THE DEVELOPMENT OF A FLEXIBLE CHARACTERISATION SYSTEM FOR SURFACE METROLOGY

Xiangqi Lan

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

School of Computing and Engineering
University of Huddersfield

April 2014

Abstract

Surface texture and its measurement are becoming more and more increasingly important in the field of high precision engineering and nano-technology. It is a significant and efficient way to predict the functional performance of an engineering product by characterising its surface. As the extraction and evaluation of surfaces features not only relies on analysis algorithms but also measurement techniques, most of surface characterisation software systems are embedded in the measurement instruments. At present, even though a series of Geometrical Product Specification (GPS) standards are released to guide the procedure of surface characterisation, there are no software systems give a fully support of them. As a consequence, evaluation results from different systems are incomparable, even worse, conflict with each other.

Surface characterisation system needs to be updated constantly with the emergence of new algorithms and methods. However, the lack of good extensibility, reusability and maintainability is a serious obstacle to the innovation of existing surface characterisation systems. As functional modules in current surface characterisation systems are tightly coupled together, it is not conducive to the reuse of function modules and innovation of overall system. A lot of redundant and duplicate works have been raised in either enhancing present characterisation systems or building a new characterisation system.

To improve the reusability of function modules and facilitate system extension, this research aims to establish a flexible surface characterisation system with an open architecture. By employing component based development technologies, the overall characterisation system is constructed by gluing various functional components together instead of being created from scratch. Each analysis algorithm or method is implemented as an independent functional component, which is separated from the system framework. And also it can be easily reused by other characterisation systems as an executable chunk. Any system functional components can be developed or maintained independently by different organisation as long as they comply with predefined protocols (interfaces).

This thesis proposed a novel surface characterisation system, which can be reconfigured as end users' expectation at any time even when it has been installed and deployed already. Functional components can be added, removed or replaced dynamically without affecting other parts of the system. Furthermore, the system is flexible such that researchers and developers can concentrate on characterisation algorithms and methods themselves, and then develop their own functional components which can be easily added to this system.

Acknowledgements

First of all, I would like to express my sincere gratitude to my supervisor, Prof. Xiangqian Jiang for setting up an appropriate research subject. She offers her unreserved help and guidance, and leads me to finish my thesis. Her words can always inspire me and bring me to a higher level of thinking. Many thanks also give to her for kindly support, advice and encouragement.

Great appreciation also goes to my second supervisor, Prof. Liam Blunt, for his exceptional support and guidance throughout my thesis. Without his kind and patient instruction, it is impossible for me to finish this thesis. He gave lots of time helping me to improve my academic writing and the capability of carrying out professional researches.

I am also indebted to Prof. Paul Scott, Dr. Shaojun Xiao and Dr. Feng Xie for their valuable guidance and suggestions. Special thanks to Dr. Wenhan Zeng and Dr. Haydn Martin. They squeeze time from their busy schedule to help me correct my thesis.

I would like to show my appreciation to Taylor Hobson Ltd. for the support and sponsorship of my studies, and I am also grateful to the EPSRC Centre for Innovative Manufacturing in Advanced Metrology, the Centre for Precision (CPT) and the School of Computing and Engineering of the University of Huddersfield for providing me the opportunity and facilities to carry out this project.

Sincere thanks to my colleagues for sharing their insights, many useful discussions and valuable suggestions, and my friends for their kindly support and encouragements.

Finally, I would like to thank my family for supporting me throughout all my studies. Their love provided my inspiration and was my driving force. I owe them everything and wish I could show them just how much I love and appreciate them.

Contents

Abstract 1

Acknowledgements 2

List of Figures 7

List of Tables 9

List of Acronyms 10

Chapter 1 Introduction 11

1.1 Background 11

1.2 Aim and Objectives 16

1.3 Approach 19

1.4 Thesis Structure 21

Chapter 2 Surface Characterisation Techniques 24

2.1 Introduction 24

2.2 Surface Verification Operator 24

2.3 Surface Measurement 32

2.3.1 Stylus Methods 32

2.3.2 Optical Methods 35

2.3.3 Other Methods 36

2.3.4 Range and Resolution 37

2.4 Fitting Techniques 38

2.4.1 Least Squares fitting 39

2.4.2 Profile fitting 40

2.4.3 Areal fitting 42

2.5 Filtering Techniques 44

2.5.1 The Process of Filtering 45

2.5.2 Filtering Methods 47

2.6 Parameterisation 49

2.6.1 Statistical Methods 50

2.6.2 Geometrical Methods 52

2.7 Summary 53

Chapter 3 Design and Implementation of Flexible System Architecture 54

3.1 Introduction 54

3.2 Component-Based Development 56
3.2.1 Component 56
3.2.2 Component-Based Software Development 57
3.2.3 Current Technology for Component-Based Architecture 58
3.2.4 Advantages of Component-Based Development 60
3.3 System Architecture Design 60
3.3.1 Separation of Analysis Functions and System Framework 60
3.3.2 Integrative Structure 61
3.3.3 Semi-detached Structure 62
3.3.4 Flexible Structure 63
3.3.5 Proposed Surface Characterisation System Architecture 64
3.4 Surfstand System Functional Component Categorisation 69
3.4.1 Data Access Component 69
3.4.2 Data Processing Component 70
3.4.3 Data Display Component 71
3.5 System Development Life Cycle 73
3.5.1 Waterfall Model 73
3.5.2 Incremental Model 75
3.5.3 Spiral Model 76
3.5.4 Development Model of SurfStand 78
3.6 Summary 81
Chapter 4 Implementation of Data Access Components 82
4.1 Conventional Surface Data File Format 82
4.2 Component Design 85
4.2.1 Surface Data Structure Design for the Proposed Surface Characterisation System 85
4.2.2 Surface Data Class Design for the Proposed Surface Characterisation System 89
4.2.3 Interface Design for the Data Access Components 90
4.3 Standard Surface Data Format: SDF 91
4.3.1 SDF File Format 92
4.3.2 Interface Implementation 96
4.4 Summary 97

Chapter 5 Design and Implementation of Data Process Components of the Developed System 98
5.1 Categorisation of Data Process Components 98
5.2 Component Interface Design of the Proposed System 103
5.2.1 Interface of Verification Operations 103
5.2.2 Interface for All Data Process Components 106
5.2.3 Interface for Subtypes of Data Process Components 108
5.3 Case Studies: Geometrical Analysis Component 109
5.3.1 Surface Segmentation 110
5.3.2 Boundary curve approximation 111
5.3.3 Geometrical features construction 114
5.3.4 Interface Implementation for Geometrical Analysis Component 116
5.4 Summary 118
Chapter 6 Implementation of Surface Visualisation Components 119
6.1 Computer Graphics 119
6.2 Computer Graphics Programming—OpenGL 121
6.2.1 Modelling 121
6.2.2 Transformation 122
6.2.3 Colour 125
6.2.4 Lighting 125
6.2.5 Other Functions of OpenGL 126
6.3 Component Interface Design 126
6.4 Three-Dimensional Display Component of the developed Surface Characterisation System 127
6.4.1 Coordinate System 127
6.4.2 Modelling 128
6.4.3 Render 129
6.4.4 Lighting and Object Materials 130
6.4.5 Interface Implementation 131
6.5 Summary 132
Chapter 7 Test and Evaluation of the Proposed Surface Characterisation System ... 133
7.1 Introduction 133
7.2 Unit Tests 133

