
- Earnshaw, R. A., Gigante, M. A. and Jones, H., 1993. Virtual Reality System. *Academic Press*, ISBN 0-12-227748-1.
- Evans, D. F., Bartelme, M. J., and Iwai, N., 1994. Essential Methods and Emerging Concepts for Developing Virtual Driving Environments. *Proceedings of the IMAGE VII Conference*, Image Society, Tempe, Arizona, pp121-131
- Gausemeier, J., Grafe, M., Wortmann, R., 1998. Layout of Manufacturing Systems with

- VR-based Construction Sets. *Proceedings of the - Virtual Environments '98 - Eurographics Workshop, Springer Computer Science*, ISBN 3-211-83233-5.
- Gimenez, A.M., and Kirner, T.G., 1997. Validation of Real Time Systems Using a Virtual Reality Simulation Tool. *Proceedings of the 1997 IEEE International Conference on Systems, Man, and Cybernetics*, Vol 5, Ch.790, pp1586-1591, ISBN 0-78-034054-x.
- Gray, J.C., 1997. Virtual Reality and Operational Design. *Visions of Tomorrow-Improving the Quality of Life Through Technology*, Vol.1997, Ch.29, No.3, pp185-190, ISBN 1-86-058098-x.
- Hirota, K. and Hirose, M., 1995, Simulation and Presentation of Curved Surface in Virtual Reality Environment Through Surface Display. *Proceedings of the Virtual Reality Annual International Symposium '95*, Ch,29, pp211-216, ISBN 0-81-867084-3.
- Hollands, R.J. and Mort, N., 1994. Manufacturing Systems Simulation Mixed Mode and Virtual Reality Simulation. *Fourth International Conference on Factory 2000-Advanced Factory Automation*, Ch.99, No.398, pp651-657, ISBN 0-85296626-1.
- Iuliano, M., and Jones, A., 1996. Controlling Activities in a Virtual Manufacturing Cell. *Proceedings of the 1996 Winter Simulation Conference*, Ch.213, pp1062-1067, ISBN 0-78-033383-7.
- Jones, K. C. 1993. Virtual Reality for Manufacturing Simulation. *Proceedings of the Winter Simulation Conference*, New York, USA, pp883-887.
- Jones, A., and Iuliano, M., 1997. A Virtual Manufacturing Testbed. *Proceedings of the 1997 IEEE International Conference on Systems, Man, and Cybernetics*, Vols.1-5, Ch.790, pp737-742, ISBN 0-78-034054-x.
- Jones, P. 1999. Three-Dimensional Input Device with Six Degrees of Freedom. *MECHATRONICS*, Vol.9, No.7, pp717 - 729.
- Karacali, O., 1995. Towards User Oriented Object Modelling in Virtual Reality, *Proceedings of the IEEE Southeast CON'95-Visualize The Future*, Ch.91, pp454-460, ISBN 0-78-032642-3.
- Kerttula, M., and Salmela, M., and Heikkinen, M., 1997. Virtual Reality Prototyping - a Framework for the Development of Electronics and Telecommunication Products. *Proceedings of the 8th IEEE International Workshop on Rapid System Prototyping*, Ch.22, pp2-11, ISBN 0-81-868064-4.
- Korves, B., and Loftus, M., 1999. The Application of Immersive Virtual Reality for Layout Planning of Manufacturing Cells. *Journal of Engineering Manufacture*,

Vol.213, No.1, pp87-91, ISSN 0954-4054.

Lee, K.I., and Noh, S.D., 1997. Virtual Manufacturing System - A Test Bed of Engineering Activities. *CIRP Annals 1997 Manufacturing Technology*, Vol.46/1/1997, Ch.114, pp347-350, ISBN 3-90-527727-1.

Loo, P. L., 1991. The Starship Manual - Version 2.0. *ISS TR91-54-0*. Institute of System Science, National University of Singapore, Singapore.

Macedonia, M. R., Zyda, M. J., Pratt, D. R., Barham, P. T. and Zeswitz, S., 1995. NPSNET: A Networksoftware Architecture for Large Scale Virtual Enviornments. *Presence: Teleoperators and Virtual Environments*, MIT Press, Boston, 3(4).

Massie, T., and Salisbury, J., 1994. The PHANToM Haptic Interface: A Device for Probing Virtual Objects. *Proceedings of the ASME Winter Annual Meeting, Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*.

Matsuda, M., and Kimura, F., 1997. Generation of Milling Data in a Virtual Manufacturing Framework. *Information Infrastructure Systems for Manufacturing*, Ch.39, pp277-288, ISBN 0-41-278800-4.

Maxfield, J., Fernando, T. and Dew, P., 1995. A Distributed Virtual Environment for Concurrent Engineering. *Virtual Reality Annual International Symposium'95*, ISBN 0-8186-7084-3/95.

Mckenna, M., and Zeltzer, D., 1992. Three Dimensional Visual Display Systems for Virtual Environments. *Presence*, Vol.1, pp421-458.

McNeely, Burdea, W., Hannaford, G., and Hirose, B., 1995. Whither Force Feedback. *IEEE Annual Virtual Reality International Symposium*, Los Alamitos, CA, USA.

Meyer, K., Applewhite, H. L., and Biocca, F. A., 1992. A Survey of Position Trackers. *Presence*, Vol.1, No.2, pp73-200.

Mironov, S., 1998. Simulation of an Intelligent Objects Behaviour in a Virtual Reality System. *New Approaches to High-Tech Materials: Non-destructive Testing and Computer Simulations in Materials Science and Engineering*, Vol.3345, Ch.54, pp285-288, ISBN 0-81-942792-6.

Mizel, O., and David W, 1994. Virtual Reality and Augmented Reality in Aircraft Design and Manufacturing. *Wescon Conference Record*, Wescon, Los Angeles, CA, USA.

MultiGen: SmartScene Version 1.1, 1997 Release Notes, MultiGen Inc., San Jose, USA, 1997.

Narayanan, S., and Bodner, D. A., 1997. Research in Object-Oriented Manufacturing

- Simulations: An Assessment of the State of the Art. *IIE Transaction*.
- Neira, C., Sadin, C., and Daniel, J., 1992. Cave. Audio Visual Experience Automatic Virtual Environment. *Communications of the ACM*, Vol.35, No. 6, pp65-72.
- NSF, 1992. US Government's National Science Foundation. *Computer Graphics*, Vol.26, No.3, ACM SIGGRAPH.
- Orady, E.A., and Osman, T.A., Bailo, C.P., 1997. Virtual Reality Software for Robotics and Manufacturing Cell Simulation. *Computers & Industrial Engineering*, Vol.33, No.1-2, pp87-90, ISSN 0360-8352.
- Ozdemirel, N. E., Mackulak, G. T., and Cochran, J. K., 1993. A Group Technology Classification and Coding Scheme for Discrete Manufacturing Simulation Models. *International Journal of Computer Integrated Manufacturing*, 31(3), pp579-601.
- Polis, M., Gifford, S. and McKeown, D., 1995. Automating the Construction of Large-Scale Virtual Worlds. *1995 IEEE Computer*, pp 57-65, ISSN 0018-9162.
- Rajamani, D., 1993. Classification and Coding of Components for Implementing a Computerised Inventory Control-System for a Television Assembling Industry. *International Journal of Production Economics*, Vol.32, No.2, pp133-154 ISSN 0925-5273.
- Rheingold, H., 1992, *Virtual Reality*, ISBN: 0749308893, MANDARIN, London.
- Ritchie, J. M., Dewar, R. G., and Simons, J. E. L., 1999. The Generation and Practical Use of Plans for Manual Assembly Using Immersive Virtual Reality. *Proceedings of the Institution of Mechanical Engineers*, Vol.213, No.5, pp461-474, ISBN 0954-4054.
- Rosen, D. W., Bras, B., Mistree, F., Goel, A., 1995. Virtual Prototyping for Product Demanufacture and Service Using a Virtual Design Studio Approach. *ACME Computers in Engineering Conference*.
- Roussos, M., Johnson, A., Moher, T., Leigh, J., Vasilakis, C., and Barnes, C., 1999. Learning and Building Together in an Immersive Virtual World. *Presence-Teleoperators and Virtual Environments*, 1999, Vol.8, No.3, pp247-263.
- Sequeira, V., Ng, K., Wolfart, E., Goncalves, J. G. M., and Hogg, D., 1999. Automated Reconstruction of 3D Models from Real Environments. *Isprs Journal of Photo Grammetry and Remote Sensing*, Vol.54, No.1, pp.1-22 ISSN 0924-2716.
- Shaw, C., Green, M., Liang, J., and Sun, Y., 1993. Decoupled Simulation in Virtual Reality with the MR Toolkit. *ACM Transactions on Information Systems*, 11(3), pp287-317.

- Singh, G., Nordhausen, B., Bhonsle, S., and Thennarangam, S., 1993. A Software Toolkit for Network-Based Virtual Worlds. *Proceedings of the InterCHI'93 Research Symposium*, Amsterdam, Netherland.
- Singh, G., Serra, L., Fairchild, K., and Poston, T., 1994. Visual Creation of Virtual Design Environments and Virtual Worlds Research at ISS. *Presence*, Vol.3, No.1.
- Singh, G., Serra, L., Png, W., & Ng, H., 1994. BrickNet: A Software Toolkit for Network-Based Virtual Worlds. *Presence: Teleoperators and Virtual Environments*, 3(1), pp19-34.
- Singh, G., Serra, L., Png, W., Wang, A., and Ng, H., 1995. BrickNet: Sharing Object Behaviours on the NET. *Proceeding of IEEE Virtual Reality Annual International Symposium (VRAIS'95)*, pp19-25, ISBN 0-8186-7084-3.
- Smith, R. P., and Heim, J. A., 1999. Virtual Facility Layout Design: the Value of an Interactive Three-Dimensional Representation. *International Journal of Production Research*, Vol.37, No.17, pp3941-2957.
- Sowizral, H., Barnes, A., and James, C., 1993. Tracking Position and Orientation in a Large Volume. *1993 IEEE Annual Virtual Reality International Symposium*, IEEE Service Center, Piscataway, NJ, USA.
- Stampe, D., Roehl, B., and Eagan, J., 1993. *Virtual Reality Creations: Explore, Manipulate, and Create Virtual Worlds on Your PC*, The Waite Group, Inc., Waite Group Press, California.
- Sun, W. and Clapworthy, G. J., 1996. Modelling Figure Animation for Virtual Environments. *Proceedings of the UK-VRSIG'3*.
- Takalo, J., Ensomaa, J., and Plomp, J., 1998. A Hierarchical Virtual Environment for a Machine Fault Diagnostic Application. *Future Generation Computer Systems*, Vol.14, No.3-4, pp179-184.
- Taylor, R., Bayliss, G., Bowyer, A., and Willis, P., 1995. A Virtual Workshop For Design By Manufacture. *15th ASME International Computers in Engineering Conference*, Boston USA.
- Tian, B., Zhu, R. M., Wang, Q. T., and Dai G. Z., 1997. Message-Driven Object-Oriented Programming: A Promising Solution to Virtual Reality Construction. *1997 IEEE International Conference on Intelligent Processing Systems*, Beijing, China.
- Trika, S. N., Banerjee, P., and Kashyap, R. L., 1997. Virtual Reality Interface for Feature-Based Computer-Aided Design Systems. *Computer-Aided Design*, Vol.29, No.8, pp565-574.

- Turner, R., Li, S., and Gobbetti, E., 1999. Metis – an Object-Oriented Toolkit for Constructing Virtual Reality Applications. *Computer Graphics Forum*, Vol.18, No.2, pp122-130.
- Ülgen, O. M., and Thomasma, T., 1990. SmartSim: An Object Oriented simulation program generator for manufacturing systems. *International Journal of Computer Integrated Manufacturing*, 28(9), pp1713-1730.
- Vance, J. M., 1996. Virtual Reality: What Potential Does it Hold for Engineering? *Proceedings of the International Conference on Mechanical Design and Production*. Vol.2.
- Vince, J., 1995. *Virtual Reality System*, ISBN 0-201-87687-6, Addison-Wesley Publishing Company.
- Viswanadham, N. and Narahari, Y., 1992. *Performance Modelling of Automated Manufacturing System*, ISBN 0-13-658824-7.
- VRT Manual, 1997. Superscape Co.Ltd, UK.
- Walczak, K., 1996. Integration of Virtual Reality and Multimedia Data in Databases. *Proceedings of the International Workshop on Multimedia Database Management Systems*, Ch.21, pp80-84, ISBN 0-81-867469-5.
- Wang, Q. J., Green, M. and Shaw, C., 1995. EM- An Environment Manager for Building Networked Virtual Environments. *Virtual Reality Annual International Symposium'95*, ISBN 0-8186-7084-3/95.
- Watanabe, H., and Ohashi, K., and Takahashi, N., and Kato, K., Fujita, S., 1997. Virtual Manufacturing Workcell for Engineering. *Architectures, Networks, and Intelligent Systems for Manufacturing Integration*, Vol.3203, Ch.22, pp116-124, ISBN 0-81-942635-0
- West, A. J., Howard, T. L. J., Hubbard, R. J., Murta, A. D., Snowdon D. N., and Butler, D. A., 1992. AVIARY - A Generic Virtual Reality Interface for Real Applications. *Proceedings of the Virtual Reality Systems Conference*, (also published in *Virtual Reality Systems*, R.A. Earnshaw, M.A. Gigante and H. Jones (Eds), Academic Press, 1993, pp 213-236.)
- Wilson, J. R., Cubb, S., Cruz, M. D. and Eastgate, R., 1996. *Virtual Reality for Industrial Application Opportunities and limitations*, Nottingham University Press. ISBN 1-897676-573.
- WorldUp User's Guide, version 4, Sense8 Co.Ltd.
- WorldToolkit Reference Manua l, version 7, Sense8 Co.Ltd.

- Xiao, P., Li. G., Yang, M., 1997. The Application of Virtual Reality to Product Development of Automobile Parts and Spares. *1997 IEEE International Conference on Intelligent Processing Systems*, Vol.1 and 2, Ch.412, pp1752-1755.
- Young, P., 1996. Virtual Reality Systems, Survey of VR and VRML Systems.
<http://www.dcs.ed.ac.uk/~dcs3py/pages/work/documents/VR-survey/index.html>,
University of Durham.
- Zetu, D., Schneider, P., and Banerjee, P., 1997. A Fast Data Input Model For Virtual Reality-Aided Factory Layout And Material Handling Decision Support. *1997 ASME Computer in Engineering Conference*, Sacramento, California.
- Zhao, Z. X, 1997. Constructing and Managing Complex Virtual Worlds for Manufacturing Applications. *Proceedings of UKVR-SIG Annual Conference*, Brunel University, UK.
- Zhao, Z. X, 1998. A Variant Approach to Constructing and Managing Virtual Manufacturing Environments. *International Journal of Computer Integrated Manufacturing*, Vol.11, No.6, pp485-499.

Glossary of Terms

6DOF: Six Degrees of Freedom. Ability to move in three spatial directions and orient about three axes passing through the center of the object.

absolute values: Position and orientation within a virtual space as measured from a single, constant point of origin – the virtual universe.

accelerator: Specialized hardware that increases the speed of graphics calculations.

ambient light: Naturally occurring illumination arising from outside the apparatus.

animation: recorded sequence of object activities.

API: Application Programmers Interface.

articulation: Objects composed of several parts that are separably moveable.

artificial reality: Simulated spaces created from a combination of computer and video systems (also called virtual reality).

augmented reality: The use of transparent glasses on which a computer displays data so that the viewer can simultaneously view computer generated and real world scenes.

autonomy: Performance or action of the object on the rule of physics, biology, or a virtual world, but not by independent decision of a human operator.

backdrop: The stationary background in a virtual world. The boundary of the world which cannot be moved or broken into smaller elements.

BOOM: Binocular Omni-Orientation Monitor. A 3-D display device suspended from a weighted boom that can swivel freely so the viewer can use the device by bringing the device up to the eyes and viewing the 3-D environment while holding it. The boom's position and orientation communicates the user's point of view to the computer.

bounding Box: Also known as Extents Box (smallest box that surrounds an object). The term bounding box sometimes refers to the fact that extents boxes can be made visible in the scene.

Browser: Indexes, lists or animated maps, to provide a means of navigating through the physical, temporal, and conceptual elements of a virtual world.

CAVE: VR world projected on the walls and ceiling of a room to give the illusion of immersion.

child object: A scene graph node that is a direct descendent of another (parent) node.

collision detection: Intersection testing of objects at either the bounding box level or at the polygon level.

coordinates: A set of data values that determine the location of a point in a space. The number of coordinates corresponds to the dimensionality of the space.

coordinate system: A positional system, containing X, Y, and Z components, by which three-dimensional entities can be described.

culling: Removing invisible pieces of geometry and only sending potentially visible geometry to the graphics subsystem. Simple culling rejects entire objects not in the view. More complex systems take into account occlusion of some objects by others, e.g. a building hiding trees behind it.

data glove: A glove wired with sensors and connected to a computer system for gesture recognition and navigation through a virtual environment. Known generically as a "wired glove."

data source: an instance of a database management system.

depth cueing: Use of shading, texture, color, interposition, or other visual characteristics to provide a cue for the distance of an object from the observer.

dynamics: The rules that govern all actions and behaviors within the environment.

environment: In VR terms, this is a computer-generated model that can be experienced by an observer as if it were a place.

force feedback: An output device that transmits pressure, force or vibrations to provide the VR participant with the sense of resisting force, typically to weight or inertia. This is in contrast to tactile feedback, which simulates sensation applied to the skin.

function: A scheme used in event-driven programs where the program registers a callback handler for a certain event. The program does not call the handler directly but when the event occurs, the handler is called, possibly with arguments describing the event.

geometry: A collection of polygons (composed of vertices) used to model physical objects.

group node: A scene graph node that has children, but no other properties.

GUI: Graphical User Interface.

haptic interfaces: Use of physical sensors to provide users with a sense of touch at the skin level, and force feedback information from muscles and joints.

head tracking: Monitoring the position and orientation of the head through various tracking device.

hidden surface: A surface of a graphics object that is occluded from view by intervening objects.

hierarchy: Used in the context of scene graphs, hierarchy refers to how the nodes in a scene graph are organized and the relationship of one node to another.