7.2.1 Test Case 1: SDF File Component 134
7.2.2 Test Case 2: 3D display component 135
7.3 Integration Tests 138
7.3.1 Test Case 1: SUR File Component 139
7.3.2 Test Case 2: Levelling Component 141
7.3.3 Test case 3: Topview component 142
7.4 System Tests 143
7.4.1 Test Case 1 Components Configuration 143
7.4.2 Test Case 2: Compound Command Test 144
7.5 Comparison 147
7.6 Summary 148
Chapter 8 Conclusions and Future Work 150
8.1 Summary 150
8.2 Contribution to Knowledge 152
8.2.1 Standardisation Analysis 152
8.2.2 Function Modularisation 153
8.2.3 System Flexibility 154
8.3 Future Work 155
References 157
Appendixes 163
A Critical Code Fragments of SurfStand SDK 163
B Code Fragments for the Connection between the Framework and Components 166
C Code Fragments for Surface Data Definition 168
D Definition of Command Interface: ISSCommand. 170
E Code Fragments for Method ModifyArgument of Feature Parameters Component ... 171
F Interface Definitions for the Three Subtypes of Data Process Components 172
G Related Publications 173

List of Figures

Figure 1.1: Thesis structure diagram. 22
Figure 2.1: Comparison of Design Intent and Verification of manufactured workpiece. 25
Figure 2.2: Standard surface verification operator [2]. 28
Figure 2.3: F-operator—first order polynomial fitting. 29
Figure 2.4: L-filter—Gaussian regression filtering. 30
Figure 2.5: Calculation results of standard surface areal parameters. 31
Figure 2.6: The stylus principle. 33
Figure 2.7: Two types of stylus—Cone and Pyramid [51]. 34
Figure 2.8: Stylus slope effect. 34
Figure 2.9: Path of light in Michelson interferometer [56]. 36
Figure 2.10: Amplitude-wavelength plot of different instruments for surface measurements [61]. 38
Figure 2.11: Roughness, waviness and form of a profile [51]. 45
Figure 2.12: Filtering process in two domains—Space and Frequency [51]. 45
Figure 2.13: Cutoffs of surface filtering [51]. 46
Figure 3.1: A component with provided and required interfaces. 57
Figure 3.2: Integrative structure of software systems. 61
Figure 3.3: Semi-detached structure of software systems. 62
Figure 3.4: Flexible structure of software systems. 63
Figure 3.5: Data flow within a surface characterisation system. 65
Figure 3.6: Development hierarchy chart for SurfStand. 67
Figure 3.7: System architecture of SurfStand. 68
Figure 3.8: Function chart of Data Access Components. 70
Figure 3.9: Function chat of Data Process Components. 71
Figure 3.10: Data Display Components—3D Display Component. 72
Figure 3.11: Software development model—Waterfall model. 74
Figure 3.12: Software development model—Increment model. 76
Figure 3.13: Software development model—Spiral model. 76
Figure 4.1: Surface data pattern—Grid Data. 82
Figure 4.2: Data transfer between surface data file and internal data structure. 87
Figure 5.1: Example of Data Process Components—Type A. 100
Figure 5.2: Example of Data Process Components—Type B. 101
Figure 5.3: Example of Data Process Components—Type C. 102
Figure 5.4: Internal command class inheritance hierarchy diagram of developed system. 105
Figure 5.5: Relationship between commands and surface verification operators. 106
Figure 5.6:Relationship between commands and system functional components. 107
Figure 5.7: Class inheritance hierarchy diagram of data process components. 108
Figure 5.8 Demonstration of surface segmentation 111
Figure 5.9: One boundary curve of a geometrical feature. 112
Figure 5.10: Curvature of each point on the boundary curve. 113
Figure 5.11: Derivative of curvature along the boundary curve. 113
Figure 5.12: Extracted joint points of the boundary curve. 114

Figure 5.13: Characteristics list of geometrical features. 115
Figure 6.1: Computer graphics system principle. 119
Figure 6.2: Comparison of drawing a pentagon and five points. 122
Figure 6.3: Perspective viewing volume. 123
Figure 6.4: Orthographic viewing volume. 124
Figure 6.5: The geometry pipeline that achieves the coordinate transformation. 127
Figure 6.6: Default 3D model coordinate system of OpenGL. 128
Figure 6.7: 3D modelling of surface data. 129
Figure 6.8: Supported colour maps for surface render. 130
Figure 6.9: An example of materials rending—Brass. 131
Figure 7.1: A SDF file opened with the SDF file component. 134
Figure 7.2 The surface data that is saved with the SDF file component. 135
Figure 7.3: 3D display component loaded with ActiveX Control Test Container. 136
Figure 7.4: Surface data display with 3D mesh representation type. 136
Figure 7.5: Surface data rendered with hot colour map. 137
Figure 7.6: Surface data display with yellow rubber material. 138
Figure 7.7: Components Configuration dialog. 139
Figure 7.8: Open file dialog with supporting SDF and SUR file formats. 139
Figure 7.9: An example of surface data imported with SUR file component. 140
Figure 7.10: Open file dialog without supporting SUR file format. 140
Figure 7.11: The emerging levelling menu item after adding the levelling component. 141
Figure 7.12: The surface data that has been processed with levelling component. 142
Figure 7.13: The surface data that is displayed with Topview Component. 142
Figure 7.14:The surface data that is processed with Bearing curve analysis component. 144
Figure 7.15: Compound command creating dialog. 145
Figure 7.16: Subcommands modification of Test command. 145
Figure 7.17: Test command occurs in the Command setting dialog. 146
Figure 7.18: Parameters setting dialog of Gaussian regression filter. 146
Figure 7.19: The surface data that is processed with Test command. 147
Figure 8.1: Categorisation of system functional components. 151
Figure 8.2: Completed system functional components. 152

List of Tables

Table 2.1: Common model functions for profile fitting. 40
Table 2.2: Standard surface profile parameters. 50
Table 2.3: Standard surface areal parameters. 51
Table 2.4: Feature parameters defined in ISO 25178-2. 52
Table 4.1: Common file formats for surface data storage. 83
Table 4.2: Essential elements for every surface data file. 84
Table 4.3 Internal surface data structure elements. 86
Table 4.4: Arguments description of methods of interface SSFileIO. 91
Table 4.5: File header description of SDF surface file. 92
Table 4.6: Compression types list. 94
Table 4.7: Supported data types list of SDF file format. 95
Table 4.8: Checksum types list. 95
Table 5.1: Description of member variables and methods of interface ISSCommand. 104
Table 5.2: Description of arguments that occurs in SSAnalysisA, SSAnalysisB and SSAnalysisC. 109
Table 5.3: Combination patterns for common geometrical features. 115
Table 6.1: Property values for the brass material. 131
Table 7.1: Comparison between Surfstand and other systems. 148

List of Acronyms

AFM	Atomic Force Microscopy
API	Application Programming Interface
ARB	OpenGL Architecture Review Board
ASCII	American Standard Code for Information Interchange
ATL	Active Template Library
AXCTC	ActiveX Control Test Container
CBA	Component-Based Architecture
CBD	Component-Based Development
CCD	Charge Couple Device
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
GPU	Graphics Processing Unit
IDL	Interface Definition Language
ISO	International Organization for Standardization
JRMP	Java Remote Method Protocol
JVM	Java Virtual Machine
LS	Least Squares
MC	Minimum Circum scribed
MI	Maximum Inscribed
MZ	Minimum Zone
OMG	Object Management Group
ORB	Object Request Broker
RC	Resistor-Capacitor
RMI	Remote Method Invocation
SDK	Software Development Kit
SDLC	System Development Life Cycle
SPM	Scanning Probe Microscopy
STM	Scanning Tunneling Microscopy
UML	Unified Modelling Language

Chapter 1 Introduction

1.1 Background

The surface of an engineering workpiece can be thought of as the boundary between the workpiece and the surrounding environment, and it is closely related to the functional properties of the entire workpiece, such as tribological, thermal, electrical and optical behaviour [1]. It has been revealed that 90% of all engineering workpiece failures in practice are surface initiated due to corrosion, erosion, fretting wear, excessive abrasive and so on [2]. Thus surface topographical features are important factors in determining the satisfactory performance of a workpiece, and it is significant to understand the functional properties of surfaces.