HMD: Head Mounted Display. A set of goggles or a helmet with tiny monitors in front of each eye to generate images seen by the wearer as three-dimensional. Often the HMD is combined with a head tracker so that the images displayed in the HMD change as the head moves.

immersion: The observer's emotional reaction to the virtual world as being part of it.

interface: A set of devices, software, and techniques that connect computers with people to perform tasks.

inverse kinematics: A specification of the motion of dynamic systems from properties of their joints and extensions.

KAMVR: A computing system developed in the University of Derby for dealing with virtual environment-based knowledge, its acquisition and management.

latency: Lag between user motion and tracker system response, sometimes measured in from as. Delay between actual change in position and reflection by the program. Delayed response time.

LCD: Liquid Crystal Display. Display devices that use bipolar films sandwiched between this panes of glass. They are lightweight and transmissive or reflective, and are often used in HMDs.

LOD: Level-of-detail. A model of a particular resolution among a series of models of the same object. Greater graphic performance can be obtained by using a lower LOD when the object occupies fewer pixels on the screen or is not in a region of significant interest.

matrix: An array of data used as mathematical entity for representing position and orientation in 3D space.

model: A computer-generated simulation physical things or events.

navigation: Purposeful motion through virtual space.

objects: Discrete 3-D shapes within the virtual world that a user can interact with.

ODBC: Open Database Connectivity. A programming utility for linking applications with various database management system.

perspective: The rules that determine the relative size of objects on a flat viewing surface to give the perception of depth.

pitch: The angular displacement of the lateral axis about a horizontal axis perpendicular to the lateral axis.

polygon: A display element that consists of an area enclosed by a set of by a set of broken straight lines.

presence: A feeling of being immersed in an environment, able to interact with object there. A defining characteristic of a VR system.

real time: Action taking place with no perceptible or significant delay after the input that initiates the action.

rendering: Generation of a graphical image from mathematical models of three-dimensional objects, i.e. a scene.

roll: Angular displacement about the lateral axis.

rotation: The turning of an object so that it has a different orientation.

scene: The virtual world being displayed.

scene graph: A scene graph is a hierarchical arrangement of nodes (such as geometry, light, fog, and positional information) representing objects in a simulation. A universe can contain more than one scene graph.

scene graph tree The scene graph is arranged as an upside down tree, where the root is on the top and the branches and leaves are on the bottom.

SDK: Superscape Developers Kit, a C function library for API programming.

sibling object: Children of the same parent node are siblings.

spatial navigation: Self-orientation and locomotion in virtual worlds.

SQL: Structured Query Language. A specialized programming language for sending queries to databases. Most industrial-strength and many smaller database applications can be addressed using SQL. Each specific application will have its own version of SQL implementing features unique to that application, but all SQL-capable databases support a common subset of SQL.

tactile displays: Devices that provide tactile and kinesthetic sensations.

telemanipulation: Robotic control of distant objects.

teleoperator: Person doing telemanipulation.

telepresence: Remote control with adequate sensory data to give the illusion of being at that remote location.

tracker: A device that provides numeric coordinates to identify the current position and/or orientation of an object or user in real space.

universe: The collection of all entities and the space they are embedded in for a VR world.

viewpoints: Points from which raytracing and geometry creation occurs. The geometric eye point of the simulation.

virtual environments: Realistic simulations of interactive scenes.

virtual prototype: Simulation of an intended design or product to illustrate the characteristics before actual construction. Usually used as an exploratory tool for developers or as a communications prop for persons reviewing proposed designs.

virtual reality. A computer system used to create an artificial world in which the user has the impression of being in that world with the ability to navigate through the world and manipulate objects in the world.

virtual world: Whole virtual environment or universe within a given simulation.

visualisation: The ability to graphically represent abstract data that would normally appear as text and numbers on a computer.

VRT: 3D authoring studio from Superscape Co.Ltd. for virtual environment editing.

WAN: Wide Area Network. Any *internet* or *network* that covers an area larger than a single building or campus.

WWW: World wide Web. A hypermedia system that allows you to browse through lots of information using a browser. It is a preferred method of presenting and accessing information on the Internet.

yaw: The angular displacement about the vertical axis.

Appendix A: KAMVR task coding scheme

- G1: Describes the type of key virtual object, which can have one of three values 0, 1 and 2. 0 is for a rotational object, 1 is for a prismatic object, and 2 is for a complex shape.
- G2: For a rotational object, it represents the length-diameter ratio (L/D). It has 4 options, 0 for L/D ratio less than 1, 1 for L/D ratio greater than 1 and less than 15, 2 for L/D ratio greater than 15 and less than 50, and 3 for L/D ratio greater than 50. If the object is a prismatic or complex shape, the digit represents the shape, 0 for a cube, 1 for a cuboid, 2 for a composite, or 3 for a triangular shape.
- G3: Defines the length of the part. It has 12 options, 0 for length of less than 40mm, 1 for greater than 40mm and less than 80 mm, until B (Hexadecimal) for less than 2900mm.
- G4: Defines features of the object. If the object is rotational, the digit has 4 options, 0 for no features, 1 for a stepped shaft, 2 for a pocket, 3 for a hole. For a non-rotational object, 0 for step milling, 1 for slot milling, 2 for a through hole, 3 for a blind hole.
- G5: Defines the material, 0 for metal or 1 for non-metal.
- S1: Defines the scale of a virtual manufacturing environment, where 0 stands for large scale and 1 for small scale.
- S2: Provides details description of the environment. Depending on the scale digit, if it is a large scale environment, then 0 is for object quantities greater than 10000, 1 is for less than 10000. Otherwise, 0 for greater than 10, 1 for single object.
- R1: Geometry detail level digit, 0 for low, 1 for high.
- R2: Simulation level digit, 0 for low, 1 for medium, 2 for high.
- R3: Interaction mode digit, 0 for immersive, 1 for desktop.

Appendix B: Function modules for VE data acquisition

```

/*
This program has registered a new SCL function - "SaveStan",
which taking an object number from data stack as function parameter,
and then writting the object standard information into a ASCII file.
*/

/*Included header files*/

#include      "APP_DEFS.H"          //Defines various short cuts and values
                                   //for the short cut

#include      <stdio.h>
#include      <stdlib.h>
#include      <string.h>
#include      <ctype.h>

/*Internal function declaration*/

static void __vrtcall SCL_Saver1(void);

/*
Defined function properties
*/

short App_Init(void);
short App_Exit(void);

/*New SCL function declaration*/

static T_COMPILEREC      NewSCL=
    {
        "SaveStan", //Name of the new
command
        0,           //Function code
        0x20,       //2 input, 0 output

        0,           //Compiler code

        E_PROCEDURE, //Function type is
'procedure'
        E_SSNONE,    //None return value
        E_SSOBJNUM, //First input
parameter type is
                                   //virtual
object number
        E_SSINTEGER, //Second input
parameter type is
                                   //an interger
number represents MVEID
    };

static short      SCLCode;          //Deinstallation code

```

```

/*Application initialisation*/

short App_Init(void)
{
    SCLCode=RegisterSCL(&NewSCL,SCL_Saver1); //Register new SCL
function
    if(SCLCode<0) //Checking
registration result
    return(E_ERROR);
    return(E_OK);
}

/*Application exit,unregisters SCL instructions*/

short App_Exit(void)
{
    UnRegisterSCL(SCLCode);

    return(E_OK);
}

/*Internal function definitions*/

static void __vrtcall SCL_Saver1(void) .
{
    short ObjNum; //Declare variable for holding
input value
    short MVEID; //Declare variable for holding
master //environment ID

    /*Declare pointers pointing to the object attribute data
structure*/

    T_WORLDCHUNK *Object, *InitSize, *InitPos, *Rotation,
*Viewpoint; //

    FILE *f; //File pointer for saving object
information

    /*This two values will be used as combined external key values
in the data file*/

    MVEID=PopN(E_SSINTEGER); //Get master environment ID from
the stack
    ObjNum=PopN(E_SSOBJNUM); //Get object number from the
stack

    /*Find addresses of required attributes data section*/

    Object=ChunkAdd(ObjNum,E_CTSTANDARD);
    InitSize=ChunkAdd(ObjNum,E_CTINITSIZE);
    InitPos=ChunkAdd(ObjNum,E_CTINITPOS);
    Rotation=ChunkAdd(ObjNum,E_CTROTATIONS);
    Viewpoint=ChunkAdd(ObjNum,E_CTVIEWPOINT);

    /*retrieving and saving information in a file*/

    if(Object!=NULL)
    {

```

```
f=fopen("StansardInfo.txt", "a");

fprintf(f, "%ld\t", Object->Std.Number);
fprintf(f, "%ld\t", MVEID);

fprintf(f, "%ld\t", Object->Std.XSize);
fprintf(f, "%ld\t", Object->Std.YSize);
fprintf(f, "%ld\t", Object->Std.ZSize);
fprintf(f, "%ld\t", Object->Std.XPos);
fprintf(f, "%ld\t", Object->Std.YPos);
fprintf(f, "%ld\t", Object->Std.ZPos);
fprintf(f, "%ld\t", InitSize->Isz.IXSize);
fprintf(f, "%ld\t", InitSize->Isz.IYSize);
fprintf(f, "%ld\t", InitSize->Isz.IZSize);
fprintf(f, "%ld\t", InitPos->Ips.IXPos);
fprintf(f, "%ld\t", InitPos->Ips.IYPos);
fprintf(f, "%ld\t", InitPos->Ips.IZPos);
fprintf(f, "%ld\t", Rotation->Rot.XCentre);
fprintf(f, "%ld\t", Rotation->Rot.YCentre);
fprintf(f, "%ld\t", Rotation->Rot.ZCentre);
fprintf(f, "%ld\t", Viewpoint->Vpt.NumVPs);
fprintf(f, "%ld\t", (Viewpoint->Vpt.View)->ObjView);
fprintf(f, "%ld\t\n", (Viewpoint->Vpt.View)->ObjCon);

fclose(f);
}
```

```

/*
This program has registered a new SCL function - "SaveDyna",
which taking an object number from data stack as function parameter,
and then writting the object dynamic information into a ASCII file.
*/

/*Included header files*/

#include      "APP_DEFS.H"          //Defines various short cuts and values
                                      //for the short cut

#include      <stdio.h>
#include      <stdlib.h>
#include      <string.h>
#include      <ctype.h>

/*Internal function declaration*/

static void __vrtcall SCL_Saver2(void);

/*
Defined function properties
*/

short App_Init(void);
short App_Exit(void);

/*New SCL function declaration*/

static T_COMPILEREC      NewSCL=
{
command
                                "SaveDyna", //Name of the new
                                0,          //Function code
                                0x20,      //2 input, 0 output
                                0,          //Compiler code
                                E_PROCEDURE, //Function type is
'procedure'
                                E_SSNONE,   //None return value
                                E_SSOBJNUM, //First input
parameter type is
                                //virtual
object number
                                E_SSINTEGER, //Second input
parameter type is
                                //an interger
number represents MVEID
};

static short      SCLCode;          //Deinstallation code

/*Application initialisation*/

short App_Init(void)
{

```

```

        SCLCode=RegisterSCL(&NewSCL,SCL_Saver2);//Register new SCL
function
    if(SCLCode<0)                                //Checking
registration result
        return(E_ERROR);
        return(E_OK);
}

/*Application exit,unregisters SCL instructions*/

short App_Exit(void)
{
    UnRegisterSCL(SCLCode);

    return(E_OK);
}

/*Internal function definitions*/

static void __vrtcall SCL_Saver2(void)
{
    short ObjNum;                                //Declare variable for holding
input value
    short MVEID;                                //Declare variable for holding
master
                                                //environment ID

    /*Declare pointers pointing to the object attribute data
structure*/

    T_WORLDCHUNK *Object, *Dynamics, *AngVel, *Rotation;

    FILE *f;                                    //File pointer for saving object
information

    /*This two values will be used as combined external key values
in the data file*/

    MVEID=PopN(E_SSINTEGER);                    //Get master environment ID from
the stack
    ObjNum=PopN(E_SSOBJNUM);                    //Get object number from the
stack

    /*Find addresses of required attributes data section*/

    Object=ChunkAdd(ObjNum,E_CTSTANDARD);
    Dynamics=ChunkAdd(ObjNum, E_CTDYNAMICS);
    AngVel=ChunkAdd(ObjNum, E_CTANGVELS);
    Rotation=ChunkAdd(ObjNum, E_CTROTATIONS);

    /*retrieving and saving information in a file*/

    if(Object!=NULL)
    {
        f=fopen("DynamicInfo.txt", "a");

        fprintf(f,"%ld\t", Object->Std.Number);
        fprintf(f,"%ld\t", MVEID);

        /*Note: 3D vector data, not array*/

```

```

fprintf(f, "%ld\t", (Dynamics->Dyn.Drive).x);
fprintf(f, "%ld\t", (Dynamics->Dyn.Drive).y);
fprintf(f, "%ld\t", (Dynamics->Dyn.Drive).z);
fprintf(f, "%ld\t", (Dynamics->Dyn.External).x);
fprintf(f, "%ld\t", (Dynamics->Dyn.External).y);
fprintf(f, "%ld\t", (Dynamics->Dyn.External).z);
fprintf(f, "%ld\t", (Dynamics->Dyn.Vel).x);
fprintf(f, "%ld\t", (Dynamics->Dyn.Vel).y);
fprintf(f, "%ld\t", (Dynamics->Dyn.Vel).z);
fprintf(f, "%ld\t", (Dynamics->Dyn.GroundFric).x);
fprintf(f, "%ld\t", (Dynamics->Dyn.GroundFric).y);
fprintf(f, "%ld\t", (Dynamics->Dyn.GroundFric).z);
fprintf(f, "%ld\t", AngVel->Ang.XAngV);
fprintf(f, "%ld\t", AngVel->Ang.YAngV);
fprintf(f, "%ld\t", AngVel->Ang.ZAngV);
fprintf(f, "%ld\t", AngVel->Rot.XRot);
fprintf(f, "%ld\t", AngVel->Rot.YRot);
fprintf(f, "%ld\t", AngVel->Rot.ZRot);
fprintf(f, "%ld\t", (Dynamics->Dyn.IDrive).x);
fprintf(f, "%ld\t", (Dynamics->Dyn.IDrive).y);
fprintf(f, "%ld\t", (Dynamics->Dyn.IDrive).z);
fprintf(f, "%ld\t", (Dynamics->Dyn.IExternal).x);
fprintf(f, "%ld\t", (Dynamics->Dyn.IExternal).y);
fprintf(f, "%ld\t", (Dynamics->Dyn.IExternal).z);
fprintf(f, "%ld\t", (Dynamics->Dyn.IVel).x);
fprintf(f, "%ld\t", (Dynamics->Dyn.IVel).y);
fprintf(f, "%ld\t", (Dynamics->Dyn.IVel).z);
fprintf(f, "%ld\t", (Dynamics->Dyn.IGroundFric).x);
fprintf(f, "%ld\t", (Dynamics->Dyn.IGroundFric).y);
fprintf(f, "%ld\t", (Dynamics->Dyn.IGroundFric).z);
fprintf(f, "%ld\t", AngVel->Ang.IXAngV);
fprintf(f, "%ld\t", AngVel->Ang.IYAngV);
fprintf(f, "%ld\t", AngVel->Ang.IZAngV);
fprintf(f, "%ld\t", AngVel->Rot.IXRot);
fprintf(f, "%ld\t", AngVel->Rot.IYRot);
fprintf(f, "%ld\t\n", AngVel->Rot.IZRot);

```

```

fclose(f);
}
}

```



```

/*
This program has registered a new SCL function - "SaveStat",
which taking an object number from data stack as function parameter,
and then writting the object shape information into a ASCII file.
*/

/*Included header files*/

#include      "APP_DEFS.H"          //Defines various short cuts and values
                                      //for the short cut

#include      <stdio.h>
#include      <stdlib.h>
#include      <string.h>
#include      <ctype.h>

/*Internal function declaration*/

static void __vrtcall SCL_Saver3(void);

/*
Defined function properties
*/

short App_Init(void);
short App_Exit(void);

/*New SCL function declaration*/

static T_COMPILEREC      NewSCL=
{
command
                                "SaveStat", //Name of the new
                                0,          //Function code
                                0x20,      //2 input, 0 output
                                0,          //Compiler code
                                E_PROCEDURE, //Function type is
'procedure'
                                E_SSNONE,   //None return value
                                E_SSOBJNUM, //First input
parameter type is
                                //virtual
object number
                                E_SSINTEGER, //Second input
parameter type is
                                //an interger
number represents MVEID
};

static short      SCLCode;          //Deinstallation code

/*Application initialisation*/

short App_Init(void)
{

```

```

        SCLCode=RegisterSCL(&NewSCL,SCL_Saver3); //Register new SCL
function
    if(SCLCode<0)                                //Checking
registration result
        return(E_ERROR);
        return(E_OK);
}

/*Application exit,unregisters SCL instructions*/

short App_Exit(void)
{
    UnRegisterSCL(SCLCode);

    return(E_OK);
}

/*Internal function definitions*/

static void __vrtcall SCL_Saver3(void)
{
    short ObjNum;                                //Declare variable for holding
input value
    short MVEID;                                //Declare variable for holding
master
                                                //environment ID

    /*Declare pointers pointing to the object attribute data
structure*/

    T_WORLDCHUNK *Object, *Color, *Distance, *Sorting, *Collision,
        *Attachment, *LightSrc, *LitColor, *Texture;

    FILE *f;                                    //File pointer for saving object
information

    /*This two values will be used as combined external key values
in the data file*/

    MVEID=PopN(E_SSINTEGER);                    //Get master environment ID from
the stack
    ObjNum=PopN(E_SSOBJNUM);                    //Get object number from the
stack

    /*Find addresses of required attributes data section*/

    Object=ChunkAdd(ObjNum,E_CTSTANDARD);
    Color=ChunkAdd(ObjNum,E_CTCOLOURS);
    Distance=ChunkAdd(ObjNum,E_CTDISTANCE);
    Sorting=ChunkAdd(ObjNum,E_CTSORTING);
    Collision=ChunkAdd(ObjNum,E_CTCOLLISION);
    Attachment=ChunkAdd(ObjNum,E_CTATTACHMENTS);
    LightSrc=ChunkAdd(ObjNum,E_CTLIGHTSOURCE);
    LitColor=ChunkAdd(ObjNum,E_CTLITCOLS);
    Texture=ChunkAdd(ObjNum,E_CTTEXTURES);

    /*retrieving and saving information in a file*/

    if(Object!=NULL)

```

```

{
  f=fopen("StaticInfo.txt", "a");

  fprintf(f, "%ld\t", Object->Std.Number);
  fprintf(f, "%ld\t", MVEID);

  fprintf(f, "%c\t", Color->Col.Colour);
  fprintf(f, "%ld\t", Distance->Dis.VisDist);
  fprintf(f, "%ld\t", Distance->Dis.InvDist);
  fprintf(f, "%ld\t", Distance->Dis.Replace);
  fprintf(f, "%ld\t", Sorting->Sor.XPos);
  fprintf(f, "%ld\t", Sorting->Sor.YPos);
  fprintf(f, "%ld\t", Sorting->Sor.ZPos);
  fprintf(f, "%ld\t", Sorting->Sor.XSize);
  fprintf(f, "%ld\t", Sorting->Sor.YSize);
  fprintf(f, "%ld\t", Sorting->Sor.ZSize);
  fprintf(f, "%ld\t", (Collision->Cln.Collision)->XSize);
  fprintf(f, "%ld\t", (Collision->Cln.Collision)->YSize);
  fprintf(f, "%ld\t", (Collision->Cln.Collision)->ZSize);
  fprintf(f, "%ld\t", (Collision->Cln.Collision)->XOff);
  fprintf(f, "%ld\t", (Collision->Cln.Collision)->YOff);
  fprintf(f, "%ld\t", (Collision->Cln.Collision)->ZOff);
  fprintf(f, "%ld\t", (Collision->Cln.Collision)->IXSize);
  fprintf(f, "%ld\t", (Collision->Cln.Collision)->IYSize);
  fprintf(f, "%ld\t", (Collision->Cln.Collision)->IZSize);
  fprintf(f, "%ld\t", (Collision->Cln.Collision)->IXOff);
  fprintf(f, "%ld\t", (Collision->Cln.Collision)->IYOff);
  fprintf(f, "%ld\t", (Collision->Cln.Collision)->IZOff);
  fprintf(f, "%ld\t", (Attachment->Att.Att)->Facet);
  fprintf(f, "%ld\t", (Attachment->Att.Att)->Object);
  fprintf(f, "%ld\t", LightSrc->Lig.XVOff);
  fprintf(f, "%ld\t", LightSrc->Lig.YVOff);
  fprintf(f, "%ld\t", LightSrc->Lig.ZVOff);
  fprintf(f, "%ld\t", LightSrc->Lig.IXVOff);
  fprintf(f, "%ld\t", LightSrc->Lig.IYVOff);
  fprintf(f, "%ld\t", LightSrc->Lig.IZVOff);
  fprintf(f, "%ld\t", LightSrc->Lig.Bright);
  fprintf(f, "%ld\t", LightSrc->Lig.IBright);
  fprintf(f, "%ld\t", LightSrc->Lig.BeamWidth);
  fprintf(f, "%ld\t", LightSrc->Lig.IBeamWidth);
  fprintf(f, "%ld\t", LightSrc->Lig.Dispersion);
  fprintf(f, "%ld\t", LightSrc->Lig.IDispersion);
  fprintf(f, "%ld\t", LightSrc->Lig.BeamEdge);
  fprintf(f, "%ld\t", LightSrc->Lig.IBeamEdge);
  fprintf(f, "%ld\t", LightSrc->Lig.ColR);
  fprintf(f, "%ld\t", LightSrc->Lig.ColG);
  fprintf(f, "%ld\t", LightSrc->Lig.ColB);
  fprintf(f, "%ld\t", LightSrc->Lig.IColR);
  fprintf(f, "%ld\t", LightSrc->Lig.IColG);
  fprintf(f, "%ld\t", LightSrc->Lig.IColB);
  fprintf(f, "%c\t", LitColor->Lit.LitCol);
  fprintf(f, "%ld\t", Texture->Txr.NTextures);
  fprintf(f, "%ld\t", (Texture->Txr.Tex)->Facet);
  fprintf(f, "%ld\t\n", (Texture->Txr.Tex)->Texture);

  fclose(f);
}
}

```

```

/*
This program has registered a new SCL function - "SaveShp",
which taking an object number from data stack as function parameter,
and then writing the object shape information into a ASCII file.
*/

/*Included header files*/

#include      "APP_DEFS.H"          //Defines various short cuts and values
                                       //for the short cut

#include      <stdio.h>
#include      <stdlib.h>
#include      <string.h>
#include      <ctype.h>

/*Internal function declaration*/

static void __vrtcall SCL_Saver4(void);

/*
Defined function properties
*/

short App_Init(void);
short App_Exit(void);

/*New SCL function declaration*/

static T_COMPILEREC      NewSCL=
    {
    "SaveShp", //Name of the new
command          0, //Function code
                0x20, //2 input, 0 output

                0, //Compiler code

                E_PROCEDURE, //Function type is
'procedure'
                E_SSNONE, //None return value
                E_SSOBJNUM, //First input
parameter type is //virtual
object number
parameter type is
number represents MVEID
                E_SSINTEGER, //Second input
                //an interger
    };

static short          SCLCode; //Deinstallation code

/*Application initialisation*/

short App_Init(void)
{

```

```

        SCLCode=RegisterSCL(&NewSCL,SCL_Saver4);//Register new SCL
function
    if(SCLCode<0)                                //Checking
registration result
        return(E_ERROR);
        return(E_OK);
}

/*Application exit,unregisters SCL instructions*/

short App_Exit(void)
{
    UnRegisterSCL(SCLCode);

    return(E_OK);
}

/*Internal function definitions*/

static void __vrtcall SCL_Saver4(void)
{
    short ObjNum;                                //Declare variable for holding
input value
    short MVEID;                                //Declare variable for holding
master
                                                //environment ID

    /*Declare pointers pointing to the object attribute data
structure*/

    T_WORLDCHUNK *Object;

    /*Declare pointers pointing to the shape attribute data
structure*/
    T_SHAPECHUNK *ShpSize, *Facet, *Line, *Point;

    FILE *f;                                    //File pointer for saving object
information

    /*This two values will be used as combined external key values
in the data file*/

    MVEID=PopN(E_SSINTEGER);                    //Get master environment ID from
the stack
    ObjNum=PopN(E_SSOBJNUM);                   //Get object number from the
stack

    /*Find address of required object data*/

    Object=ChunkAdd(ObjNum,E_CTSTANDARD);

    /*Find addresses of required shape attributes data section*/

    ShpSize>(*C_ShapeAdd)[Object->Std.Type];
    Facet>(*C_ShapeAdd)[Object->Std.Type];
    Line>(*C_ShapeAdd)[Object->Std.Type];
    Point>(*C_ShapeAdd)[Object->Std.Type];

    /*retrieving and saving information in a file*/

```

```
if(Object!=NULL)
{
  f=fopen("ShapeInfo.txt", "a");

  fprintf(f,"%ld\t", Object->Std.Number);
  fprintf(f,"%ld\t", MVEID);
  fprintf(f,"%ld\t", Object->Std.Type);

  fprintf(f,"%ld\t", ShpSize->Siz.XSize);
  fprintf(f,"%ld\t", ShpSize->Siz.YSize);
  fprintf(f,"%ld\t", ShpSize->Siz.ZSize);
  fprintf(f,"%ld\t", Facet->Fac.NumFacets);
  fprintf(f,"%ld\t", Line->Lin.NumLines);
  fprintf(f,"%ld\t", Point->Pnt.NumPoints);

  fclose(f);
}
}
```

```

/*
This program has registered a new SCL function - "ObjList".
It scans all objects inside a virtual environment in terms of
their name, base shape number, position in the scene graph tree,
parent, children, and sibling relationships. And then writing
the objects information into a ASCII file.
*/

/*Included header files*/

#include      "APP_DEFS.H"          //Defines various short cuts and values
                                      //for the short cut

#include      <stdio.h>
#include      <stdlib.h>
#include      <string.h>
#include      <ctype.h>

/*Internal function declaration*/

static void __vrtcall SCL_Saver5(void);

/*
Defined function properties
*/

short App_Init(void);
short App_Exit(void);

/*New SCL function declaration*/

static T_COMPILEREC      NewSCL=
{
command
                                "ObjList", //Name of the new
                                0,          //Function code
                                0x20,      //2 input, 0 output
                                0,          //Compiler code
                                E_PROCEDURE,//Function type is
'procedure'
                                E_SSNONE,   //None return value
                                E_SSOBJNUM, //First input
parameter type is
                                //virtual
object number
                                E_SSINTEGER,//Second input
parameter type is
                                //an interger
number represents MVEID
};

static short      SCLCode;          //Deinstallation code

/*Application initialisation*/

```

```

short App_Init(void)
{
    SCLCode=RegisterSCL(&NewSCL,SCL_Saver5);//Register new SCL
function
    if(SCLCode<0)                                //Checking
registration result
    return(E_ERROR);
    return(E_OK);
}

/*Application exit,unregisters SCL instructions*/

short App_Exit(void)
{
    UnRegisterSCL(SCLCode);

    return(E_OK);
}

/*Internal function definitions*/

static void __vrtcall SCL_Saver5(void)
{
    short ObjNum;                                //Declare variable for holding
input value
    short MVEID;                                //Declare variable for holding
master
                                                //environment ID

    /*Declare pointers pointing to the object attribute data
structure*/

    T_WORLDCHUNK *Object, **Parent;

    FILE *f;                                    //File pointer for saving object
information

    /*This two values will be used as combined external key values
in the data file*/

    MVEID=PopN(E_SSINTEGER);                    //Get master environment ID from
the stack
    ObjNum=PopN(E_SSOBJNUM);                    //Get object number from the
stack

    /*Find address of required object data*/

    Object=ChunkAdd(ObjNum,E_CTSTANDARD);
    Parent=Object->Std.Parent;

    /*retrieving and saving information in a file*/

    if(Object!=NULL)
    {
        f=fopen("ObjList.txt", "a");

        fprintf(f,"%ld\t", Object->Std.Number);
        fprintf(f,"%ld\t", MVEID);
        fprintf(f,"%ld\t", Object->Std.TotLen);
        fprintf(f,"%ld\t", Object->Std.Type);
    }
}

```



```
fprintf(f,"%ld\t", Object->Std.Layer);

/*Get the parent object number through its absolute address*/
fprintf(f,"%ld\t", (*Parent)->Std.Number);

/*if has child object, and sibling*/
fprintf(f,"%ld\t", Object->Std.Child);
fprintf(f,"%ld\t", Object->Std.Sibling);

fclose(f);
}
}
```

Appendix C: Run-time platform and SCL communication code

```
/* The following codes are abstracted from the container programme
for communicating virtual environments and database.*/
```

```
// CXuVE2Set implementation

IMPLEMENT_DYNAMIC(CXuVE2Set, CRecordset)

CXuVE2Set::CXuVE2Set(CDatabase* pdb)
    : CRecordset(pdb)
{
    //{{AFX_FIELD_INIT(CXuVE2Set)
    //Record set from the GeneralRef Table
    m_MVEID = _T("");
    m_MVEName = _T("");
    m_Envcode = _T("");
    m_PrimaryRes = _T("");
    m_SecondaryRes = _T("");
    //Record set from the ObjectLists Table
    m_VirObjID = _T("");;
    m_TotLen = _T("");;
    m_ShpNum = _T("");;
    m_Layer = _T("");;
    m_Parent = _T("");;
    m_Child = _T("");;
    m_Sibling = _T("");;
    //Record set from the Property Tables
    .....
    ///Record set from the Manufacturing data Table
    m_Part_ID = 0;
    m_Part_Name = _T("");;
    m_Part_category = 0;
    m_Machining_process = _T("");;
    m_Shape = 0;
    m_Dimension = 0;
    m_Raw_material_form = _T("");;
    m_Feature_list = _T("");;
    m_Build_method = _T("");;
    m_nFields = 9;
    //}}AFX_FIELD_INIT
    m_nDefaultType = dynaset;
}

CString CXuVE2Set::GetDefaultConnect()
{
    return _T("ODBC;DSN=VME1");
}

CString CXuVE2Set::GetDefaultSQL()
{
    return _T("[GeneralRef]");
}

void CXuVE2Set::DoFieldExchange(CFieldExchange* pFX)
{
    //{{AFX_FIELD_MAP(CXuVE2Set)
    pFX->SetFieldType(CFieldExchange::outputColumn);
```

```

//Environment tables record set exchange
.....
//part tables record set exchange
RFX_Int(pFX, _T("[Part ID]"), m_Part_ID);
RFX_Text(pFX, _T("[Part Name]"), m_Part_Name);
RFX_Int(pFX, _T("[Part category]"), m_Part_category);
RFX_Text(pFX, _T("[Machining process]"), m_Machining_process);
RFX_Int(pFX, _T("[Shape]"), m_Shape);
RFX_Int(pFX, _T("[Dimension]"), m_Dimension);
RFX_Text(pFX, _T("[Raw material form]"), m_Raw_material_form);
RFX_Text(pFX, _T("[Feature list]"), m_Feature_list);
RFX_Text(pFX, _T("[Build method]"), m_Build_method);
//}}AFX_FIELD_MAP
}

/*The main class of handling environment and container programme
communications, XuVE2View.cpp : implementation of the CXuVE2View
class*/

#include "stdafx.h"
#include "XuVE2.h"

#include "XuVE2Set.h"
#include "XuVE2Doc.h"
#include "XuVE2View.h"
#include "Machine.h"
#include "MVE.h"

/*Enable the direct interaction between the two parts. By clicking
the buttons on the container, different parts will be displayed*/
IMPLEMENT_DYNCREATE(CXuVE2View, CRecordView)

BEGIN_MESSAGE_MAP(CXuVE2View, CRecordView)
    //{{AFX_MSG_MAP(CXuVE2View)
    ON_BN_CLICKED(IDC_LOAD_WORLD, OnLoadWorld)
    ON_BN_CLICKED(IDC_ABOUT_BTN, OnAboutBtn)
    ON_BN_CLICKED(IDC_TOGGLE_TOOL, OnToggleTool)
    ON_BN_CLICKED(IDC_PART1, OnPart1)
    ON_BN_CLICKED(IDC_PART3, OnPart3)
    ON_BN_CLICKED(IDC_PART4, OnPart4)
    ON_BN_CLICKED(IDC_PART5, OnPart5)
    ON_BN_CLICKED(IDC_PART6, OnPart6)
    ON_BN_CLICKED(IDC_PART7, OnPart7)
    ON_BN_CLICKED(IDC_PART8, OnPart8)
    ON_BN_CLICKED(IDC_PART9, OnPart9)
    ON_BN_CLICKED(IDC_PART2, OnPart2)
    ON_BN_CLICKED(IDC_TOOL_INFO, OnToolInfo)
    ON_BN_CLICKED(IDC_MACHINE, OnMachine)
    ON_BN_CLICKED(IDC_LOAD_DATA, OnLoadData)
    ON_NOTIFY(TCN_SELCHANGE, IDC_TAB1, OnSelchangeTab1)
    //}}AFX_MSG_MAP
    // Standard printing commands
    ON_COMMAND(ID_FILE_PRINT, CRecordView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_DIRECT, CRecordView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_PREVIEW,
CRecordView::OnFilePrintPreview)
END_MESSAGE_MAP()

//Environment configuration

```

```

void CXuVE2View::DoDataExchange(CDataExchange* pDX)
{
    CRecordView::DoDataExchange(pDX);
    //{AFX_DATA_MAP(CXuVE2View)
    DDX_Control(pDX, IDC_TAB1, m_resources);
    DDX_Control(pDX, IDC_SUPERSCAPE1, m_3dcontrol);
    DDX_Text(pDX, IDC_EDIT1, m_xpos);
    DDX_Text(pDX, IDC_EDIT2, m_ypos);
    DDX_FieldText(pDX, IDC_FIELD1, m_pSet->m_MaxSpindleSpeed,
m_pSet);
    DDX_FieldText(pDX, IDC_FIELD2, m_pSet->m_MinSpindleSpeed,
m_pSet);
    .....
    //})AFX_DATA_MAP
}

//Loading environment from local disk

void CXuVE2View::OnLoadWorld()
{
    // TODO: Add your control notification handler code here
    CString m_Filename;
    FILE *f;

    char *FileName=NULL;

    //Can only load XVR worlds so create a filter for this filetype
only.
    char BASED_CODE szFilter[] = "Superscape XVR (*.xvr)|*.xvr|";

    //Create and handle an open dialog box.
    CFileDialog *dialog;
    CString exten;

    dialog = new CFileDialog(TRUE,"XVR",NULL, NULL,szFilter);
    if (dialog->DoModal() == IDOK)
    {
        //Now handle file open procedures.
        exten = dialog->GetFileExt();
        exten.MakeUpper();
        if (exten == "XVR")
        {
            //Just inform the user that we are loading a world.
            MessageBox("Loading selected
world.", "Message", MB_ICONINFORMATION+MB_OK);

            m_Filename = dialog->GetPathName();
            f = fopen(dialog->GetPathName(), "rb");
            if (f == NULL)
            {
                MessageBox("Can't open selected
file.", "ERROR", MB_OK + MB_ICONSTOP);
                return;
            }
            else
            {
                //Set the source for the world to the file
selected in the open dialog.
                m_3dcontrol.SetSrc(m_Filename);
            }
            fclose(f);
        }
    }
}