Surface metrology is the science of measuring geometrical features on surfaces. The approach is to measure and characterise surface features in order to be able to understand how the features are influenced by its history (e.g. manufacture, wear, fracture) and how it influences its behaviour (e.g. adhesion, gloss, friction) [3]. From the occurrence which can be traced back to the 1930s up to now, surface metrology has undergone a series of huge paradigm shifts [4, 5]. Nowadays, it has become one of the fastest-growing areas of engineering and quality management, and its study is becoming more and more commonplace in industrial and research environments [6].

To understand and analyse surface features, surface measurement is a critical and necessary procedure to acquire surface information from engineering workpieces. The original way to measure surfaces was using the thumbnail and the eye, both of which were subjective and effective only if used by an experienced practitioner. However, the texture features of surfaces are too small to be assessed quantitatively by such a “touch and see” way [7]. Demand for quantitative results led to two parallel branches of instrumentation; one following the tactile example of the nail, the other mimicking the eye. Stylus profilometry and optical interferometry are two common techniques used in carrying out surface metrology. In addition, there are also many other methods for surface measurement such as using pneumatic technique, capacitance, ultrasound, and so on [7].

Normally, the aim of surface measurement methods is to acquire the surface information effectively and accurately. There is no one method that can be adopted to measure all kinds of surfaces. For a certain surface, the measurement method is decided by many factors such as precision, scale, material, etc. However, measurement results consist of a set of coordinates, which cannot be adopted to characterise the surface directly. A series of data processing based on the measurement data is required to extract surface features. With the benefit from modern digital technologies, e.g. Computer, which offers adequate speed of operations and processing, the extraction and characterisation could be realised by software systems. Nowadays, surface characterisation systems are mainly developed by instrument companies, and they are usually deployed in surface measurement instruments. For example, Talymap is such a software system which is primary embedded in surface measurement instruments manufactured by Taylor Hobson [8]. With these convenient and efficient systems, metrologists are able to carry out surface characterisation and evaluation after measurements. Besides, some institutes also develop their own surface characterisation system [8-10]. Although there are many surface characterisation systems already, it is still necessary to implement flexible system architecture. The reason will be elaborated from two different perspectives below.

From the point view of surface metrology, it is expected to get the same characterisation and evaluation result for a particular measurement data. Conversely, there are always large variations in the output results among different characterisation systems, which are caused by a couple of reasons, such as ambiguous definitions, choices of analysis methods and various implementations. ISO (International Organization for Standardization) issued a series of characterisation standards which aim to standardise the characterisation methods. For example, ISO 4287 specifies terms, definitions and parameters for the determination of surface texture by profiling methods [11], while ISO 25178-2 specifies them by areal methods [12]. However, due to the mathematical ambiguity of some definitions presented in these ISO standards, there is scope for different interpretations of how to calculate the parameters. Consequently, software engineers may design mathematically diverse algorithms to implement the definitions [13]. That explains why it is hard to get the same result from different characterisation systems even if they realise the algorithm strictly in accordance with ISO standards. Thus,

the characterisation results as the communication information of a surface are incomparable and untraceable, and it would be meaningless to compare analysis results, which are derived from different characterisation systems.

In addition, surface characterisation and verification results are figured out by several sequential steps of processing. According to GPS (Geometrical Product Specification) standards, the whole procedure is recognised as an operator, while each step of processing is thought of as one operation [2]. In other words, a surface verification operator is usually composed of a set of sequential operations. Verification results are closely related to the verification operator. At present, the realisation of surface characterisation systems primary relies on operations instead of operators. Operators cannot be provided by existing characterisation systems, and they are generated by metrologists during the course of surface characterisation and verification. When evaluating a particular surface, various verification operators are likely to be used for the same surface by different metrologists, moreover, it is inevitable even though the metrologists are the same person, therefore, evaluation results will be greatly affected by the subjective determination. To reduce the influence caused by human factors, it is essential to provide verification operators for a surface characterisation system.

From the perspective of software engineering, as aforementioned, there are a number of characterisation systems developed by various companies or research institutes [8-10]. It is well known that most of the functions of these characterisation systems are similar, which should be implemented according to ISO standards. As a result, a great deal of redundant and reduplicated work has been done, and each characterisation system has to provide such a functional implementation. These characterisation systems suffer from a couple of insufficient capabilities, which are elaborated from following aspects:

- Reusability: Some basic and common functional modules are essential parts for every surface characterisation system. Since they are developed to satisfy a specific characterisation system, they cannot be reused by other systems. For one thing, the development environment of a functional module, such as programming language, tools and platform, is selected in accordance with the whole characterisation system, which will encapsulate this functional module. For

another, functional modules are normally tightly coupled with the whole characterisation system. They are not independent components and cannot execute separately. When developing a new characterisation system, the developer has to re-implement the similar and basic functional modules from scratch instead of reusing existing ones, which will cost much time and effort [14].

- Extensibility: Surface characterisation system is apt to be updated with the rapid innovation of surface metrology techniques; evolving from profile to areal characterisation, from stochastic to structured surface analysis and from simple geometries to complex free form geometries. With the development of processing and manufacturing technologies, novel characterisation techniques are expected to satisfy new emerging requirements[15, 16]. Hence, not only the conventional characterisation methods but also some state-of-the-art analysis algorithms and techniques will be taken into account. However, it will cost too much to embed those emerging techniques to the existing characterisation systems as developers are supposed to have good knowledge of the characterisation system architecture. Even worse, after the extension times and times again, the characterisation system tends to become unstable and error-prone.

- Maintainability: Maintenance is always a significant activity during the whole life-cycle of modern software systems. It is inevitable to maintain surface characterisation systems to keep their functionality in line with requirements and ensure their availability [17]. For existing characterisation systems, code modification is quite common during the procedure of maintenance as they are tightly coupled. However, code modification can only be done assuming the original code has been understood by maintainers. They also need to have intimate knowledge of the system architecture, and that would cost too much time and effort for them who have not participated in the design and development of the system. Because of the different coding style, most software engineers prefer writing the code themselves rather than reading others'. With the incessant modifications, surface characterisation systems will become increasingly complicated and hard to maintain. Another momentous drawback with respect to

maintenance is that any modifications may cause the recompiling and rebuilding of the whole system, and then redeployment on target computers, even if it is just a slight modification [18].

- Customisation: Generally, functions of surface characterisation systems are determined by developers, which cannot be changed by end users. As they do not provide any "space" for users to reconfigure its functions, what users can do is just to utilise existing analysis functional modules [19]. Sometimes, not all of the functional modules provided by a characterisation system may be really useful for some users, and they might want to use their own algorithm or idea to analyse surfaces. However, it is impossible to embed users' algorithms to current characterisation systems, and they have to develop a simplified version to practise their algorithms. In other words, users will have to implement many other functions which are not relevant to the algorithms themselves, such as file access and data visualisation. That could be notoriously difficult and may lead to plenty of irrelevant work for users.