```

```

        }
    }
    delete dialog;
}

//Controlling the appearances of the objects
void CXuVE2View::OnToggleTool()
{
    // TODO: Add your control notification handler code here
    UINT m_check;

    //Handle the tool checkbox. If checked, tool is OFF otherwise
    tool is ON.
    if ( (m_check=IsDlgButtonChecked( IDC_TOGGLE_TOOL ) ) == 0)
        m_3dcontrol.SetBoolProperty("Cutter","Propb",FALSE);
    else
        m_3dcontrol.SetBoolProperty("Cutter","Propb", TRUE);
}

//Changing default part
void CXuVE2View::OnPart1()
{
    // TODO: Add your control notification handler code here
    m_3dcontrol.SetMarker(1,1);
}
void CXuVE2View::OnPart2()
{
    // TODO: Add your control notification handler code here
    m_3dcontrol.SetMarker(1,2);
}
void CXuVE2View::OnPart3()
{
    // TODO: Add your control notification handler code here
    m_3dcontrol.SetMarker(1,3);
}
.....

//Getting current tool information
void CXuVE2View::OnToolInfo()
{
    // TODO: Add your control notification handler code here
    //Get the position data.
    long m_xpos = m_3dcontrol.GetLongProperty("slot1", "X
Position");
    long m_ypos = m_3dcontrol.GetLongProperty("slot1", "Y
Position");
    //Refresh the data display on the dialog.
    UpdateData(FALSE);
}

//Handling event fired from the environment
BEGIN_EVENTSINK_MAP(CXuVE2View, CRecordView)
    //{AFX_EVENTSINK_MAP(CXuVE2View)
    ON_EVENT(CXuVE2View, IDC_SUPERSCAPE1, 6 /* SCLEvent */,
OnSCLEventSuperscapel, VTS_I4 VTS_I4)
    //}}AFX_EVENTSINK_MAP
END_EVENTSINK_MAP()

void CXuVE2View::OnSCLEventSuperscapel(long arg1, long arg2)

```

```
{
    // TODO: Add your control notification handler code here
    m_ObjID = arg1;

    //arg2 is the total movie length in ms.
    m_Timer = arg2;

    //Refresh the data display on the dialog.
    UpdateData(FALSE);
}

//Set simulation parameters and markers

void CMySettingDlg::OnDiverLoop()
{
    //Pass over the parameter to the lathe.
    m_3dcontrol.SetBoolProperty("Lathe", "mainswitch", m_marker);

    //Set the marker to stop user turn on machine
    m_3dcontrol.SetMarker(2,1);
    .....
}
```

```

/*Simulation program (partial) on the lathe*/

//Configure machine specification
short Marker, eventcounter, ObjID, timer, P=0, speed;
long MachineConfig[], setting[];
MachineConfig[0] = property("Lathe-Holder", "property1");
MachineConfig[1] = property("Lathe-Holder", "property2");
MachineConfig[2] = property("Lathe-Holder", "property3");
MachineConfig[3] = property("Lathe-Holder", "property4");
MachineConfig[4] = property("Lathe-Holder", "property5");
MachineConfig[5] = property("Lathe-Holder", "property6");

//Taking user input settings from various controllers*/
if(activate("#12", 0))
{
    speed = rot(me)/30;
    switch (speed)
    {
        case 0;
            Setting[0] = 1;
        case 0;
            Setting[0] = 2;
        .....
    }
    else if (activate("#13",0))
    {
        .....
    }
    .....
//Check event occurred
if(eventcounter == 1)

//Passing object ID to container
Fireevnt(ObjID, timer);

//Object data retrieved by container and processed
wait(1000);

//Decide the route for the simulation and interaction
Marker = property ("Machine", "Marker");

```

```

//Running simulation routes
if (Marker == 1)
{
  zrot (me)=0;
  resume (0, 2);
  if (actchild (me, 0))
  {
    while (mouseb)
      waitf;
    sound (1, 64, -100000, 0);
    repeat (3)
    {
      zrot (me)+=12;
      waitf;
    }
    ++P;
    if (P==10)
      P=0;
    actchild (me, 0);
  }
  if (actchild (me, 13))
  {
    while (mouseb)
      waitf;
    sound (1, 60, -100000, 0);
    repeat (3)
    {
      zrot (me)--=12;
      waitf;
    }
    --P;
    if (P==--1)
      P=9;
    actchild (me, 13);
  }
  if (Marker == 2)
  {
    xrot ('Vice - Handle[4]')=90;
    resume (0, 2);
    if (activate (me, 0))
    {
      if (xrot ('Vice - Handle[4]')>80 && xrot
        ('Vice - Handle[4]')<100)
        h=1;
      if (xrot ('Vice - Handle[4]')>40 && xrot
        ('Vice - Handle[4]')<60)
        h=2;
      switch (h);
      case 1:
      {
        repeat (3)
        {
          xrot ('Vice - Handle[4]')-=15;
          waitf;
        }
        xangv ('chunk')=zrot ('SAFE:dial');
        waitf;
      }
      case 2:
      {
        repeat (3)

```



```

    {
        xrot ('Vice - Handle[4]')+15;
        waitf;
    }
    xangv ('chunk')=0;
    waitf;
}
}
else if(Marker == 3)
{
    short  mx, my;
    fixed  rx, ry;

resume (1, 0);
if (activate (me, 0) && zrot ('Vice - Handle[210]')==90)
{
    mx=mousex;
    my=mousey;
    ry=yrot (parent (me));
    while (mouseb)
    {
        xrot (me)=ry+mousex-mx;
        waitf;
        xpos ('TailGroup')+ry+mousex-mx;
        waitf;
    }
    clrtrig (me, 0);
}
else if(Marker == 5)
{
    short  mx, my;
    fixed  rx, ry;

resume (1, 0);
if (activate (me, 0))
{
    mx=mousex;
    my=mousey;
    ry=yrot (parent (me));
    while (mouseb && xpos ('NewCone')>0 && xpos ('NewCone')<1100
        && yrot ('handle2')==135)
    {
        xrot (me)=ry+mousex-mx;
        waitf;
        xpos ('NewCone')+ry+mousex-mx;
        waitf;
        if (xpos ('NewCone')<=0 || xpos ('NewCone')>=1100)
            xpos ('NewCone')=ixpos ('NewCone');
    }
    clrtrig (me, 0);
}
//quill or tailstock locker
resume (0, 1);
if (activate (me, 0) && zrot ('Vice - Handle[210]')<80)
{
    zrot ('Vice - Handle[210]')+45;
    waitf;
    sptext (me)="Tailstock moveable";
    waitf;
}
if (activate (me, 13) && zrot ('Vice - Handle[210]')>55)

```

```

{
  zrot ('Vice - Handle[210]')==45;
  waitf;
  sptext (me)="Tailstock locked";
  waitf;
}
}
else
{
//Apron holder
short  mx, my;
fixed  rx, ry;

resume (1, 0);
if (activate (me, 0))
{
  mx=mousex;
  my=mousey;
  ry=yrot (parent (me));
  while (mouseb)
  {
    zrot (me)=ry+mousex-mx;
    waitf;
    xpos ('ApronHolder')+ry+mousex-mx;
    waitf;
  }
  clrtrig (me, 0);
}
if (first)
{
  marker (1)==0;
  invis ('P-Contour-turn');
  invis ('P-Boring');
  invis ('P-Drilling');
  invis ('P-Parting');
  invis ('P-Shoulder-face');
  invis ('P-Sra-turn');
  invis ('P-Taper-Boring');
  invis ('P-Taper-turn');
  invis ('P-Threading');
}
else
{
  if (marker (1)==1)
  {
    vis ('P-Contour-turn');
    invis ('P-Boring');
    invis ('P-Drilling');
    invis ('P-Parting');
    invis ('P-Shoulder-face');
    invis ('P-Sra-turn');
    invis ('P-Taper-Boring');
    invis ('P-Taper-turn');
    invis ('P-Threading');
    marker (1)==0;
  }
  else
  {
    if (marker (1)==2)
    {
      invis ('P-Contour-turn');
    }
  }
}

```

```

vis ('P-Boring');
invis ('P-Drilling');
invis ('P-Parting');
invis ('P-Shoulder-face');
invis ('P-Sra-turn');
invis ('P-Taper-Boring');
invis ('P-Taper-turn');
invis ('P-Threading');
marker (1)==0;
}
else
{
  if (marker (1)==3)
  {
    invis ('P-Contour-turn');
    invis ('P-Boring');
    vis ('P-Drilling');
    invis ('P-Parting');
    invis ('P-Shoulder-face');
    invis ('P-Sra-turn');
    invis ('P-Taper-Boring');
    invis ('P-Taper-turn');
    invis ('P-Threading');
    marker (1)==0;
  }
  else
  {
    if (marker (1)==4)
    {
      invis ('P-Contour-turn');
      invis ('P-Boring');
      invis ('P-Drilling');
      vis ('P-Parting');
      invis ('P-Shoulder-face');
      invis ('P-Sra-turn');
      invis ('P-Taper-Boring');
      invis ('P-Taper-turn');
      invis ('P-Threading');
      marker (1)==0;
    }
    else
    {
      if (marker (1)==5)
      {
        invis ('P-Contour-turn');
        invis ('P-Boring');
        invis ('P-Drilling');
        invis ('P-Parting');
        vis ('P-Shoulder-face');
        invis ('P-Sra-turn');
        invis ('P-Taper-Boring');
        invis ('P-Taper-turn');
        invis ('P-Threading');
        marker (1)==0;
      }
      else
      {
        if (marker (1)==6)
        {
          invis ('P-Contour-turn');
          invis ('P-Boring');

```


Appendix D: Virtual and real manufacturing cell communication

```

/*--robot main menu--*/
#include "stdio.h"
#include "graphics.h"
#include "stdlib.h"
#include "head-c.h"
#include "conio.h"
#include "dos.h"
#include "ctype.h"
#include "math.h"

void mainl();
void mainn();
void menuu();
void mouse(),mousea();
void mouse_function();
void show_mouse();
void get_mouse();
void hide_mouse();
void init_mouse();
void mainl1();
void main22();
void edit();
void jobteach();
void progteach();
void playback();
void split_space();
void test_file0(char filename[20]);
void test_file1(char filename[20]);
void test_file(char filename[20]);
void puma_com(),bpot_com(),audit_com();
void receiving_com(),receivingl_com();
void transmit_com(),transmit0_com(),transmit1_com(),transmit01_com();
void test_save(char data[100]),command_test(),menu();
void nett(),rec_com(),trans_com(),trans0_com();
void send_file_name(),wait(),get_file_name();
void robot_init();
namecomp();
character();
monitor_com();
chmov();

FILE * fileptr=NULL;

unsigned int x1,y1,n;
unsigned short int buttons;
char str[25], str1[20], strname[50], address[50], midvar[50];
char *name, *name1,namexu[20];
char chv;
unsigned int i, x, pos,pos1, pos2, name_pos;
unsigned int filesize();

main()
{
    char ch;
    int x,y;
    int driver, mode;
    register int i;
    driver=DETECT;
    mode=0;

```

```

initgraph(&driver, &mode, "c:\\tc\\bgi");

settextstyle(DEFAULT_FONT, HORIZ_DIR, 3);
setbkcolor(MAGENTA);
outtextxy(140, 60, "INDUSTRIAL ROBOT");
outtextxy(100, 100, "LOCAL NETWORK SYSTEM");
settextstyle(TRIPLEX_FONT, HORIZ_DIR, 1);

outtextxy(200, 400, "Please Input Your Choice");
outtextxy(200, 440, "by Mouse Cursor");
settextstyle(TRIPLEX_FONT, HORIZ_DIR, 1);

setfillstyle(SOLID_FILL, LIGHTBLUE);
bar3d(79, 223, 159, 375, 20, 1);
outtextxy(94, 231, "< 1 >");
outtextxy(91, 259, "PUMA");
outtextxy(91, 299, "ROBOT");
outtextxy(91, 339, "WORLD");

setfillstyle(SOLID_FILL, GREEN);
bar3d(279, 223, 359, 375, 20, 1);
outtextxy(294, 231, "< 2 >");
outtextxy(283, 259, "LANSING");
outtextxy(283, 299, "ROBOT");
outtextxy(283, 339, "WORLD");

setfillstyle(SOLID_FILL, RED);
bar3d(479, 223, 559, 375, 20, 1);
outtextxy(494, 231, "< 3 >");
outtextxy(495, 259, "END");
outtextxy(495, 299, "OF");
outtextxy(495, 339, "WORK");

gotoxy(1, 1);
show_mouse();
/* init_mouse(); */
do
{
    get_mouse();
    gotoxy(1, 1);
    printf("%d, %d", x1, y1);

    if(x1>=10&&x1<=20&&y1>=28&&y1<=47&&buttons==1)

    {
        hide_mouse();
        cleardevice();
        mainl();
    }
/*    if((ch=getch())=='1')
    {
        cleardevice();
        mainl();
    } */
    if(x1>=35&&x1<=45&&y1>=28&&y1<=47&&buttons==1)
    {
        hide_mouse();
        cleardevice();
        mainn();
    }
    if(x1>=60&&x1<=70&&y1>=28&&y1<=47&&buttons==1)

```

```

    {
        hide_mouse();
        closegraph();
        exit(0);
    }

    }while(buttons!=3);
cleardevice();
exit(0);

return 0;
}

void main1()
{
    int x,y,c;

    settextstyle(TRIPLEX_FONT,HORIZ_DIR,4);
    setbkcolor(LIGHTBLUE);
    outtextxy(200,120,"WELCOME TO");
    outtextxy(145,220,"DNC COMMUNICATION");
    settextstyle(TRIPLEX_FONT,HORIZ_DIR,1);
    outtextxy(250,400,"CLICK TO START");
    do
    {
        get_mouse();
        if(buttons==1)
        {
            cleardevice();
            menu();
        }
    }while(buttons!=3);
}

void menu()
{
    int x,y,c;
/* int driver,mode;
register int i;
driver=DETECT;
mode=0;
initgraph(&driver,&mode,"c:\\tc\\bgi"); */

    setbkcolor(LIGHTBLUE);
    settextstyle(TRIPLEX_FONT,HORIZ_DIR,1);
    outtextxy(120,50,"DNC for Flexible Manufacturing System");
    setfillstyle(SOLID_FILL,RED);
    bar3d(128,144,536,184,0,0);
    outtextxy(140,150,"1. Communication for PUMA Robot");
    setfillstyle(SOLID_FILL,RED);
    bar3d(128,192,536,232,0,0);
    outtextxy(140,200,"2. Communication for AUDIT Lathe");
    setfillstyle(SOLID_FILL,RED);
    bar3d(128,240,536,280,0,0);
    outtextxy(140,250,"3. Communication for Bridgeport");
    setfillstyle(SOLID_FILL,RED);
    bar3d(128,296,536,336,0,0);
    outtextxy(140,300,"4. Communication for NETWORK SERVER");
    setfillstyle(SOLID_FILL,RED);
    bar3d(128,344,536,384,0,0);
    outtextxy(140,350,"5. PCB Assembly");
}

```

```

        setfillstyle(SOLID_FILL, RED);
        bar3d(128, 392, 536, 432, 0, 0);
        outtextxy(140, 400, "6.  EXIT");
        outtextxy(210, 450, "[ Enter Your Choice ]");
gotoxy(1, 1);
show_mouse();
do
{
    get_mouse();
    if(x1>=16&&x1<=67)
    {
        if(y1>=18&&y1<=23&&(buttons==1||buttons==2))
        {
            hide_mouse();
            puma_com();
            closegraph();
        }
        if(y1>=24&&y1<=29&&(buttons==1||buttons==2))
        {
            hide_mouse();
            closegraph();
            audit_com();
        }
        if(y1>=30&&y1<=35&&(buttons==1||buttons==2))
        {
            hide_mouse();
            closegraph();
            bpot_com();
        }
        if(y1>=37&&y1<=42&&(buttons==1||buttons==2))
        {
            hide_mouse();
            closegraph();
            nett_com();
        }
        if(y1>=43&&y1<=48&&(buttons==1||buttons==2))
        {
            hide_mouse();
            assembly();
            closegraph();
        }
        if(y1>=49&&y1<=54&&(buttons==1||buttons==2))
        {
            hide_mouse();
            cleardevice();
            main();
        }
    }
}while(buttons!=3);
/* do
{
    c=getch();
    switch(c)
    {
        case '1':
            puma_com();
            cleardevice();
            break;

        case '2':
            audit_com();
    }
}

```



```

        break;

        case'3':
        bpot_com();
        break;

        case'4':
        nett_com();
        break;

        case'5':
        assembly();
        cleardevice();
        break;

        case'6':
        //      restorecrtmode();
        //      clrscr();
        //      exit(0);
        cleardevice();
        main();
        break;
        default:
        //      exit(0);
        break;
    }
}while(c!='1' && c!='2' && c!='3'&& c!='4'&&c!='5'&&c!='6'); */
}

void puma_com()
{
    int          i,j, port, port1,stat,stat1;
    int          card_no, total_port,data_item;
    char         data[20],filename[20];
    char         w_data[20],command[20];
    char         ch,str[20];

    restorecrtmode();
    clrscr();
    printf("Reset the moxa cards....\n");
    if ((card_no = sio_reset()) == 0)
    {
        printf("No Card Found !\n");
        exit(0);
    }
    printf("Total Card : %d\n", card_no);

    printf("Read the ID....\n");
    for (i = 1; i <= card_no; i++)
    {
        printf("(%d)\tSerial no: %d", i, sio_id(i));
        printf("\tMapping   : %X\n", sio_bank(i));
    }

    printf("Setting port 7 to 9600, N, 8, 1....\n");
    port = 7;
    if ( sio_ioctl(port, B9600, BIT_8 | P_NONE | STOP_1) !=
0)
    {
        printf("Port # %d IOCTL error !\n", port);
    }
}

```

```

        exit(0);
    }
    for(port=6;port<=13;port++)
    {
        stat=sio_open(port);
        if(stat != 0)
            {printf("Port #%d can not be opened!\n",port);
            exit(0);}
    }
    /*
    port=8;
    stat1=sio_enableTX(port);
    if(stat1!=0)
        { printf("PORT 8 TRANSMIT ERROR\n");
        exit(0);
        }
    */
    printf("Read information from PUMA Controller....\n");
    do
    {
        while(kbhit()==0)
        {
            port=7;
            stat1=sio_read(port,data,100);
            /*
            sio_timeout(50);
            stat1=sio_lininput_t(port,data,100,13);
            */
            if(stat1>=0)
            {
                data[stat1]=0;
                printf("%s",data);
            }
            /*
            else exit(0);
            */

            /*
            if((i=sio_read(port,data,100))==0)
                {
                    printf("No data received\n");
                    exit(0);
                }

            else if ((i = sio_read(port, data, 100)) < 0) {
                printf("Port #%d READ error,i= %d !\n",
port,i);
                exit(0);
            }
            else if(i) {
                data[i] = 0;
                printf("%s",data);
            }
            */
        }
    }

    do
    {
        ch=getch();
        switch(ch)
        {
            case 'N':
            case 'n':
                w_data[0] = ch;
                w_data[1] = '\r';
                sio_write(7,w_data,2);
                /*
                if((i = sio_write(7,w_data,2)) < 0) {
                    if((i = sio_write(6,w_data,2)) < 0) {

```

```

                printf("\naaa");
                exit(0);
            }
        } /*
        break;
        default:break;
    }
    }while(ch!='n' && ch!='N' && ch!='.' && ch!='y' &&
ch!='Y');
    }while(ch!='.' && ch!='y');
    //  command_test();

    for(;;)
    {
        printf("\n\nInput the program file name:");
        gets(filename);
        if (strcmp(filename,"esc")==0 || strcmp(filename,"ESC")==0) {
            command_test();
            break;
        }
        test_file(filename);
    }
}

void test_file(char filename[20])
{
    FILE *fp,*testfile;
    char p_data[20];
    int
port,i,temp_c,read_statu,data_item,ask_sign=0,find_file_end=0;
    char data[20];

    if (access("test.dat",00)==0)
        remove("test.dat");

    if ((fp=fopen(filename,"r+"))==NULL){
        printf("cannot open file\n");
        return;
    }
    data[0]='\0';
    for(;;)
    {
        data_item=0;
        for (;;)
        {
            if ((p_data[data_item]=getc(fp))==EOF)
            {
                find_file_end=0;
                break;
            }
            if (p_data[data_item]==10 ||
p_data[data_item]==13)
            {
                p_data[data_item]='\r';
                data_item++;
                p_data[data_item]=0;
                break;
            }
            data_item++;
        }
    }
}

```

```

        if (find_file_end==1)
            break;
        if(sio_write(7,p_data,data_item+1)<=0)
        {
            printf("port #%d write error !\n",7);
            exit(0);
        }
        read_statu=0;
        ask_sign++;
        while(read_statu==0)
        {
            port=7;
            if ((i = sio_read(port,data,100)) < 0)
            {
                printf("Port #%d READ error,i=%d !\n",
port,i);
                exit(0);
            }
            else if(i)
            {
                data[i] = 0;
                printf("%s",data);
                test_save(data);
                if((testfile=fopen("test.dat","a"))==NULL)
                {
                    printf("Can not open file\n");
                    getch();
                    exit(1);
                }
                fprintf(testfile,"%s",data);
                fclose(testfile);

                if (ask_sign==1)
                {
                    if (strchr(data,'\r')!=NULL)
                        read_statu=1;
                }
                if (strchr(data,'?')!=NULL)
                    read_statu=1;
                if (strchr(data,'*')!=NULL && ask_sign>2)
                    read_statu=2;
            }
        }
        if (read_statu==2)
            break;
        read_statu=0;
        /*      test_save(data); */
        strcpy(p_data,"");
    }
    fclose(fp);
    command_test();
    exit(0);
}

void test_save(char data[20])
{
    FILE *testfile;
    char filename[20],num[4];
    int x;

```

```

        if((testfile=fopen("test.dat","w+"))==NULL)
        {
                printf("can not open file\n");
                getch();
                exit(1);
        }
        /*      for (x=1;x<=500;x++);
        printf("Enter the Line_Num: ");
        fscanf(stdin,"%s",num);      */
        fprintf(testfile,"%s",data);
        fclose(testfile);
}

void command_test()
{
        /* send the command to the controller      */
        char  str[20], endch='x';
        int  driver,mode;

        for( ; ; ) {
                if( endch=='?' ) {
                        // printf("\nPlease input data: " );
                        gets(str);
                        strcat(str,"\r");
                        sio_write(7,str,strlen(str));
                        endch=get_data(str);
                }
                printf("\nInput the command_line:\n");
                gets(str);
                if(fileptr!=NULL)  fclose(fileptr);
                if(!strcmp(str,"esc"))  main();
                /* the exit of the unlimited recursion      */
                strcat(str,"\r");
                sio_write(7,str,strlen(str)); /* send the command to puma
controller
                                                and the controller runs the command */
                endch=get_data(str);
        }
}

void audit_com()
{
        int      i, port, port1;
        int      card_no, total_port;
        char     data[20], filename[20];
        char     w_data[20];
        char     ch;

        restorecrtmode();
        clrscr();
        printf("Reset the sio cards....\n");
        if ((card_no = sio_reset()) == 0)
        {
                printf("No Card Found !\n");
                exit(0);
        }
        printf("Total Card : %d\n", card_no);

        printf("Read the ID....\n");
        for (i = 1; i <= card_no; i++)
        {

```

```

        printf("(%d)\tSerial no: %d", i, sio_id(i));
        printf("\tMapping : %X\n", sio_bank(i));
    }

    printf("Setting port 3 to 300, N, 7, 1....\n");
    port = 3;
    if ( sio_ioctl(3, B300, BIT_7 | P_EVEN | STOP_1) != 0)
    {
        printf("Port #%d IOCTL error !\n", port);
        exit(0);
    }

    clrscr();
    printf("\nDo you want to receive file from the audit
machine?\n");
    printf("        'Y' or 'N' ?");
    do
    {
        ch=getch();
        switch(ch)
        {
            case 'Y':
            case 'y':
                receiving_com();
                exit(0);
                break;

            case 'N':
            case 'n':
                transmit_com();
                break;
            default:
                break;
        }
    }while(ch!='y' && ch!='Y' && ch!='n' && ch!='N');
}

void receiving_com()
{
    int    i, port, port1, read_statu=0;
    int    card_no, total_port;
    char   data[20], filename[20];
    char   w_data[20];
    char   ch;
    FILE   *testfile;

    if((testfile=fopen("test.dat", "w"))==NULL) {
        printf("can not open file\n");
        getch();
        exit(1);
    }
    printf("\nReceive File from AUDIT Lathe Machine....\n");

    read_statu=0;
    for(;;)
    {
        port=3;
        if ((i = sio_read(port, data, 100)) < 0) {
            printf("Port #%d READ error !\n", port);
            exit(0);
        }
        else if(i) {

```

```

        data[i] = 0;
        printf("%s",data);
        fprintf(testfile,"%s",data);
        if (strchr(data,'\n')!=NULL)
            read_statu++;
        if (strchr(data,'%')!=NULL && read_statu>1)
            break;
    }
}

fclose(testfile);
}

void transmit_com()
{
    int    c;

    printf("\nDo you want to transmit file to the audit
machine?\n");
    printf("        'Y' or 'N' ?");
    do
    {
        c=getch();
        switch(c)
        {
            case'Y':
            case'y':
                transmit0_com();
                break;

            case'N':
            case'n':
                menu();
                exit(0);
                break;
            default:
                exit(0);
                break;
        }
    }while(c!='Y' && c!='y' && c!='N' && c!='n');
}

void transmit0_com()
{
    int    i, port, port1;
    char   data[20],filename[20];
    char   w_data[100];

    printf("\nTransmit File to AUDIT Lathe Machine...\n");
    for(;;)
    {
        printf("\n\nInput the program file name:");
        scanf("%s",filename);
        if (strcmp(filename,"esc")==0 || strcmp(filename,"ESC")==0)
            menu();
        test_file0(filename);
    }
}
}

```

```

void test_file0(char filename[20])
{
    FILE *fp;
    char p_data[20];
    int port,i,temp_c,read_statu,data_item,find_file_end=0;
    char test_in[50];
    char data[20];

    if ((fp=fopen(filename,"r+"))==NULL){
        printf("cannot open file/n");
        return;
    }
    do
    {
        data_item=0;
        for (;;)
        {
            if ((p_data[data_item]=getc(fp))==EOF)
            {
                find_file_end=1;
                break;
            }

            if (p_data[data_item]==10 ||
p_data[data_item]==13)
            {
                p_data[data_item]='\n';
                data_item++;
                p_data[data_item]=0;
                break;
            }
            data_item++;
        }

        if (find_file_end==1)
            break;
        if(sio_write(3,p_data,data_item+1)!=0)
        {
            printf("port %#d write error !\n",2);
            exit(0);
        }

        }while(find_file_end!=1);
    fclose(fp);
    menu();
    exit(0);
}

void bpot_com()
{
    int i,port, port1;
    int card_no, total_port;
    char data[20],filename[20],temp_data[20],old_data[20];
    char w_data[20];
    char ch,*ptr;

    restorecrtmode();
    clrscr();

```



```

printf("Reset the sio cards....\n");
if ((card_no = sio_reset()) == 0)
{
    printf("No Card Found !\n");
    exit(0);
}
printf("Total Card : %d\n", card_no);

printf("Read the ID....\n");
for (i = 1; i <= card_no; i++)
{
    printf("(%d)\tSerial no: %d", i, sio_id(i));
    printf("\tMapping : %X\n", sio_bank(i));
}

printf("Setting port 4 to 9600, N, 8, 1....\n");
port = 4;
if ( sio_ioctl(4, B9600, BIT_7 | P_EVEN | STOP_2) != 0)
{
    printf("Port # %d IOCTL error !\n", port);
    exit(0);
}

printf("read line lctrl status \n");
port=4;
if(sio_lctrl(4,C_RTS|C_DTR)<0){
    printf("port # %d lctrl error!\n",port);
    exit(0);
}
printf("port 4 line lctrl
status:%04x\n",4,sio_lctrl(4,C_RTS|C_DTR));

printf("read line status from port 4 \n");
port=4;
printf("port 4 line
status:%04x\n",4,sio_lstatus(4));

clrscr();
printf("\nDo you want to receive file from the brigeport
machine?\n");
printf("      'Y' or 'N' ?");
do
{
    ch=getch();
    switch(ch)
    {
        case'Y':
        case'y':
            receivingl_com();
            exit(0);
            break;

        case'N':
        case'n':
            transmitl_com();
            menu();
            break;
        default:
            break;
    }
}

```

```

        }while(ch!='y' && ch!='Y' && ch!='n' && ch!='N');
    }

void receiving1_com()
{
    int i, j, port, port1, read_statu=0;
    int card_no, total_port;
    char    data[20],filename[20];
    char    w_data[20], *ptr;
    char    ch, temp_data[20], old_data[20];
    FILE    *testfile;

    if((testfile=fopen("test.dat","w"))==NULL) {
        printf("can not open file\n");
        getch();
        exit(1);
    }

    printf("\nReceive File from Bridge PORT Milling Machine....\n");
    read_statu=0;
    for(;;) {
        port=4;
        if ((i = sio_read(port, data, 100)) < 0) {
            printf("Port #%d READ error !\n", port);
            exit(0);
        }
        else if(i) {
            data[i] = 0;
            printf("%s",data);
            fprintf(testfile,"%s",data);

            strcpy(temp_data,data);
            if (old_data[strlen(old_data)-1]=='E') {
                if (temp_data[0]=='N' && temp_data[1]=='D')
                    read_statu=1;
            }
            if (old_data[strlen(old_data)-2]=='E' &&
old_data[strlen(old_data)-1]=='N')
            {
                if (temp_data[0]=='D')    read_statu=1;
            }
            if (read_statu!=1) {
                for (;;) {
                    if ((ptr=strchr(temp_data,'E'))==NULL)
                        break;
                    if (strlen(ptr)<3)
                        break;
                    if (ptr[1]=='N' && ptr[2]=='D') {
                        read_statu=1;
                        break;
                    }
                    ptr=strchr(ptr,ptr[1]);
                    strcpy(temp_data,ptr);
                }
            }
            for (i=1;i<=10000;i++)    i++;
            if (read_statu==1) {
                if (strchr(temp_data,'\n')!=NULL)
                    break;
            }
        }
        strcpy(old_data,data);
    }
}

```

```

    }
    }
    fclose(testfile);
}

void transmit1_com()
{
    int    c;

    printf("\nDo you want to transmit file to the bridgeport
machine?\n");
    printf("      'Y' or 'N' ?");
    do
    {
        c=getch();
        switch(c)
        {
            case'Y':
            case'y':
                transmit01_com();
                menu();
                break;

            case'N':
            case'n':
                menu();
                exit(0);
                break;
            default:
                exit(0);
                break;
        }
    }while(c!='Y' && c!='y' && c!='N' && c!='n');
}

void transmit01_com()
{
    int    i, port, port1;
    char   data[20],filename[20];
    char   w_data[20];

    printf("\nTransmit File to BridgePort Milling Machine....\n");
for(;;)
{
    printf("\n\nInput the program file name:");
    scanf("%s",filename);
    if (strcmp(filename,"esc")==0 || strcmp(filename,"ESC")==0)
        break;
    test_file1(filename);
}

}

void test_file1(char filename[20])
{
    FILE *fp;
    char p_data[20];
    int port,i,temp_c,read_statu,data_item,find_file_end=0;
    char test_in[50];
    char data[20];

```

```

if ((fp=fopen(filename,"r+"))==NULL){
    printf("cannot open file/n");
    return;
}

    data_item=0;
    p_data[data_item]=02;
    data_item++;
    p_data[data_item]=0;
    sio_write(4,p_data,data_item+1);
    data_item=0;
    p_data[data_item]=0;
    sio_write(4,p_data,data_item+1);

do
{
    data_item=0;
    p_data[data_item]=13;
    data_item++;
    p_data[data_item]=13;
    data_item++;
    p_data[data_item]=13;
    data_item++;
    p_data[data_item]=10;
    data_item++;
    p_data[data_item]=0;
    sio_write(4,p_data,data_item);
    data_item=0;
    for (;;)
    {
        if ((p_data[data_item]=getc(fp))==EOF)
        {
            find_file_end=1;
            break;
        }
        if (p_data[data_item]==10 ||
p_data[data_item]==13)
        {
            p_data[data_item]=0;
            break;
        }
        data_item++;
    }
    if(find_file_end==1)
    break;
    if(sio_write(4,p_data,data_item+1)!=0)
    {
        printf("port #%d write error !\n",2);
        exit(0);
    }
}while(find_file_end!=1);
    p_data[data_item]=03;
    data_item++;
    p_data[data_item]=0;
    sio_write(4,p_data,data_item);

    fclose(fp);
    menu();
    exit(0);
}

nett_com()

```

```

{

int      i,j, port, port1;
int      card_no, total_port;
char     data[100],filename[20];
char     w_data[10];
char     ch,c[2];

restorecrtmode();
clrscr();
printf("Reset the sio cards...\n");
if ((card_no = sio_reset()) == 0)
{
    printf("No Card Found !\n");
    exit(0);
}
printf("Total Card : %d\n", card_no);

printf("Read the ID...\n");
for (i = 1; i <= card_no; i++)
{
    printf("(%d)\tSerial no: %d", i, sio_id(i));
    printf("\tMapping   : %X\n", sio_bank(i));
}

printf("Setting port 0 to 9600, N, 8, 1...\n");
port = 0;
if ( sio_ioctl(0, B9600, BIT_8 | P_NONE | STOP_2) != 0)
{
    printf("Port #%d IOCTL error !\n", port);
    exit(0);
}

clrscr();
printf("\nDo you want to receive file or command from Network
Computer? 'Y' or 'N' ?");
do
{
    ch=getch();
    switch(ch)
    {
        case'Y':
        case'y':
            sio_write(0,"s",1);
            do{
                sio_read(0,c,1);
            }while(!(strchr(c,'.')!=NULL));
            rec_com();
            break;

        case'N':
        case'n':
            trans_com();
            break;
        default:
            exit(0);
            break;
    }
}while(ch!='y' && ch!='Y' && ch!='n' && ch!='N');
return 0;
}

```

```

void rec_com()
{
    FILE          *fp;
    char          ch, fname[20], data[20];
    int          port, i;
    union{
        char      c[2];
        unsigned int count;
    }cnt;
    printf("\n\nReceive File or command from Network
Computer....\n");

    get_file_name(fname);

    printf("Receiving file %s\n", fname);
    remove(fname);
    if((fp=fopen(fname, "w"))==NULL){
        printf("cannot open input file\n");
        exit(0);
    }
    sio_write(0, ".", 2);
    sio_read(0, data, 1);
    cnt.c[0]=data[0];
    sio_write(0, ".", 2);
    sio_read(0, data, 1);
    cnt.c[1]=data[0];
    sio_write(0, ".", 2);
    for(; cnt.count; cnt.count--)
    {
        if ((i = sio_read(0, data, 20)) < 0) {
            printf("Port #%d READ error !\n", 5);
            exit(0);
        }
        else if(i)
        {
            data[i] = 0;
            printf("%s", data);
            fputs(data, fp);
        }
    }
    fclose(fp);
    printf("\nFile %s has been received.\n");
    printf("\nPress any key to continue.\n");
}

```

```

void          get_file_name(f)
char          *f;
{
char          ch[2], data[20], temp[2];
int          i;

printf("Receiver Waiting...\n");
do{
sio_read(0, data, 2);
}while(!(strchr(data, '?')!=NULL));
sio_write(0, ".", 1);
for(;;) {

```

```

        sio_write(0, ".", 1);
        do{
        sio_read(0, data, 2);
        *f=data[0];
        ch[0]=*f;
        }while(ch[0]!='?');
        for(i=1;i<=30000;i++)    i++;
        f++;
        if(ch[0]=='\0')
        break;
    }
}

void    wait()

{
    char    c[2];
    c[0]='\0';
    do{
        sio_read(0, c, 2);
        }while(c[0]!='?');
}

void trans_com()
{
    int    c;
    char    ch[2];
    printf("\n\n\nDo you want to transmit file to Network
Computer? 'Y' or 'N' ?");
    do
    {
        c=getch();
        switch(c)
        {
            case'Y':
            case'y':
                sio_write(0, "r", 1);
                ch[0]='\0';
                do{
                sio_read(0, ch, 1);
                }while(!(strchr(ch, '.')!=NULL));
                trans0_com();
                break;

            case'N':
            case'n':
                exit(0);
                break;
            default:
                exit(0);
                break;
        }
        }while(c!='Y' && c!='y' && c!='N' && c!='n');
}

void trans0_com()
{
    FILE    *fp;
    char    fname[20], buffer[2], data[20], ch[2];