As stated above, drawbacks of current surface characterisation systems have significant effects not only on the realisation of standard surface characterisation process, but also the renovation of themselves. Due to the fact they are implemented by various companies and research institutes individually, the evaluation results from different characterisation systems are not exactly the same, and what is worse is that some of them may be contradicted with the others. According to the intention of GPS standards, the evaluation result should be consistent no matter which characterisation system is adopted. Hence, reducing ambiguous of GPS standard documents and affording relevant functional modules would conduce to acquire comparable and traceable evaluation results. Meanwhile, the low capability of functional module reuse, system extension, maintenance, and user customisation is also a barrier of system evolution and characterisation technology progress. To overcome aforementioned issues, it is essential to develop a novel and flexible characterisation system for surface metrology. This project sets out to realise such a surface characterisation system with the state-of-the-art techniques of software engineering, and establish an open flexible architecture with the absolute separation between functional modules and system framework [20].

In the realm of surface metrology, the crucial thing is how to acquire surface features and evaluate surfaces by qualitative or quantitative analysis of these features. Currently, some researchers have to be involved in the development of characterisation systems, and they need a deep knowledge of the system architecture. Even worse, they will have to spend substantial time to evolution and maintenance of the characterisation system [14], which is not the primary work for a researcher of surface metrology. However, with the flexible characterisation system, researchers can throw themselves into the research of characterisation technologies rather than the implementation and maintenance of the characterisation software system. They just realise their own analysis algorithms or methods and build them as executable functional modules which can be embedded into the flexible characterisation system.

1.2 Aim and Objectives

This thesis aims to establish a flexible software system for surface evaluation based on the investigation and improvement on present surface characterisation techniques. The most essential difference from current characterisation systems is that the designed system will be constructed by various independent functional components instead of being created as a standalone chunk. Namely, the entire system will be divided into one system framework and a number of functional components: The system framework mainly manages the interaction between the overall system and end user, event driven and components management; Functional components, which are loosely coupled with system framework, will be designed as an independent and executable block. Both the system framework and functional components will be developed independently and then can be seamless connected together at run-time.

It is envisaged that this project will contribute to the provision of a common characterisation system for the researchers of surface metrology. Each user of the user group can develop its own functional modules or outsource certain functional modules to third party. The platform can provide a convenient method to share novel algorithms or techniques and comparisons of analysis results will be realised easily. In order to achieve this purpose, the research objectives of this project are classified as follows:

1) **Investigate the standard surface characterisation techniques.** Surface parameters are widely accepted for evaluation of surface topography, and most common parameters for profile and areal have been defined in ISO 4287 and ISO 25178-2 respectively. Generally, there a series of algorithmic operations needs to be completed prior to calculating parameters, where the whole procedure for evaluating surfaces can be recognised as a verification operator. To apply this standard method to evaluate surfaces, it is therefore essential to develop and understand the meaning of each parameter and the effect of each operation on the resulting parametric outcome.

2) **Design an open architecture for the software system.** Open architecture means that the specifications of the architecture are public, and all the parts of the architecture are loosely coupled. Specific functional analysis modules can be added, removed or replace dynamically without affecting other parts of the software system. Based on this open architecture, the ability to extend and maintain the system no longer belongs to the system creators. Other developers can also participate and share in the system innovation.

3) **Categorise the analysis function modules.** One functional module is an implementation of certain operation, and a series of sequenced operations are composed to be an operator. In a characterisation system, there are many functional modules. Usually, various functional modules need to be organised and invoked in a different way. In order to reconfigure them dynamically, it is necessary that they are discernible for the system framework. Thus, they ought to be categorised by distinguishing their inherent attributes such as input and output.

4) **Formulate the interaction protocols between functional modules and the system framework.** As all analysis functional modules will be physically separated from the system framework, in this way the system architecture will not become increasingly complex in nature. When evaluating surfaces, the standard interaction protocols are prerequisites to connect functional modules and the system framework together. These protocols specify how to carry out data exchange, process control, and message map.

5) **Develop the system framework and functional modules.** As functional modules are independent parts and separated from others, they can be implemented without taking into account others but only within predefined interaction protocols. The system framework is an essential part for the entire software system, and it functions as a container in which functional modules can be executed. Functional modules are responsible for practical analysis activities, and they can be reconfigured dynamically.

6) **Construct the user customisation mechanism.** The development of functional modules are not only realised by system developers but often by system end users. Although developers can implement as many functional modules as they can, it is still not enough for all users in that it sometimes requires specific analysis algorithms for particular surface characterisation. Furthermore, the emergence of novel analysis methods is ceaseless, and it is impossible for developers to cover all of them. The customisation mechanism enables users to develop proper algorithms or methods, for a certain surface evaluation cases, where the developed functional module can be integrated into the software and can also be directly distributed to other users without any modification.

The fundamental idea behind the software system is to separate functional modules from the entire characterisation system absolutely. After achieving the above six objectives, a novel flexible characterisation system can be established. Future end users can integrate and customise new functional modules, yet the complexity of system architecture will not increase when reconfiguring functional modules. In addition, establishing such a flexible characterisation system is aimed at supplying a powerful software system to characterise surface textures using multiple and customised functional modules. Every end user can concentrate on individualised characterisation algorithms and methods, and then develop his or her own functional modules and distribute them. In essence, the proposed system will facilitate communications among different organisations and promote the deepening use surface characterisation techniques.

1.3 Approach

During recent decades, with the advancement of computer technology and the consequent software development is undergoing a huge paradigm shift: From the beginnings of process-oriented approaches, through to object-oriented and later, to the current aspect-oriented and component-oriented, software engineers have been working to simplify the development, implementation, and maintenance of software applications [20].

At present, object-oriented programming is still widely used in software development. Nevertheless, it is not convenient and suitable for establishing flexible architecture to achieve fine-grained cohesion. CBD (Component-Based Development) as the successor of object-oriented development is normally highly cohesive, and it has become an increasingly popular approach to facilitate the development of evolving systems as it promises to address some of the problems of object-oriented development technologies [21]. The advantages of component-oriented development over object-oriented method are listed as follows:

- Extensibility: Object-oriented applications can normally be adaptive to additional requirements by modifying some existing code or appending some new codes [22]. As the generic architecture is not explicit enough, a great deal of detailed study is required to find out where the amendment is needed. By recompiling and rebuilding the whole framework, an extended version can be accomplished and re-released. In contrast with object-oriented, CBA (Component-Based Architecture) is very systematic and the applications are composed of components, which are independent blocks. Without modifying the system framework, the applications can be extended by adding new components or replacing improper components.

- Reusability: It is well known that class is the elementary unit of the object-oriented development. The reusability of the class is concentrated on, while encapsulation and inheritance are the mechanisms to facilitate reuse. However, this is white box reuse and developers need to have a detailed understanding of the structure of class [23, 24]. It takes much time and effort for developers to realise the reuse. Thus the object-oriented development does not give an optimal

solution for the reusability of software development. CBD technology has enhanced the reusability of software. Components are thought as independent software modules, and the applications are composed of combinations of various components.

- Maintainability: As the system application is a whole chunk of binary code with the object-oriented development, it is quite difficult to find out where any bugs are in the system. In addition, any code modification may cause a system to be unstable. The objective of the component based software development technology is to take elements from a collection of reusable software components and build applications by simply "gluing" them together [25]. Thus, it is the component that is the unit for maintenance instead of the whole system. After modifying or replacing the incorrect component, there is no need to re-compile, distribute and deploy the system again. Each component can be connected seamlessly to the system, and so the system application is easy to maintain.

Simply, compared to the object-oriented development technique, component-oriented is supposed to realise a coarse-grained and loosely coupled system architecture, which supports dynamic function integration. As mentioned previously, ideally system functions need to keep pace with any innovation in surface analysis and evaluation techniques, thus component-oriented development is supposed to ensure the flexibility of the system architecture which facilitates the reconfiguration of system functional modules. Therefore, the component-oriented development technique is more competent, and it is adopted here to establish the proposed characterisation system which can satisfy the indefinite function requirements, so that frequent changes and expansion will not affect the stability of system architecture [26].