```

```

int      i,data_item;

union {
    char    c[2];
    unsigned int count;
} cnt;
send_file_name(fname);
buffer[0]='\0';
if((fp=fopen(fname,"r"))==NULL){
    printf("cannot open input file\n");
    exit(0);
}
ch[0]='\0';
printf("sending file %s\n",fname);
cnt.count=filesize(fp);
printf("File size %d\n",cnt.count);
ch[0]=cnt.c[0];
sio_write(0,ch,1);
wait();
ch[0]=cnt.c[1];
sio_write(0,ch,1);
do{
    buffer[0]=getc(fp);
    if(ferror(fp)) {
        printf("error reading input file");
        break;
    }
    if(!feof(fp)){
        for(i=1;i<=30000;i++)    i++;
        sio_write(0,buffer,1);
    }
} while(!feof(fp));
fclose(fp);
printf(" \n\nThe file have been sent out\n");
}

unsigned int filesize(fp)
FILE      *fp;
{
    unsigned long int i;

    i=0;
    do{
        getc(fp);
        i++;
    }while(!feof(fp));
    rewind(fp);
    return i-1;
}

void      send_file_name(fname)
char      *fname;
{
    char buffer[2],ch[2];
    printf("\n\nTransmitter Waiting...\n");
    printf("\n\nInput the file name:");
    scanf("%s",fname);
    if (strcmp(fname,"esc")==0 || strcmp(fname,"ESC")==0)
        exit(0);
}

```



```

do {
    sio_write(0,"?",1);
    sio_read(0,ch,1);
} while(!(strchr(ch, '.')!=NULL));
printf("\n\nsending filename %s\n",fname);
for(;;) {
    do{
        buffer[0]=*fname;
        sio_write(0,&buffer,1);
        sio_read(0,ch,1);
    } while(ch[0]!='. ');
    fname++;
    if(*fname=='\0') {
        sio_write(0,'\0',1);
        printf(" \n\nThe filename have been sent out\n");
        break;
    }
}
}

get_data(char *cmdline) {
    int port, i,j=1, rlt=1;
    char anychar, endch='x', data[100], *p,ch;
    do {
        port=7;
        if ((i= sio_read(port, data, 100)) < 0) {
            /* read the result after the controller run the command str
*/
            printf("Port #%d READ error !\n", port);
            exit(0);
        }
        if ( (i==0)&(j==0)&(endch=='.')) rlt=0;
            /* the exit of the unlimited recursion */
        if(i) {
            j=0;
            data[i]=0;
            printf("%s",data);
            endch=data[i-1]; /* set the end status */
            p=strchr(data, '?');
            if (p!=NULL ) return '?';
        }
    } while ( rlt);

    if ( (cmdline[0]=='l')||(cmdline[0]=='L') ) {
        printf(" save? (Y/N): ");
        anychar=getch();
        if ( (anychar=='y')||(anychar=='Y') ) {
            sio_write(7,cmdline,strlen(cmdline));
            write_data(cmdline);
        }
    }
    return endch;
}

write_data(char * cmdline)
{
    int port, i,j=1, rlt=1;
    char endch='x', data[100];
    char *fname, *filename;
    FILE *fp;

```

```

filename= strchr(cmdline, ' ');
split_space(fname, filename);
if ( (fp=fopen(fname, "w+") )==NULL) {
    printf("Cannot open file");
    exit(1);
}
do {
    port=7;
    if ((i= sio_read(port, data, 100)) < 0)      {
        /* read the result after the controller run the command str
*/
        printf("Port #%d READ error !\n", port);
        exit(0);
    }
    if ( (i==0)&(j==0)&(endch=='.')) rlt=0;
        /* the exit of the unlimited recursion */
    if(i) {
        j=0;
        data[i]=0;
        /* printf(" I am ZJX          "); */
        fprintf(fp, "%s", data);
        endch=data[i-1]; /* set the end status */
    }
} while ( rlt);
fileptr=fp;
return 0;
}

```

```

void split_space(char * out, char* str)
{
    int t=0;
    do {

        if (isspace(str[t])) t++;
        else {
            *out=str[t];
            t++;
            out++;
        }
    } while(! str[t]=='\0');
    *out='\0';
}

```

```

assembly()
{
    char *power;
    int k;
    float delay1;

    cleardevice();
    restorecrtmode();

    robot_init();

    // monitor_com();

    printf("\nPlease input your name: ");
    gets(name);
    strlwr(name);
}

```

```

pos1=strlen(name);

printf("\n");

for (i=0; i<=pos1-1; i++)
  { strname[i]='?'; }

charact();

for (i=0; i<=pos1-1; i++)
  { midvar[i]=strname[i]; }

strev(name);
charact();
strcpy (namexu,"do move home");
monitor_com();
strcpy (namexu,"ex main3");
monitor_com();
strcpy (namexu,"ex main2");
monitor_com();

// printf("The Input Name is  \n ");

for (i=0; i<=pos1-1; i++)
  {
    if ( midvar[i]=='?')
      {
        midvar[i]=strname[pos1-i-1];
      }
  }

// printf("%c", midvar[i]);
if (i==0)
  { strcpy (namexu,"do set lo=p21");
    monitor_com();
    for (k=1; k<=100; k++) // this is for delay.
      {
        delay1=sin (k);
      }
    chmove();
  }
if (i==1)
  { strcpy (namexu,"do set lo=p22");
    monitor_com();
    for (k=1; k<=100; k++) // this is for delay.
      {
        delay1=sin (k);
      }
    chmove();
  }
if (i==2)
  { strcpy (namexu,"do set lo=p23");
    monitor_com();
    for (k=1; k<=100; k++) // this is for delay.
      {
        delay1=sin (k);
      }
    chmove();
  }
if (i==3)
  { strcpy (namexu,"do set lo=p24");
    monitor_com();
  }

```

```

    for (k=1; k<=100; k++) // this is for delay.
    {
        delay1=sin (k);
    }
    chmove();
}
if (i==4)
{ strcpy (namexu,"do set lo=p25");
  monitor_com();
  for (k=1; k<=100; k++) // this is for delay.
  {
      delay1=sin (k);
  }
  chmove();
}
    if (i==5)
{ strcpy (namexu,"do set lo=p26");
  monitor_com();
  for (k=1; k<=100; k++) // this is for delay.
  {
      delay1=sin (k);
  }
  chmove();
}
if (i==6)
{ strcpy (namexu,"do set lo=p27");
  monitor_com();
  for (k=1; k<=100; k++) // this is for delay.
  {
      delay1=sin (k);
  }
  chmove();
}
if (i==7)
{ strcpy (namexu,"do set lo=p28");
  monitor_com();
  for (k=1; k<=100; k++) // this is for delay.
  {
      delay1=sin (k);
  }
  chmove();
}
if (i==8)
{ strcpy (namexu,"do set lo=p29");
  monitor_com();
  for (k=1; k<=100; k++) // this is for delay.
  {
      delay1=sin (k);
  }
  chmove();
}
if (i==9)
{ strcpy (namexu,"do set lo=p20");
  monitor_com();
  for (k=1; k<=100; k++) // this is for delay.
  {
      delay1=sin (k);
  }
  chmove();
}
}

```

```

    }
    main();

    getch();
    return 0;

}

charact()
{
    for (i =1; i<=28; i++)          /* locat the charaters and
position */
    { if (i==1)
      { chv='a';
        namecomp();
      }

      if (i==2 )
      { chv='b';
        namecomp();
      }

      if (i==3)
      { chv='c';
        namecomp();
      }

      if (i==4)
      { chv='d';
        namecomp();
      }

      if (i==5)
      { chv='e';
        namecomp();
      }

      if (i==6)
      { chv='f';
        namecomp();
      }

      if (i==7 )
      { chv='g';
        namecomp();
      }
      if (i==8 )
      { chv='h';
        namecomp();
      }
      if (i==9 )
      { chv='i';
        namecomp();
      }
      if (i==10 )

```

```
{ chv='j';
  namecomp();
}
if (i==11 )
{ chv='k';
  namecomp();
}
if (i==13 )
{ chv='l';
  namecomp();
}
if (i==13 )
{ chv='m';
  namecomp();
}
if (i==14 )
{ chv='n';
  namecomp();
}
if (i==15 )
{ chv='o';
  namecomp();
}
if (i==16 )
{ chv='p';
  namecomp();
}
if (i==17 )
{ chv='q';
  namecomp();
}
if (i==18 )
{ chv='r';
  namecomp();
}
if (i==19 )
{ chv='s';
  namecomp();
}
if (i==20 )
{ chv='t';
  namecomp();
}
if (i==21 )
{ chv='u';
  namecomp();
}
if (i==22 )
{ chv='v';
  namecomp();
}
if (i==23 )
{ chv='w';
  namecomp();
}
if (i==24 )
{ chv='x';
  namecomp();
}
if (i==25 )
{ chv='y';
```

```

        namecomp();
    }
    if (i==26 )
    { chv='z';
      namecomp();
    }
    if (i==27 )
    { chv=' ';
      namecomp();
    }
    if (i==28 )
    { chv='.';
      namecomp();
    }
}
return 0;
}

```

```
chmove()
```

```

{
int k;
float delay1;

if (midvar[i]=='a')
{ strcpy (namexu,"ex aa");
  monitor_com();
  for (k=1; k<=100; k++) // this is for delay.
  {
    delay1=sin (k);
  }
}
if (midvar[i]=='b')
{ strcpy (namexu,"ex bb");
  monitor_com();
  for (k=1; k<=100; k++) // this is for delay.
  {
    delay1=sin (k);
  }
}
if (midvar[i]=='c')
{ strcpy (namexu,"ex cc");
  monitor_com();
  for (k=1; k<=100; k++) // this is for delay.
  {
    delay1=sin (k);
  }
}
if (midvar[i]=='d')
{ strcpy (namexu,"ex dd");
  monitor_com();
  for (k=1; k<=100; k++) // this is for delay.
  {
    delay1=sin (k);
  }
}
if (midvar[i]=='e')
{ strcpy (namexu,"ex ee");

```

```

    monitor_com();
    for (k=1; k<=100; k++) // this is for delay.
    {
        delay1=sin (k);
    }
}
if (midvar[i]=='f')
{ strcpy (namexu,"ex ff");
  monitor_com();
  for (k=1; k<=100; k++) // this is for delay.
  {
      delay1=sin (k);
  }
}
if (midvar[i]=='g')
{ strcpy (namexu,"ex gg");
  monitor_com();
  for (k=1; k<=100; k++) // this is for delay.
  {
      delay1=sin (k);
  }
}
if (midvar[i]=='h')
{ strcpy (namexu,"ex hh");
  monitor_com();
  for (k=1; k<=100; k++) // this is for delay.
  {
      delay1=sin (k);
  }
}
if (midvar[i]=='i')
{ strcpy (namexu,"ex ii");
  monitor_com();
  for (k=1; k<=100; k++) // this is for delay.
  {
      delay1=sin (k);
  }
}
if (midvar[i]=='j')
{ strcpy (namexu,"ex jj");
  monitor_com();
  for (k=1; k<=100; k++) // this is for delay.
  {
      delay1=sin (k);
  }
}
if (midvar[i]=='k')
{ strcpy (namexu,"ex kk");
  monitor_com();
  for (k=1; k<=100; k++) // this is for delay.
  {
      delay1=sin (k);
  }
}
if (midvar[i]=='l')
{ strcpy (namexu,"ex ll");
  monitor_com();
  for (k=1; k<=100; k++) // this is for delay.
  {
      delay1=sin (k);
  }
}

```



```

}
if (midvar[i]=='m')
{ strcpy (namexu,"ex mm");
  monitor_com();
  for (k=1; k<=100; k++) // this is for delay.
  {
    delay1=sin (k);
  }
}
if (midvar[i]=='n')
{ strcpy (namexu,"ex nn");
  monitor_com();
  for (k=1; k<=100; k++) // this is for delay.
  {
    delay1=sin (k);
  }
}
if (midvar[i]=='o')
{ strcpy (namexu,"ex oo");
  monitor_com();
  for (k=1; k<=100; k++) // this is for delay.
  {
    delay1=sin (k);
  }
}
if (midvar[i]=='p')
{ strcpy (namexu,"ex pp");
  monitor_com();
  for (k=1; k<=100; k++) // this is for delay.
  {
    delay1=sin (k);
  }
}
if (midvar[i]=='q')
{ strcpy (namexu,"ex qq");
  monitor_com();
  for (k=1; k<=100; k++) // this is for delay.
  {
    delay1=sin (k);
  }
}
if (midvar[i]=='r')
{ strcpy (namexu,"ex rr");
  monitor_com();
  for (k=1; k<=100; k++) // this is for delay.
  {
    delay1=sin (k);
  }
}

if (midvar[i]=='s')
{ strcpy (namexu,"ex ss");
  monitor_com();
  for (k=1; k<=100; k++) // this is for delay.
  {
    delay1=sin (k);
  }
}
if (midvar[i]=='t')
{ strcpy (namexu,"ex tt");
  monitor_com();
}

```

```

    for (k=1; k<=100; k++) // this is for delay.
    {
        delay1=sin (k);
    }
}
if (midvar[i]=='u')
{ strcpy (namexu,"ex uu");
  monitor_com();
  for (k=1; k<=100; k++) // this is for delay.
  {
      delay1=sin (k);
  }
}
if (midvar[i]=='v')
{ strcpy (namexu,"ex vv");
  monitor_com();
  for (k=1; k<=100; k++) // this is for delay.
  {
      delay1=sin (k);
  }
}
if (midvar[i]=='w')
{ strcpy (namexu,"ex ww");
  monitor_com();
  for (k=1; k<=100; k++) // this is for delay.
  {
      delay1=sin (k);
  }
}
if (midvar[i]=='x')
{ strcpy (namexu,"ex xx");
  monitor_com();
  for (k=1; k<=100; k++) // this is for delay.
  {
      delay1=sin (k);
  }
}
if (midvar[i]=='y')
{ strcpy (namexu,"ex yy");
  monitor_com();
  for (k=1; k<=100; k++) // this is for delay.
  {
      delay1=sin (k);
  }
}
if (midvar[i]=='z')
{ strcpy (namexu,"ex zz");
  monitor_com();
  for (k=1; k<=100; k++) // this is for delay.
  {
      delay1=sin (k);
  }
}
}

namecomp()
{

```

```

char *ptr;

// clrscr();

strcpy(str, name);
ptr = (char *) memchr(str, chv, strlen(str));
if (ptr)
{
// printf("The character  %c", chv, " is at position: %d\n", ptr
- str);
// printf("\n present string is      %s ", ptr);
name1=ptr;
// printf("the name1 is      %s\n", name1);
pos=ptr-str;
// printf("the pos is%d\n", pos);
strname[pos]=chv;
}
// else
// printf("The character was not found\n");

// getch();
return 0;
}
void robot_init()
{

int          i,j, port, port1,stat,stat1;
int          card_no, total_port,data_item;
char         data[20],filename[20];
char         w_data[20],command[20];
char         power, ch,str[20];

restorecrtmode();
clrscr();
printf("Reset the moxa cards....\n");
if ((card_no = sio_reset()) == 0)
{
printf("No Card Found !\n");
exit(0);
}
printf("Total Card : %d\n", card_no);

printf("Read the ID....\n");
for (i = 1; i <= card_no; i++)
{
printf("(%d)\tSerial no: %d", i, sio_id(i));
printf("\tMapping   : %X\n", sio_bank(i));
}

printf("Setting port 7 to 9600, N, 8, 1....\n");
port = 7;
if ( sio_ioctl(port, B9600, BIT_8 | P_NONE | STOP_1) !=
0)
{
printf("Port #%d IOCTL error !\n", port);
exit(0);
}
for(port=6;port<=13;port++)
{
stat=sio_open(port);
if(stat != 0)

```

```

        {printf("Port #%d can not be opened!\n",port);
        exit(0);}
    }

printf("Do you want to run the power on program? \n");
power=getch();
if ( power=='Y' || power=='y')
{

    printf("Read information from PUMA Controller....\n");
    do
    {
        while(kbhit()==0)
        {
            port=7;
            stat1=sio_read(port,data,100);

            if(stat1>=0)
            {
                data[stat1]=0;
                printf("%s",data);
            }

        }

        do
        {
            ch=getch();
            switch(ch)
            {
                case 'N':
                case 'n':
                    w_data[0] = ch;
                    w_data[1] = '\r';
                    sio_write(7,w_data,2);

                    break;
                default:break;
            }
        }while(ch!='n' && ch!='N' && ch!='.' && ch!='y' &&
ch!='Y');
        }while(ch!='.' && ch!='y');
    } // for power switch.

    // strcpy (namexu,"cal");
    // monitor_com();

}

monitor_com()

{
    /* send the monitor command to the controller, It accepts
    the monitor varibale "str1" and send to Robot Controller
    */
    char endch='x';

```

```

// for( ; ;) {
// if( endch=='?') {
//     /*printf("\nPlease input data: " ); */
//     // gets(str1);
//     // strcat(str1,"\r");
//     // sio_write(7,str1,strlen(str1));
//     // endch=get_data(str1);
// }
// printf("\nInput the command_line:\n");
strcpy (str1, namexu);
if(fileptr!=NULL) fclose(fileptr);
// if(!strcmp(str1,"esc")) menu();
//     /* the exit of the unlimited recursion */
strcat(str1,"\r");
sio_write(7,str1,strlen(str1)); /* send the command to puma
controller
and the controller runs the command */
endch=get_data(str1);

// }
}

```

```

void mainn()
{
int x1,y1,c;
/* int driver,mode;
register int i;
driver=DETECT;
mode=0;
initgraph(&driver,&mode,"c:\\tc\\bgi"); */

settextstyle(TRIPLEX_FONT,HORIZ_DIR,4);
setbkcolor(CYAN);
outtextxy(200,120,"WELCOME TO");
outtextxy(145,220,"LANSING ROBOT WORLD");
settextstyle(TRIPLEX_FONT,HORIZ_DIR,1);
outtextxy(250,400,"CLICK TO START");
do
{
get_mouse();
if(buttons==1)
{
cleardevice();
menuu();
}
}while(buttons!=3);
/* getch();
cleardevice();
menuu(); */

}
void menuu()
{
int c;
/* int driver,mode;
register int i;
driver = DETECT;
mode = 0;
initgraph(&driver,&mode,"c:\\tc\\bgi"); */