Currently, component based software development has already been supported by commercial component frameworks such as COM/COM+ (Component Object Model) [27], CORBA (Common Object Request Broker Architecture) [28] and Java/RMI (Remote Method Invocation) [29]. The choice of a component-oriented technique that is more suitable for the surface characterisation software system depends on the application platform [30]. COM/COM+ is suited for software solutions developed for the windows

operating systems, CORBA for mission-critical and high-availability applications on mainframe and UNIX platforms, Java/RMI is best for the Internet and e-commerce applications that need to be ported across a large number of platforms. The proposed surface characterisation system is designed to be a stand-alone application and executes on the windows operating system. Hence, COM/COM+ is the primary component-oriented technology to be adopted.

One of the most important breakthroughs in software engineering and component based development is the UML (Unified Modelling Language) [20, 31]. UML provides a broad and precise notation for component specification, component dependencies, and their realization through collaborations among objects. The component diagram is used to model the static implementation view of a system. These features make UML the best choice to develop components and component oriented applications [32]. Therefore, UML will be employed during the design and implementation of the proposed flexible system. In addition, as COM/COM+ is a binary interface standardised for software componentry, it is also language-neutral and facilitates objects being implemented by different programming languages, all function modules of the proposed system will be implemented as system components using the C++ programming language, while the system framework will be implemented using C# programming language because of its reflection mechanism [33].

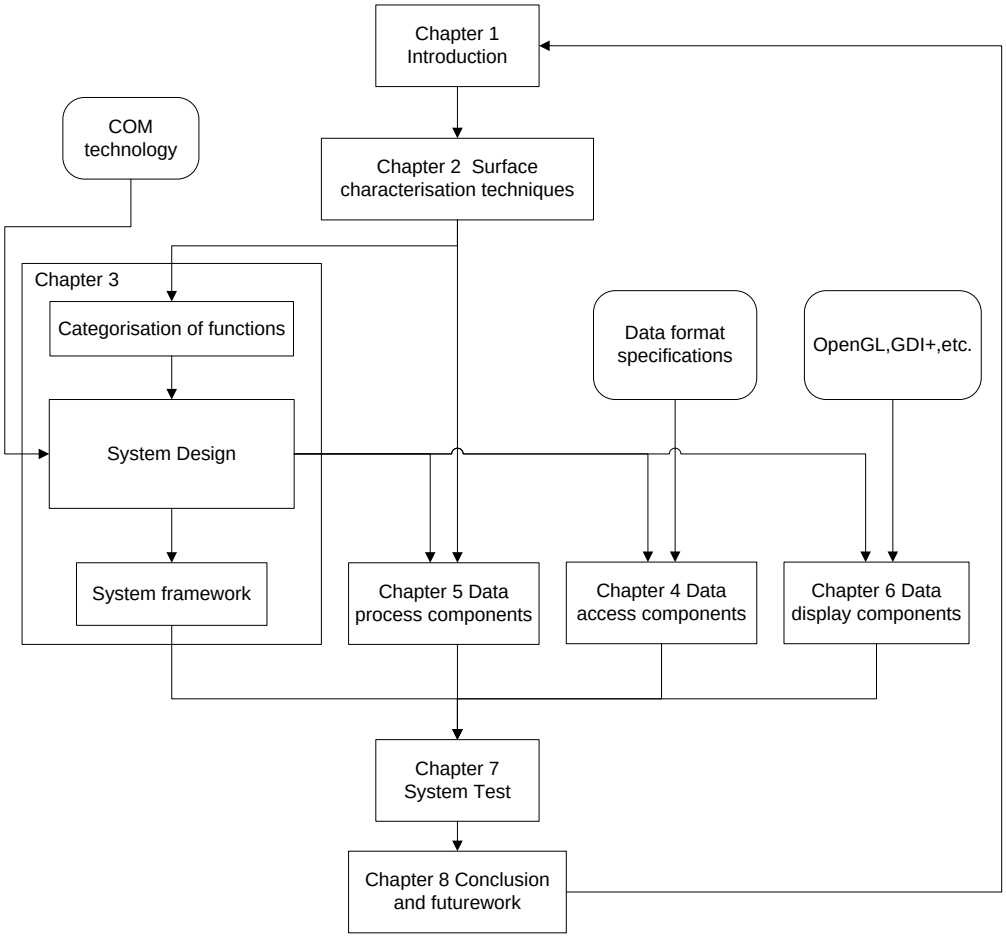
1.4 Thesis Structure

This thesis starts with an extensive literature review of advanced surface characterisation techniques in the field of surface metrology, and then constructs the flexible characterisation system with state-of-the-art software development methodologies [34, 35]. There are 8 chapters in total. Figure 1.1 illustrates the relationships and the connection between them:

Chapter 1 introduces the background and states the deficiencies of current software systems for surface characterisation in respect to reusability, extensibility, maintainability and customisation. The aims, objectives and approach of the research are put forward.

Chapter 2 provides a literature review and introduces common surface characterisation techniques which are used to evaluate the quality of a surface.

Chapter 3 focuses on the design of flexible system architecture. Firstly, component based development is discussed, which includes the framework of component based architecture, current technologies for CBA and its advantages. Then, the flexible surface characterisation system architecture is designed based on the categorisation of system functions. Finally, the development models are discussed and applied to the development of the system framework and functional components.



.

Figure 1.1: Thesis structure diagram.

Chapter 4 describes the design and development of data access components. This chapter starts with the introduction of conventional surface file formats. Then, the internal surface data structure is designed and the interface of data access components is defined. Moreover, the SDF file component is selected as an example to demonstrate the implementation of data access components.

Chapter 5 presents the design and development of data process components. As the data process components are more complex than the other two types of functional components, the categorisation of them is discussed first. Then, the interface of each type of data process components is defined respectively. Similarly, a case study concerning the geometrical analysis component is finally given.

Chapter 6 elaborates the design and development of data display components. This chapter starts with the introduction of computer graphics. Then, the interface of data display components is defined. Finally, the implementation of 3D display component using OpenGL technology is presented.

Chapter 7 deals with the system test and evaluation which is carried out by various test cases and comparison with other systems.

Chapter 8 concludes the outcomes and contributions, and suggests possibilities for future work.

Chapter 2 Surface Characterisation Techniques

2.1 Introduction

Surface metrology, which is the measurement of the deviations of a workpiece surface from its intended shape, is one of the fastest-growing areas of engineering and quality management [36]. Surface texture is an important factor in determining how a real object will interact with its environment, and it has been accepted as being significant in many fields. What happens during the interaction between two surfaces can affect the functionality and life span of a product, understanding the role of surface features in the function of a component is vital for creating and producing effective product designs and manufacturing protocols.

Using the fingernail and the eye was the original way to measure and assess the quality of surfaces. However, this is subjective and arbitrary in the sense that evaluation of the results is mostly determined by the tester. Surface features are usually too small to be assessed effectively by such a "touch and sight" [7]. The demand for effective methods to extract these surface features has stimulated engineers to explore new characterisation methods. Nowadays, surface characterisation techniques have made considerable progress. Many mathematical algorithms have been adopted to extract the significant surface topographical features, and these features can be represented and quantified by a set of characterisation parameters, which are calculated after sequential data processing. A series of GPS standards have also been released to the guarantee commonality of the characterisation process. This chapter gives a literature review of such existing surface characterisation and evaluation techniques.