```

```

setbkcolor(LIGHTRED);
settextstyle(TRIPLEX_FONT,HORIZ_DIR,1);
outtextxy(120,50,"Selection of Lansing Robot Control ");
setfillstyle(SOLID_FILL,CYAN);
bar3d(128,144,488,184,0,0);
outtextxy(140,150,"1. Mouse and Screen Method");
setfillstyle(SOLID_FILL,CYAN);
bar3d(128,192,488,232,0,0);
outtextxy(140,200,"2. User Friendly Teaching Method");
setfillstyle(SOLID_FILL,CYAN);
bar3d(128,240,488,280,0,0);
outtextxy(140,250,"3. EXIT");
outtextxy(210,400," Enter Your Choice ");
gotoxy(1,1);
show_mouse();
do
{
  get_mouse();
  if(x1>=16&&x1<=61)
  {
    if(y1>=18&&y1<=23&&(buttons==1||buttons==2))
    {
      hide_mouse();
      cleardevice();
      mouse();
    }
    if(y1>=24&&y1<=29&&(buttons==1||buttons==2))
    {
      hide_mouse();
      closegraph();
      main11();
    }
    if(y1>=30&&y1<=35&&(buttons==1||buttons==2))
    {
      hide_mouse();
      cleardevice();
      main();
    }
  }
}while(buttons!=3);
/* do
{
  c=getch();
  switch(c)
  {
    case'1':
      cleardevice();
      mouse();
      break;
    case'2':
      closegraph();
      main11();
      break;
    case'3': cleardevice();
      main();
      break;
    default:
      break;
  }
}while(c!='1' && c!='2' && c!='3'); */
}

```

```

void mouse()
{ /* request auto detection */
/*   int driver = DETECT, mode, errorcode; */
   int i;

   /* initialize graphics, local variables*/
  //   initgraph(&driver, &mode, "c:\\tc\\bgi");

  /* read result of initialization */
  //   errorcode = graphresult();
  //   if (errorcode != grOk) /* an error occurred */
  //   {
  //       printf("Graphics error: %s\n", grapherrormsg(errorcode));
  //       printf("Press any key to halt:");
  //       getch();
  //       exit(1); /*terminate with error code */
  //   }

/*   midx = getmaxx() / 2;
   midy = getmaxy() / 2;   */

   setbkcolor(BLUE);
   setfillstyle(EMPTY_FILL, getmaxcolor());
   setttextstyle(DEFAULT_FONT, HORIZ_DIR, 1);
   /* draw the 3-d bar */
   setfillstyle(SOLID_FILL, RED);
   bar3d(592, 16, 623, 47, 0, 0);
   outtextxy(593, 30, "EXIT");
   setfillstyle(SOLID_FILL, GREEN);
   bar3d(592, 56, 623, 87, 0, 0);
   outtextxy(596, 64, "JOB");
   outtextxy(593, 76, "SAVE");
   bar3d(592, 96, 623, 127, 0, 0);
   outtextxy(593, 104, "SAVE");
   outtextxy(596, 116, "END");
   setfillstyle(SOLID_FILL, LIGHTGRAY);
   bar3d(16, 152, 87, 183, 0, 0);
   outtextxy(28, 166, "DISP 1");
   bar3d(96, 152, 167, 183, 0, 0);
   outtextxy(108, 166, "DISP 2");
   bar3d(176, 152, 247, 183, 0, 0);
   outtextxy(188, 166, "DISP 3");
   bar3d(256, 152, 327, 183, 0, 0);
   outtextxy(268, 166, "DISP 4");
   bar3d(384, 152, 415, 183, 0, 0);
   outtextxy(392, 166, "UP");
   bar3d(472, 152, 503, 183, 0, 0);
   outtextxy(486, 166, "1");
   bar3d(512, 152, 543, 183, 0, 0);
   outtextxy(526, 166, "2");
   bar3d(552, 152, 583, 183, 0, 0);
   outtextxy(566, 166, "3");
   bar3d(592, 152, 623, 183, 0, 0);
   outtextxy(606, 166, "4");

   bar3d(16, 192, 47, 223, 0, 0);
   outtextxy(17, 200, "CALL");
   outtextxy(30, 212, "n");
   bar3d(56, 192, 87, 223, 0, 0);

```

```

outtextxy(57,200,"CALL");
outtextxy(70,212,"x1");
bar3d(96,192,127,223,0,0);
outtextxy(104,200,"DO");
outtextxy(110,212,"n");
bar3d(136,192,167,223,0,0);
outtextxy(144,200,"DO");
outtextxy(137,212,"WHIL");
bar3d(176,192,207,223,0,0);
outtextxy(190,206,"C");
bar3d(216,192,247,223,0,0);
outtextxy(230,206,"D");
bar3d(256,192,287,223,0,0);
outtextxy(270,206,"E");
bar3d(296,192,327,223,0,0);
outtextxy(310,206,"F");
bar3d(344,192,375,223,0,0);
outtextxy(345,206,"LEFT");
bar3d(384,192,415,223,0,0);
outtextxy(398,206,"H");
bar3d(424,192,455,223,0,0);
outtextxy(425,206,"RIGH");
bar3d(472,192,503,223,0,0);
outtextxy(473,200,"PAGE");
outtextxy(476,212,"CTL");
bar3d(512,192,543,223,0,0);
outtextxy(513,200,"MACH");
outtextxy(513,212,"LOCK");
bar3d(552,192,583,223,0,0);
outtextxy(556,200,"OUT");
outtextxy(560,212,"ON");
bar3d(592,192,623,223,0,0);
outtextxy(593,200,"MEAS");
outtextxy(600,212,"ON");

bar3d(16,232,47,263,0,0);
outtextxy(20,240,"PFM");
outtextxy(30,252,"n");
bar3d(56,232,87,263,0,0);
outtextxy(64,246,"IF");
bar3d(96,232,127,263,0,0);
outtextxy(97,246,"THEN");
bar3d(136,232,167,263,0,0);
outtextxy(137,246,"ELSE");
bar3d(176,232,207,263,0,0);
outtextxy(190,246,"8");
bar3d(216,232,247,263,0,0);
outtextxy(230,246,"9");
bar3d(256,232,287,263,0,0);
outtextxy(270,246,"A");
bar3d(296,232,327,263,0,0);
outtextxy(310,246,"B");
bar3d(384,232,415,263,0,0);
outtextxy(385,246,"DOWN");
bar3d(472,232,503,263,0,0);
outtextxy(473,246,"EDIT");
bar3d(512,232,543,263,0,0);
outtextxy(513,246,"DIAG");
bar3d(552,232,583,263,0,0);
outtextxy(553,246,"TAPE");
bar3d(592,232,623,263,0,0);

```



```

outtextxy(593,246,"MONI");

bar3d(16,272,47,303,0,0);
outtextxy(17,286,"BEGN");
bar3d(56,272,87,303,0,0);
outtextxy(60,286,"END");
bar3d(96,272,127,303,0,0);
outtextxy(97,286,"WAIT");
bar3d(136,272,167,303,0,0);
outtextxy(140,286,"OUT");
bar3d(176,272,207,303,0,0);
outtextxy(190,286,"4");
bar3d(216,272,247,303,0,0);
outtextxy(230,286,"5");
bar3d(256,272,287,303,0,0);
outtextxy(270,286,"6");
bar3d(296,272,327,303,0,0);
outtextxy(310,286,"7");
bar3d(344,272,375,303,0,0);
outtextxy(348,280,"JOB");
outtextxy(345,292,"STEP");
bar3d(384,272,415,303,0,0);
outtextxy(385,280,"PROG");
outtextxy(385,292,"STEP");
bar3d(424,272,455,303,0,0);
outtextxy(428,286,"TWO");
bar3d(472,272,503,303,0,0);
outtextxy(476,280,"JOB");
outtextxy(473,292,"TECH");
bar3d(512,272,543,303,0,0);
outtextxy(513,280,"PROG");
outtextxy(513,292,"TECH");
bar3d(552,272,583,303,0,0);
outtextxy(553,280,"PLAY");
outtextxy(553,292,"BACK");
bar3d(592,272,623,303,0,0);
outtextxy(593,280,"MODE");
outtextxy(606,292,"x1");

bar3d(16,312,47,343,0,0);
outtextxy(20,326,"AND");
bar3d(56,312,87,343,0,0);
outtextxy(64,326,"OR");
bar3d(96,312,127,343,0,0);
outtextxy(100,326,"EOR");
bar3d(136,312,167,343,0,0);
outtextxy(140,326,"NOT");
bar3d(176,312,207,343,0,0);
outtextxy(190,326,"0");
bar3d(216,312,247,343,0,0);
outtextxy(230,326,"1");
bar3d(256,312,287,343,0,0);
outtextxy(270,326,"2");
bar3d(296,312,327,343,0,0);
outtextxy(310,326,"3");
bar3d(344,312,375,343,0,0);
outtextxy(348,326,"JOB");
bar3d(384,312,415,343,0,0);
outtextxy(385,326,"PROG");
bar3d(424,312,455,343,0,0);
outtextxy(428,326,"ONE");

```

```

bar3d(16,352,47,383,0,0);
outtextxy(17,366,"ENTR");
bar3d(56,352,87,383,0,0);
outtextxy(57,366,"CONS");
bar3d(96,352,127,383,0,0);
outtextxy(100,366,"DUP");
bar3d(136,352,167,383,0,0);
outtextxy(137,366,"REMV");
bar3d(176,352,207,383,0,0);
outtextxy(190,360,"/");
outtextxy(177,372,"HEXA");
bar3d(216,352,247,383,0,0);
outtextxy(230,366,"-");
bar3d(256,352,287,383,0,0);
outtextxy(270,366,".");
bar3d(296,352,327,383,0,0);
outtextxy(310,366,"");
bar3d(472,352,543,383,0,0);
outtextxy(488,366,"RESET");
bar3d(552,352,623,383,0,0);
outtextxy(562,366,"RECOVER");

```

```

bar3d(16,392,47,423,0,0);
outtextxy(20,400,"JOB");
outtextxy(20,412,"END");
bar3d(56,392,87,423,0,0);
outtextxy(60,400,"CLR");
outtextxy(60,412,"CTR");
bar3d(96,392,127,423,0,0);
outtextxy(100,400,"ADD");
outtextxy(100,412,"CTR");
bar3d(136,392,167,423,0,0);
outtextxy(140,400,"CHK");
outtextxy(140,412,"CTR");
bar3d(176,392,207,423,0,0);
outtextxy(184,406,"SP");
bar3d(216,392,247,423,0,0);
outtextxy(230,406,"+");
bar3d(256,392,287,423,0,0);
outtextxy(270,406,"*");
bar3d(296,392,327,423,0,0);
outtextxy(310,406,"&");
bar3d(344,392,375,423,0,0);
outtextxy(348,406,"DEL");
bar3d(384,392,415,423,0,0);
outtextxy(388,406,"CHG");
bar3d(424,392,455,423,0,0);
outtextxy(428,406,"ADD");
bar3d(472,392,543,423,0,0);
outtextxy(488,406,"START");
bar3d(552,392,623,423,0,0);
outtextxy(558,406,"SERVO ON");

```

```

bar3d(16,432,47,463,0,0);
outtextxy(20,440,"SEL");
outtextxy(17,452,"MREG");
bar3d(56,432,87,463,0,0);
outtextxy(60,440,"CLR");
outtextxy(57,452,"MREG");
bar3d(96,432,127,463,0,0);

```

```

    outtextxy(100,440,"ADD");
    outtextxy(97,452,"MREG");
    bar3d(136,432,167,463,0,0);
    outtextxy(137,440,"UNIT");
    outtextxy(137,452,"MREG");
    bar3d(176,432,207,463,0,0);
    outtextxy(177,446,"CNCL");
    bar3d(216,432,247,463,0,0);
    outtextxy(220,440,"RUB");
    outtextxy(220,452,"OUT");
    bar3d(256,432,327,463,0,0);
    outtextxy(280,446,"SET");
    bar3d(384,432,415,463,0,0);
    outtextxy(385,446,"TECH");
    bar3d(472,432,543,463,0,0);
    outtextxy(492,446,"STOP");
    bar3d(552,432,623,463,0,0);
    outtextxy(553,446,"SERVO OFF");
/*  settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
    outtextxy(592,23,"UNIT");
    outtextxy(592,35,"MREG");
    outtextxy(592,29,"EXIT");    */
/* int xl,y1,c;
int driver,mode;
register int i;
driver=DETECT;
mode=0;

initgraph(&driver,&mode,"c:\\tc\\bgi");
setbkcolor(LIGHTBLUE);
settextstyle(TRIPLEX_FONT,HORIZ_DIR,1);
restorecrtmode();
clrscr(); */
outtextxy(120,50,"KEYBOARD OF LANSING ROBOT");
/* bar3d(15,15,576,143,0,0);*/
/* bar3d(); */
mouse_function();
}
void mouse_function()
{
    FILE *fp;
    char fn[20];

    gotoxy(1,1);
    show_mouse();
/*  init_mouse();*/
do
{
    get_mouse();
    gotoxy(1,1);
    printf("%d,%d",xl,y1);
    if(xl>=74&&xl<=77&&y1>=2&&y1<=5&&buttons==1)
    {
        /*  cleardevice();
            outtextxy(150,50,"ARE YOU ALRIGHT");
            menuu();    */
            hide_mouse();
            cleardevice();
            menuu();
        /*  exit(0); */
    }
}

```

```

/*  if(x1>=74&&x1<=77&&y1>=7&&y1<=10&&buttons==1)
{
printf("Please input file name: ");
gets(fn);

    if((fp=fopen(fn,"w"))==NULL)
    {
        printf("Can not create file\n");
        return;
    }
    printf("Please input JOB procedure:\n");
} */
if(y1>=19&&y1<=22)
{
    if(x1>=2&&x1<=10&&buttons==1)
    {
        sio_putch(13,50);
        sound(2500);
        delay(200);
        nosound();
        // sound(2500);delay(100);nosound();
        sio_putch(13,0);
    }
    if(x1>=13&&x1<=20&&buttons==1)
    {
        sio_putch(13,194);
        sound(2500);delay(100);nosound();
        sio_putch(13,0);
    }
    if(x1>=22&&x1<=30&&buttons==1)
    {
        sio_putch(13,34);
        sound(2500);delay(100);nosound();
        sio_putch(13,0);
    }
    if(x1>=32&&x1<=40&&buttons==1)
    {
        sio_putch(13,210);
        sound(2500);delay(100);nosound();
        sio_putch(13,0);
    }
    if(x1>=48&&x1<=51&&buttons==1)
    {
        sio_putch(13,207);
        sound(2500);delay(100);nosound();
        sio_putch(13,0);
    }
    if(x1>=59&&x1<=62&&buttons==1)
    {
        sio_putch(13,52);
        sound(2500);delay(100);nosound();
        sio_putch(13,0);
    }
    if(x1>=64&&x1<=67&&buttons==1)
    {
        sio_putch(13,196);
        sound(2500);delay(100);nosound();
        sio_putch(13,0);
    }
    if(x1>=69&&x1<=72&&buttons==1)

```

```

{
  sio_putch(13,36);
  sound(2500);delay(100);nosound();
  sio_putch(13,0);
}
if(x1>=74&&x1<=77&&buttons==1)
{
  sio_putch(13,212);
  sound(2500);delay(100);nosound();
  sio_putch(13,0);
}
}

if(y1>=24&&y1<=27)
{
  if(x1>=2&&x1<=5&&buttons==1)
  {
    sio_putch(13,118);
    sound(2500);delay(100);nosound();
    sio_putch(13,0);
  }
  if(x1>=7&&x1<=10&&buttons==1)
  {
    sio_putch(13,134);
    sound(2500);delay(100);nosound();
    sio_putch(13,0);
  }
  if(x1>=13&&x1<=15&&buttons==1)
  {
    sio_putch(13,102);
    sound(2500);delay(100);nosound();
    sio_putch(13,0);
  }
  if(x1>=17&&x1<=20&&buttons==1)
  {
    sio_putch(13,150);
    sound(2500);delay(100);nosound();
    sio_putch(13,0);
  }
  if(x1>=22&&x1<=25&&buttons==1)
  {
    sio_putch(13,86);
    sound(2500);delay(100);nosound();
    sio_putch(13,0);
  }
  if(x1>=27&&x1<=30&&buttons==1)
  {
    sio_putch(13,166);
    sound(2500);delay(100);nosound();
    sio_putch(13,0);
  }
  if(x1>=32&&x1<=35&&buttons==1)
  {
    sio_putch(13,70);
    sound(2500);delay(100);nosound();
    sio_putch(13,0);
  }
  if(x1>=37&&x1<=40&&buttons==1)
  {
    sio_putch(13,182);
    sound(2500);delay(100);nosound();
  }
}

```

```

    sio_putch(13,0);
}
if(x1>=43&&x1<=46&&buttons==1)
{
    sio_putch(13,47);
    sound(2500);delay(100);nosound();
    sio_putch(13,0);
}
if(x1>=48&&x1<=51&&buttons==1)
{
    sio_putch(13,63);
    sound(2500);delay(100);nosound();
    sio_putch(13,0);
}
if(x1>=53&&x1<=56&&buttons==1)
{
    sio_putch(13,31);
    sound(2500);delay(100);nosound();
    sio_putch(13,0);
}
if(x1>=59&&x1<=62&&buttons==1)
{
    sio_putch(13,20);
    sound(2500);delay(100);nosound();
    sio_putch(13,0);
}
if(x1>=64&&x1<=67&&buttons==1)
{
    sio_putch(13,228);
    sound(2500);delay(100);nosound();
    sio_putch(13,0);
}
if(x1>=69&&x1<=72&&buttons==1)
{
    sio_putch(13,4);
    sound(2500);delay(100);nosound();
    sio_putch(13,0);
}
if(x1>=74&&x1<=77&&buttons==1)
{
    sio_putch(13,244);
    sound(2500);delay(100);nosound();
    sio_putch(13,0);
}
}

if(y1>=29&&y1<=32)
{
    if(x1>=2&&x1<=5&&buttons==1)
    {
        sio_putch(13,124);
        sound(2500);delay(100);nosound();
        sio_putch(13,0);
    }
    if(x1>=7&&x1<=10&&buttons==1)
    {
        sio_putch(13,140);
        sound(2500);delay(100);nosound();
        sio_putch(13,0);
    }
    if(x1>=13&&x1<=15&&buttons==1)