2.2 Surface Verification Operator

It is well known that every workpiece is anticipated to satisfy all the criteria implied by its design, and that is the natural purpose of manufacturing. Normally, a workpiece is formed by different geometrical features and described by the inscription of sizes and angles in its nominal form. There are many geometrical features available for the designer to define the workpiece, these are described by various mathematical parameters according to the principle of computer-aided manufacturing. The goal of measuring a

workpiece is to check whether the tolerances and specifications defined by designer are met. Intuitively, a surface is a set of infinite points that separate a workpiece from its surrounding. On a real workpiece only a limited number of points can be used metrologically. Currently, the skin mode, which is a geometrical model of the physical interface between a workpiece and its environment, is widely used for the verification and evaluation of a workpiece [2].

The skin mode presents a description for geometrical product specification and verification with its associated details and on this basis—every workpiece can be geometrically defined and considered by applying manipulations of the workpiece geometry. This determination is based on mathematical rules and definitions. This therefore means that according to this determination every workpiece can be designed and on the other hand according to the design it can be measured efficiently. The skin model defines non-ideal features by consideration of ideal features at the workpiece surface. A real feature is a non-ideal feature the shape which depends on the production process and its conditions whereas ideal features exist only in theory [37].

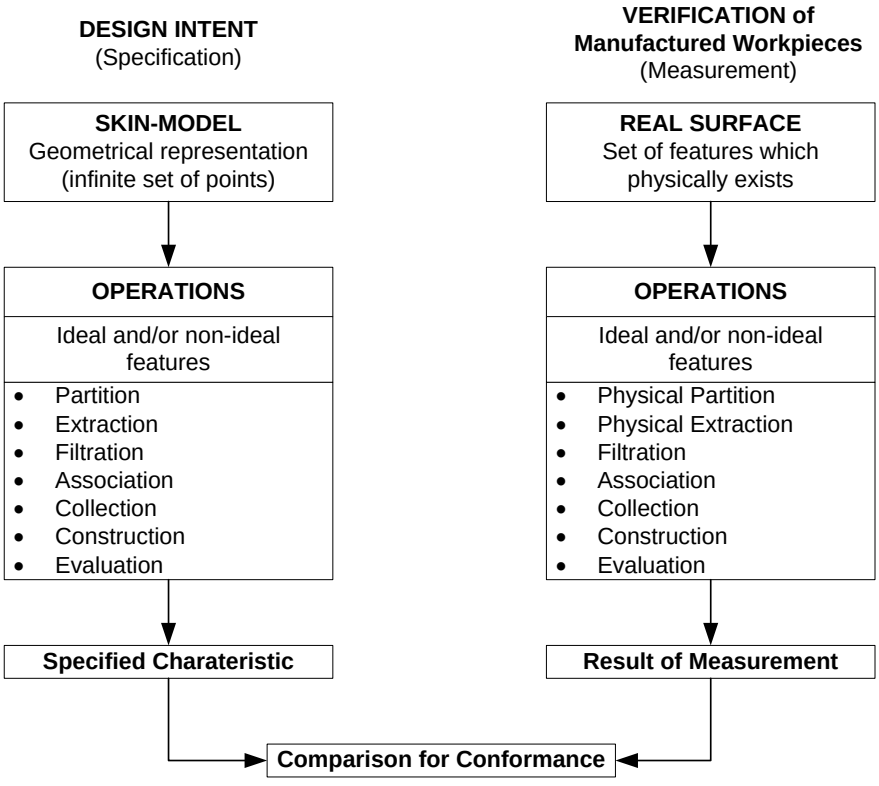


Figure 2.1: Comparison of Design Intent and Verification of manufactured workpiece.

Figure 2.1 shows the parallel procedures between “Design Intent” and the “Verification of Manufactured Workpieces” so that they comply with the design intent [38]. The Skin-Model is based on some general and basic definitions and using tools, which are named “Operations” that can be compared with mathematical operations using arithmetic techniques. The operations which are applied within the skin model are [39]:

1) Partition; used to identify bounded features.
2) Extraction; used to identify a finite number of points from a feature, with specific rules.
3) Filtration; used to distinguish between roughness, waviness, structure, form, etc.
4) Association; used to fit ideal feature(s) to non-ideal feature(s) according to specific rules, which are called criteria.
5) Collection; used to identify and consider some features together, which together play a functional role.
6) Construction; used to build ideal feature(s) from other feature(s) whilst respecting defined constraints.
7) Evaluation; used to identify either the value of a characteristic or its nominal value and its limit(s). The evaluation is always used after the feature operation(s) defining one specification or one verification.

In the Skin Model, no ordering of operations is implied in the above list. Various verification operators can be found by these feature operations applied with different orders, which are usually determined by the effects of each operation and the actual requirements. Although there are no perfect verification operators that can be used for all real workpieces or products, most of them may be verified by using the common operator consisting of all operations listed in the skin model.

The Skin Model can also be extended and applied for surface characterisation and evaluation in theory. When verifying a surface, several data processing steps are required to extract significant surface features. Thus, the whole procedure is considered to be analogous to a verification operator, while each data processing step is one operation. Not all operations defined in the Skin Model are suitable for surface data processing and some

of them are not necessary. For example, the operation "partition" is not suitable for surface data as the measured surface itself is a bounded feature of a workpiece. In the field of surface metrology, to achieve a reasonable and logical evaluation, some additional operations are used to extract specific features of surfaces. Regardless of whether operations used for surface characterisation are defined in skin model or not, these operations comprise a concrete surface verification operator. With different operations or same operations but in different orders, various operators can be formed to satisfy a variety of surface characterisation requirements.

Theoretically, any surface evaluation results should be acquired from a completed verification operator and it will be meaningless to compare the evaluation results without considering which verification operator is used. Whereas, during the procedure of surface characterisation verification operators are often less emphasised and sometimes neglected. Instead, the techniques used in each data processing step, namely operations, are well-developed and widely used for various kinds of surfaces. To enable traceability and comparability of evaluation results, it is necessary to keep records of all operations adopted in order to realise the repeatable characterisation of any surfaces. Surface verification operators can be used to precisely save and express those operations applied during the surface characterisation process. Therefore, appropriate verification operators for the realisation of surface characterisation should be encouraged and promoted widely.

At present, although no operators have been defined in ISO standards explicitly, one operation chain has already been implied in ISO 25178 and it is widely accepted and used for real surface evaluation. This operation chain can be thought as a standard operator for general surface characterisation. Each operation involved in this operator has already been separately defined in relevant GPS standards [11, 12, 40-46]. Generally, there are four significant kinds of operations in one surface verification operator, and they comprise measurement, fitting, filtering and parameterisation. Figure 2.2 shows the flowchart of the evaluation of a surface with the standard verification operator. In addition, metrologists can also generate their own verification operators for different kinds of characterisation cases. Therefore, surface verification operators should be supported in the characterisation system. Unfortunately, they are neglected in present characterisation system.