```

```

{
  sio_putch(13,108);
  sound(2500);delay(100);nosound();
  sio_putch(13,0);
}
if(x1>=17&&x1<=20&&buttons==1)
{
  sio_putch(13,156);
  sound(2500);delay(100);nosound();
  sio_putch(13,0);
}
if(x1>=22&&x1<=25&&buttons==1)
{
  sio_putch(13,92);
  sound(2500);delay(100);nosound();
  sio_putch(13,0);
}
if(x1>=27&&x1<=30&&buttons==1)
{
  sio_putch(13,172);
  sound(2500);delay(100);nosound();
  sio_putch(13,0);
}
if(x1>=32&&x1<=35&&buttons==1)
{
  sio_putch(13,76);
  sound(2500);delay(100);nosound();
  sio_putch(13,0);
}
if(x1>=37&&x1<=40&&buttons==1)
{
  sio_putch(13,188);
  sound(2500);delay(100);nosound();
  sio_putch(13,0);
}
if(x1>=48&&x1<=51&&buttons==1)
{
  sio_putch(13,223);
  sound(2500);delay(100);nosound();
  sio_putch(13,0);
}
if(x1>=59&&x1<=62&&buttons==1)
{
  sio_putch(13,61);
  sound(2500);delay(100);nosound();
  sio_putch(13,0);
}
if(x1>=64&&x1<=67&&buttons==1)
{
  sio_putch(13,205);
  sound(2500);delay(100);nosound();
  sio_putch(13,0);
}
if(x1>=69&&x1<=72&&buttons==1)
{
  sio_putch(13,45);
  sound(2500);delay(100);nosound();
  sio_putch(13,0);
}
if(x1>=74&&x1<=77&&buttons==1)
{

```

```

    sio_putch(13,221);
    sound(2500);delay(100);nosound();
    sio_putch(13,0);
}
}
if(y1>=34&&y1<=37)
{
    if(x1>=2&&x1<=5&&buttons==1)
    {
        sio_putch(13,119);
        sound(2500);delay(100);nosound();
        sio_putch(13,0);
    }
    if(x1>=7&&x1<=10&&buttons==1)
    {
        sio_putch(13,135);
        sound(2500);delay(100);nosound();
        sio_putch(13,0);
    }
    if(x1>=13&&x1<=15&&buttons==1)
    {
        sio_putch(13,103);
        sound(2500);delay(100);nosound();
        sio_putch(13,0);
    }
    if(x1>=17&&x1<=20&&buttons==1)
    {
        sio_putch(13,151);
        sound(2500);delay(100);nosound();
        sio_putch(13,0);
    }
    if(x1>=22&&x1<=25&&buttons==1)
    {
        sio_putch(13,87);
        sound(2500);delay(100);nosound();
        sio_putch(13,0);
    }
    if(x1>=27&&x1<=30&&buttons==1)
    {
        sio_putch(13,167);
        sound(2500);delay(100);nosound();
        sio_putch(13,0);
    }
    if(x1>=32&&x1<=35&&buttons==1)
    {
        sio_putch(13,71);
        sound(2500);delay(100);nosound();
        sio_putch(13,0);
    }
    if(x1>=37&&x1<=40&&buttons==1)
    {
        sio_putch(13,183);
        sound(2500);delay(100);nosound();
        sio_putch(13,0);
    }
    if(x1>=43&&x1<=46&&buttons==1)
    {
        sio_putch(13,62);
        sound(2500);delay(100);nosound();
        sio_putch(13,0);
    }
}

```



```

}
if(x1>=48&&x1<=51&&buttons==1)
{
  sio_putch(13,206);
  sound(2500);delay(100);nosound();
  sio_putch(13,0);
}
if(x1>=53&&x1<=56&&buttons==1)
{
  sio_putch(13,46);
  sound(2500);delay(100);nosound();
  sio_putch(13,0);
}
if(x1>=59&&x1<=62&&buttons==1)
{
  sio_putch(13,29);
  sound(2500);delay(100);nosound();
  sio_putch(13,0);
  bar3d(15,15,576,143,3,1);
}
if(x1>=64&&x1<=67&&buttons==1)
{
  sio_putch(13,237);
  sound(2500);delay(100);nosound();
  sio_putch(13,0);
}
if(x1>=69&&x1<=72&&buttons==1)
{
  sio_putch(13,13);
  sound(2500);delay(100);nosound();
  sio_putch(13,0);
}
if(x1>=74&&x1<=77&&buttons==1)
{
  sio_putch(13,253);
  sound(2500);delay(100);nosound();
  sio_putch(13,0);
}
}

if(y1>=39&&y1<=42)
{
  if(x1>=2&&x1<=5&&buttons==1)
  {
    sio_putch(13,123);
    sound(2500);delay(100);nosound();
    sio_putch(13,0);
  }
  if(x1>=7&&x1<=10&&buttons==1)
  {
    sio_putch(13,139);
    sound(2500);delay(100);nosound();
    sio_putch(13,0);
  }
  if(x1>=13&&x1<=15&&buttons==1)
  {
    sio_putch(13,107);
    sound(2500);delay(100);nosound();
    sio_putch(13,0);
  }
  if(x1>=17&&x1<=20&&buttons==1)

```

```

    {
        sio_putch(13,155);
        sound(2500);delay(100);nosound();
        sio_putch(13,0);
    }
if(x1>=22&&x1<=25&&buttons==1)
{
    sio_putch(13,91);
    sound(2500);delay(100);nosound();
    sio_putch(13,0);
}
if(x1>=27&&x1<=30&&buttons==1)
{
    sio_putch(13,171);
    sound(2500);delay(100);nosound();
    sio_putch(13,0);
}
if(x1>=32&&x1<=35&&buttons==1)
{
    sio_putch(13,75);
    sound(2500);delay(100);nosound();
    sio_putch(13,0);
}
if(x1>=37&&x1<=40&&buttons==1)
{
    sio_putch(13,187);
    sound(2500);delay(100);nosound();
    sio_putch(13,0);
}
if(x1>=43&&x1<=46&&buttons==1)
{
    sio_putch(13,222);
    sound(2500);delay(100);nosound();
    sio_putch(13,0);
}
if(x1>=48&&x1<=51&&buttons==1)
{
    sio_putch(13,30);
    sound(2500);delay(100);nosound();
    sio_putch(13,0);
}
if(x1>=53&&x1<=56&&buttons==1)
{
    sio_putch(13,238);
    sound(2500);delay(100);nosound();
    sio_putch(13,0);
}
}

if(y1>=44&&y1<=47)
{
    if(x1>=2&&x1<=5&&buttons==1)
    {
        sio_putch(13,120);
        sound(2500);delay(100);nosound();
        sio_putch(13,0);
    }
    if(x1>=7&&x1<=10&&buttons==1)
    {
        sio_putch(13,136);
        sound(2500);delay(100);nosound();
    }
}

```

```

    sio_putch(13,0);
}
if(x1>=13&&x1<=15&&buttons==1)
{
    sio_putch(13,104);
    sound(2500);delay(100);nosound();
    sio_putch(13,0);
}
if(x1>=17&&x1<=20&&buttons==1)
{
    sio_putch(13,152);
    sound(2500);delay(100);nosound();
    sio_putch(13,0);
}
if(x1>=22&&x1<=25&&buttons==1)
{
    sio_putch(13,88);
    sound(2500);delay(100);nosound();
    sio_putch(13,0);
}
if(x1>=27&&x1<=30&&buttons==1)
{
    sio_putch(13,168);
    sound(2500);delay(100);nosound();
    sio_putch(13,0);
}
if(x1>=32&&x1<=35&&buttons==1)
{
    sio_putch(13,72);
    sound(2500);delay(100);nosound();
    sio_putch(13,0);
}
if(x1>=37&&x1<=40&&buttons==1)
{
    sio_putch(13,184);
    sound(2500);delay(100);nosound();
    sio_putch(13,0);
}
if(x1>=59&&x1<=67&&buttons==1)
{
    sio_putch(13,53);
    sound(2500);delay(100);nosound();
    sio_putch(13,0);
}
if(x1>=69&&x1<=77&&buttons==1)
{
    sio_putch(13,213);
    sound(2500);delay(100);nosound();
    sio_putch(13,0);
}
}

if(y1>=49&&y1<=52)
{
    if(x1>=2&&x1<=5&&buttons==1)
    {
        sio_putch(13,122);
        sound(2500);delay(100);nosound();
        sio_putch(13,0);
    }
    if(x1>=7&&x1<=10&&buttons==1)

```

```

{
  sio_putch(13,138);
  sound(2500);delay(100);nosound();
  sio_putch(13,0);
}
if(x1>=13&&x1<=15&&buttons==1)
{
  sio_putch(13,106);
  sound(2500);delay(100);nosound();
  sio_putch(13,0);
}
if(x1>=17&&x1<=20&&buttons==1)
{
  sio_putch(13,154);
  sound(2500);delay(100);nosound();
  sio_putch(13,0);
}
if(x1>=22&&x1<=25&&buttons==1)
{
  sio_putch(13,90);
  sound(2500);delay(100);nosound();
  sio_putch(13,0);
}
if(x1>=27&&x1<=30&&buttons==1)
{
  sio_putch(13,170);
  sound(2500);delay(100);nosound();
  sio_putch(13,0);
}
if(x1>=32&&x1<=35&&buttons==1)
{
  sio_putch(13,74);
  sound(2500);delay(100);nosound();
  sio_putch(13,0);
}
if(x1>=37&&x1<=40&&buttons==1)
{
  sio_putch(13,186);
  sound(2500);delay(100);nosound();
  sio_putch(13,0);
}
if(x1>=43&&x1<=46&&buttons==1)
{
  sio_putch(13,51);
  sound(2500);delay(100);nosound();
  sio_putch(13,0);
}
if(x1>=48&&x1<=51&&buttons==1)
{
  sio_putch(13,195);
  sound(2500);delay(100);nosound();
  sio_putch(13,0);
}
if(x1>=53&&x1<=56&&buttons==1)
{
  sio_putch(13,35);
  sound(2500);delay(100);nosound();
  sio_putch(13,0);
}
if(x1>=59&&x1<=67&&buttons==1)
{

```

```

    sio_putch(13,197);
    sound(2500);delay(100);nosound();
    sio_putch(13,0);
}
if(x1>=69&&x1<=77&&buttons==1)
{
    sio_putch(13,21);
    sound(2500);delay(100);nosound();
    sio_putch(13,0);
}
}

if(y1>=54&&y1<=57)
{
    if(x1>=2&&x1<=5&&buttons==1)
    {
        sio_putch(13,121);
        sound(2500);delay(100);nosound();
        sio_putch(13,0);
    }
    if(x1>=7&&x1<=10&&buttons==1)
    {
        sio_putch(13,137);
        sound(2500);delay(100);nosound();
        sio_putch(13,0);
    }
    if(x1>=13&&x1<=15&&buttons==1)
    {
        sio_putch(13,105);
        sound(2500);delay(100);nosound();
        sio_putch(13,0);
    }
    if(x1>=17&&x1<=20&&buttons==1)
    {
        sio_putch(13,153);
        sound(2500);delay(100);nosound();
        sio_putch(13,0);
    }
    if(x1>=22&&x1<=25&&buttons==1)
    {
        sio_putch(13,89);
        sound(2500);delay(100);nosound();
        sio_putch(13,0);
    }
    if(x1>=27&&x1<=30&&buttons==1)
    {
        sio_putch(13,169);
        sound(2500);delay(100);nosound();
        sio_putch(13,0);
    }
    if(x1>=32&&x1<=40&&buttons==1)
    {
        sio_putch(13,73);
        sound(2500);delay(100);nosound();
        sio_putch(13,0);
    }
    if(x1>=48&&x1<=51&&buttons==1)
    {
        sio_putch(13,211);
        sound(2500);delay(100);nosound();
        sio_putch(13,0);
    }
}

```

```

    }
    if(x1>=59&&x1<=67&&buttons==1)
    {
        sio_putch(13,37);
        sound(2500);delay(100);nosound();
        sio_putch(13,0);
    }
    if(x1>=69&&x1<=77&&buttons==1)
    {
        sio_putch(13,229);
        sound(2500);delay(100);nosound();
        sio_putch(13,0);
    }
}
if(buttons==2)
{
    cleardevice();
}
}while(buttons!=3);
hide_mouse();
menuu();
}
void show_mouse()
{
    struct REGPACK reg;
    reg.r_ax=1;
    intr(0x33,&reg);
}
void init_mouse()
{
    struct REGPACK reg;
    reg.r_ax=0;
    intr(0x33,&reg);
}
void get_mouse()
{
    struct REGPACK reg;
    reg.r_ax=3;
    intr(0x33,&reg);
    x1=reg.r_cx/8;
    y1=reg.r_dx/8;
    buttons=reg.r_bx;
}

void hide_mouse()
{
    struct REGPACK reg;
    reg.r_ax=2;
    intr(0x33,&reg);
}
void main11()
{
    int i,j,port,stat;
    int card_no,total_port,data_item;
    char command[20];

    restorecrtmode();
    clrscr();
    printf("RESET THE MOXA CARDS.....\n");
    if ((card_no = sio_reset()) == 0)
    {

```

```

    printf("NO CARD FOUND !\n");
    exit(0);
}
printf("TOTAL CARD : %d\n",card_no);

printf("READ THE ID.....\n");
for (i=1; i<=card_no; i++)
{
    printf("(%d)\tSerial no: %d",i,sio_id(i));
    printf("\tMapping : %X\n",sio_bank(i));
}
printf("Setting port 13 to 9600, n, 8, 1.....\n");
port = 13;
if ( sio_ioctl(port,B9600,BIT_8 | P_NONE | STOP_1) != 0)
{
    printf("Port #%d IOCTL error !\n",port);
    exit(0);
}
printf("Open port 13.....\n");
port = 13;
stat = sio_open(port);
if(stat != 0)
{
    printf("Port 13 can't be opened !\n");
    exit(0);
}

printf("\n\n\nPRESS ANY KEY TO CONTINUE !\n");
getch();
clrscr();
main22();
}

void main22()
{
    char command[20];
    char ch;
    int i,c;

    printf("NOW LET'S START !\n");
    printf("1, SWITCH THE CONTROL UNIT POWER ON, WAITING 15 SECONDS\n");
    printf("2, TURN THE POSITION OF SELECTION SWITCH TO LOW SPEED\n");
    printf("3, INTUT 'SERVO ON' TO START ROBOT\n");
    sio_putch(13,0);
    do
    {
        gets(command);
        i=strcmp(command,"SERVO ON");
        if(i!= 0)
            printf("Input fault,try it again.....\n");
    }while(i!=0);
    sio_putch(13,21);
    sound(2500);delay(100);nosound();
    sio_putch(13,0);
    printf("Press 'ZERO ADJ' on Teaching Box to get zero adjustment\n");
    printf("Waiting for ready\n");
    printf("\n\n\nIf Emergency Stop occurs, press and hold ABNOR REL on T.B\n");
    printf("Input 'R' or 'r'to recover, then adjust axes position by T.B,\n");
}

```

```

printf("or input 'Y' or 'y' to be continue at normal
circumstances.\n");
do
{
ch=getch();
switch(ch)
{
case 'r':
case 'R':
sio_putch(13,213);
sound(2500);delay(100);nosound();
sio_putch(13,0);
for(n=1;n<=20000;n++){
sio_putch(13,21);
sound(2500);delay(100);nosound();
sio_putch(13,0);
printf("Try zero adjustment again\n");
break;
case 'y':
case 'Y':
break;
default: break;
}
}while(ch!='y' && ch!='Y');
printf("Input SERVO OFF to stop preparation\n");
do
{
gets(command);
i = strcmp(command,"servo off");
if(i!=0)
printf("Input fault, try it again\n");
}while(i!=0);
sio_putch(13,229);
sound(2500);delay(100);nosound();
sio_putch(13,0);
printf("Turn the key position to MOTOR POWER OFF\n");
printf("Press any button to continue\n");
getch();
clrscr();
printf("Selection of Working Mode\n");
printf("\n\n1.  EDIT\n");
printf("\n2.  JOB TEACH\n");
printf("\n3.  PROGRAM TEACH\n");
printf("\n4.  PLAYBACK\n");
printf("\n5.  EXIT\n");
printf("\n\n[ Enter Your Choice ]\n");
do
{
c=getch();
switch(c)
{
case '1': edit();
break;
case '2': jobteach();
break;
case '3': progteach();
break;
case '4': playback();
break;
case '5': clrscr();
exit(0);
}
}
}

```



```

        break;
    default:
        break;
    }
}while(c!='1' && c!='2' && c!='3' && c!='4'&&c!='5');
}
void edit()
{
    main();
}
void jobteach()
{
    char ch,str[20];
    int n,i,x1;

    clrscr();
    printf("Press and hold button 'A' on T.B,\n");
    printf("then input 'J' or 'j' to enter Teach Mode....\n");
    do
    {
        ch=getch();
        switch(ch)
        {
            case 'J':
            case 'j':
                sio_putch(13,29);
                sound(2500);delay(100);nosound();
                sio_putch(13,0);
                break;
            default:
                printf("Input fault, try it again\n");
                break;
        }
    }while(ch!='J' && ch!='j');
    printf("\n");
    for(;;)
    {
        printf("Input command number.....\n");

        gets(str);
        if(strcmp(str,"esc")==0||strcmp(str,"ESC")==0)
        {
            main();
        }
        i=atoi(str);

        sio_putch(13,i);
        sound(2500);delay(100);nosound();
        sio_putch(13,0);
        printf("Input string is %d\n",i);
    }
}

}
void progteach()
{
    main();
}
void playback()
{
    main();
}

```