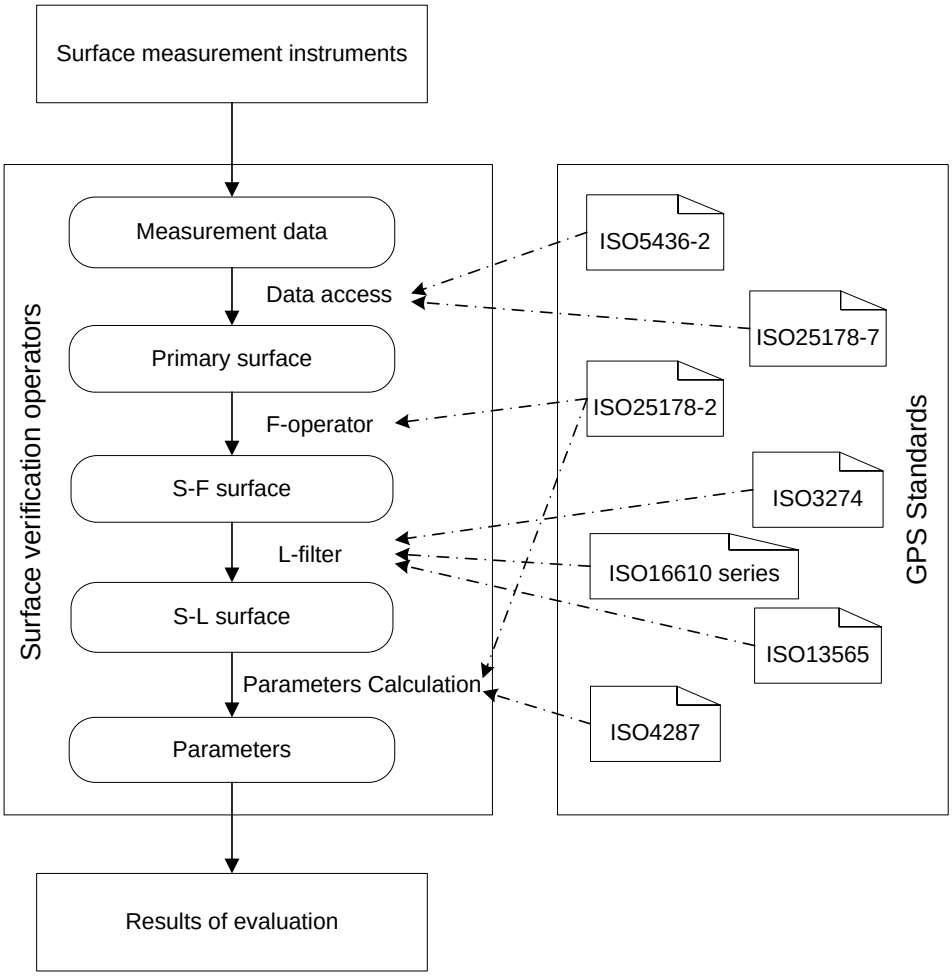


Figure 2.2: Standard surface verification operator [2].

Surface analysis and characterisation usually starts from the measurement data which is considered to be the digital representation of the original "real" surface. The measurement data is acquired from metrology instruments and usually consists of a set of discrete height information defined over the measurement plane. As various measurement techniques and measurement environments may lead to different measurement data and the accuracy and precision of the measurement instruments used have a significant influence on the final evaluation result, it is necessary to measure a surface with an appropriate instrument. In the past, a judgement was of the measurement technique appropriateness from visual inspection of the magnified representation of the surface which is reconstructed from measurement data. Nevertheless, it might be too simplistic to evaluate the measured surface in this way. As surface intrinsic features are hidden in the

height data, a set of sequential operations are required to extract salient features from the measurement data.

The standard surface verification operator mainly focuses on extracting surface roughness and waviness, and quantifying them for quality assessment. For a surface characterisation system, the initial step is to import the measured data which is stored in a digital file with a specific file format. Although profile and areal data formats have already been defined in ISO 5436-2 and ISO 25178-71 respectively, there are still a number of file formats widely used, in practice which are usually specified by instrument companies. For example, SUR is a surface format used by Mountains map and Talymap, while OPD is a Wyko specified format. It is necessary to have knowledge of the file format specification in order to access relevant measurement data files. However the fact is that these file formats can only be accessed by their conterpart surface characterisation systems. This is mainly caused by two reasons: the file format is not open because of the confidentiality of business; and it is impossible to include all these file formats in a given characterisation system. Hence, a standard file format is greatly encouraged for reasons of the compatibility and interchangeability of surface data files.

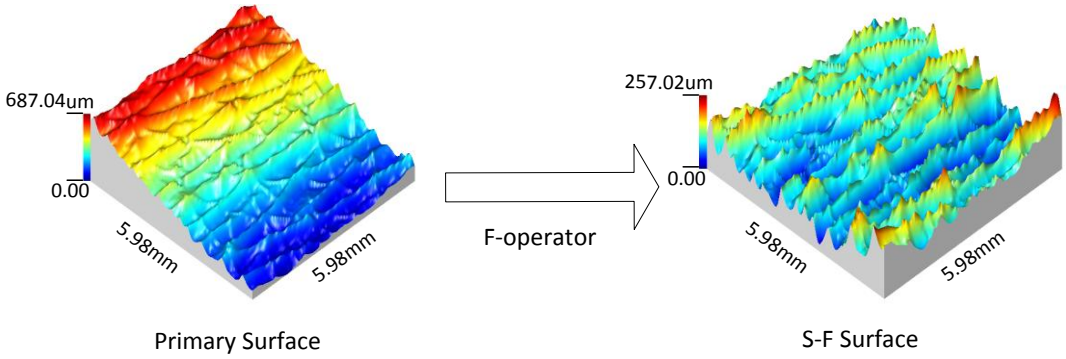


Figure 2.3: F-operator—first order polynomial fitting.

After importing the measurement data to a characterisation system, the reconstructed surface without any processing is defined as the primary surface. When measuring a surface, even after careful alignment with the measurement instrument, there is still some residual slope and slight curvature. So fitting is considered to be the first step of the process, which can rectify the measurement setup errors. This operation is recognised as

the F-operator and it removes the form component from the primary surface [12]. Figure 2.3 shows an example of using the first order polynomial fitting to remove the tilt form.

The S-F surface is derived from the primary surface by the fitting operation, and the subsequent operation is filtering. Filtering is very important in surface analysis, and it is mostly used as the L-filter defined in ISO 25178-2. The action of filtering is to extract the significant information from the measurement data for further analysis. The S-F surface topography can be divided into two parts by filtering, and usually the more significant part is the high-frequency band which is termed the S-L surface (as shown in Figure 2.4). As the surface feature of interest may be varying and the requirements of the extraction can also be diverse, taking advantage of the suitable filter with the right parameters can contribute to an effective extraction.

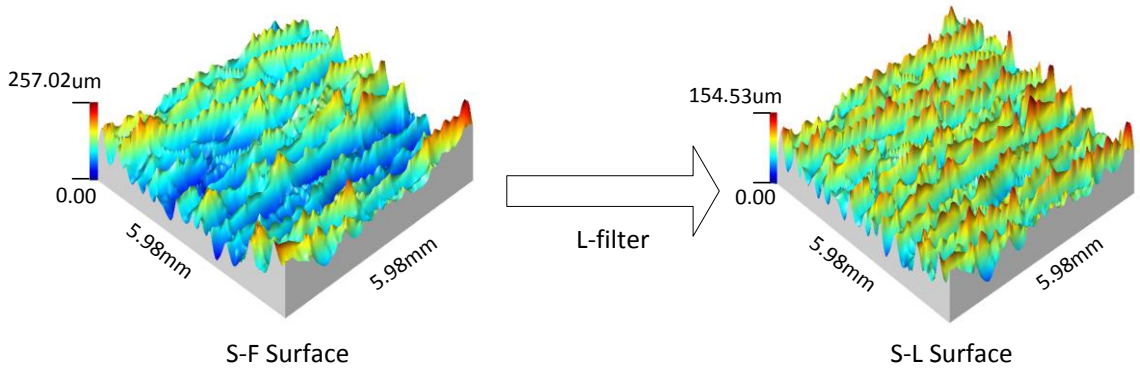


Figure 2.4: L-filter—Gaussian regression filtering.

In the last decade, significant advances have been achieved in filtering techniques. Although the Gaussian filter is widely used today and is referred to as the standard filter, it suffers from several shortcomings. The profile has distortion near the edges, and it does not follow the texture in the presence of large form deviation and is not robust against outliers. The Spline and Gaussian regression filters overcome the problem of edge distortion and poor performance in the presence of large form, while the robust Spline and robust Gaussian regression filters overcome all three problems[47, 48]. Wavelet-based filters provide an efficient way to partition a profile into multiple narrow wavelength bands. Morphological filters provide an entirely different perspective to filtering. They are curvature dependent instead of wavelength dependent. Features with sharp curvatures are not touched by the morphological structuring element, while features

with larger curvatures are touched [48, 49]. Therefore, the selection of filter type is critical when acquiring elements of the surface texture such as waviness and roughness.

Once the S-L surface is obtained, it is then more useful to calculate surface topography parameters. Parameter calculation is mostly but not always based on the S-L surface which is normally regarded as the "expected" surface. Most of the profile parameters are defined in ISO 4287 [11], while the areal parameters are defined in ISO 25178-2 [50]. Different parameters reflect the surface texture from different perspectives. There is no universal parameter that represents the surface texture well for all cases, but there are numerous parameters that successfully characterise the surface for special applications. Figure 2.5 shows the result of a typical parameter calculation for the previous S-L surface presented in Figure 2.4.

Parameters	
AMPLITUDE	
Sq	19.662(um)
Ssk	0.641
Sku	3.176
Sp	84.851(um)
Sv	69.675(um)
Sz	154.527(um)
SPACING	
Sds	8.753(1/mm^2)
Str	0.132
Sal	0.112(mm)
HYBRID	
Sdq	0.296
Ssc	0.010(1/um)
Sdr	4.430(%)
CURVES RELATED	
Vmp	1.261e+006(um^3/mm^2)
Vmc	1.696e+007(um^3/mm^2)
Vvc	2.572e+007(um^3/mm^2)
Vvv	1.062e+006(um^3/mm^2)
SK FAMILY	
Spk	23.530(um)
Sk	46.758(um)
Svk	11.165(um)
Smr1	15.1(%)
Smr2	95.6(%)
OTHERS	
Std	90.000(deg)
S5z	113.980(um)
Sa	15.807(um)

Figure 2.5: Calculation results of standard surface areal parameters.

These parameters are calculated according to their definitions in ISO 25178-2. For example, the S_a value, which is the most commonly used parameter in areal surface texture, is the arithmetic mean of the magnitude of the deviation of the surface from the mean plane [51]. It is defined as:

$$S_a = \frac{1}{L_1 L_2} \int_0^{L_1} \int_0^{L_2} \left| f(x,y) - \overline{f} \right| dxdy \qquad (2.1)$$

Where f is the height of the mean plane and L1 and L2 are the extent of the sample. F (x, y) is the surface height at (x, y). $\left| \right|$ takes the absolute value.

This standard verification operator provides a vital and effective method of quantifying surface topography. Many practical verification operators can be derived from the standard operator by applying different kinds of implementation technique or changing the parameters of certain mathematical operations. Moreover, not all surfaces are suitable to be evaluated by this standard verification operator and its derived operators. In these cases it may be more logical to create some special verification operators for a specific case of surface characterisation and evaluation, depending on the functions of the surfaces to be analysed.

2.3 Surface Measurement

The earliest methods of measuring surfaces were using the thumb nail and the eye. Although both methods were effective, they were completely subjective, did not provide quantitative results and required high skill levels. Surfaces are increasingly produced by a large variety of manufacturing processes, where each process leaves its own fingerprint on surface of workpieces, it is impossible to utilise subjective methods to assess these "fingerprints" by using such crude techniques. Demands for reliable evaluation results led to two parallel branches of instrumentation: one following the tactile example of the nail, the other imitating the eye, namely, the stylus methods and the optical methods. Currently, these are the two broad solutions for surface measurement, and they evolve to measure different types of surfaces [6]. There are other methods developed for surface measurement including capacitance techniques pneumatic methods but these are now rarely used.

2.3.1 Stylus Methods

Stylus methods were the first techniques developed for surface measurement. In the second half of the 1930s, a number of companies began to develop instruments to generate the objective information from the original surface. In 1941 for example, the first commercial instrument "Talysurf 1" emerged from Taylor Hobson in the UK, and its

guises was the first one to introduce a graphical chart representing the roughness profile [6]. However, the Brush development company in the USA was the first to coin the term profilometer [4]. From that time, the measurement of surface texture was largely based on profile profilometr. The surface profilometers instruments did not appear until the beginning of the 1980s and they were largely based on adaptations of existing profilometers using extra translation stages and independent references.

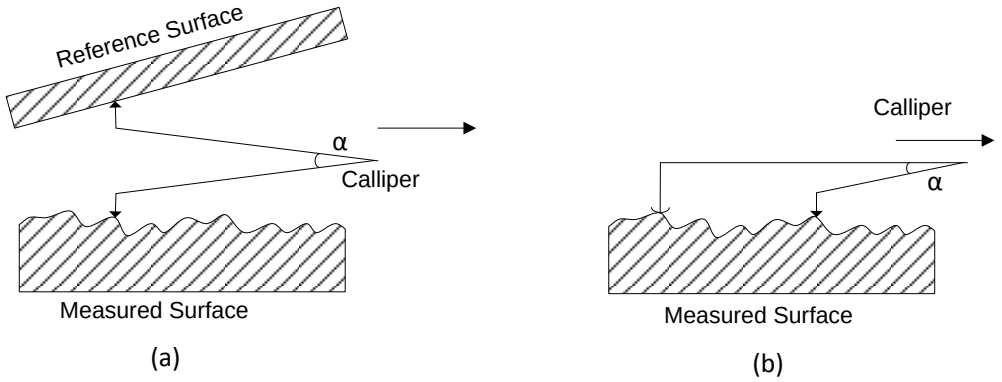


Figure 2.6: The stylus principle.

The stylus method essentially uses a form of calliper, where one arm makes contact with the surface to be measured while the other arm contacts a reference surface [52]. The arm contacting the measured surface ends with a diamond stylus whose tip dimension can penetrate the detailed geometry of the measured surface. The other arm contacts the reference surface. Figure 2.6(a) shows the basic configuration. A more common methodology is to use the measured surface as a reference by using a much blunter stylus and referencing the measurements to the blunter stylus (the skid) path as shown in Figure 2.6(b). Here the 'skid' technique provides an 'intrinsic' reference. What needs to be measured is the deviation from the intended surface shape rather than the position or dimension, so the actual position of the reference relative to the measured surface is not important [7].

The most important factor of a stylus measurement system is the stylus as the name implies and it was Schmaltz pushed the idea forward the development of the mechanical probe [53]. As the tips contact the sample surface, it is necessary to choose hard materials to avoid tip. For this reason, diamond is widely adopted as the stylus material. Figure 2.7

shows two different stylus types. The conical form is the most common stylus type by now [51].

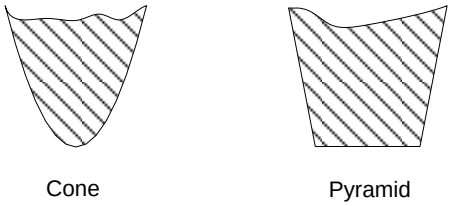


Figure 2.7: Two types of stylus—Cone and Pyramid [51].

During the measurement, stylus geometry and tip radius needs to be considered as shown in Figure 2.8. The tip can be infinitely sharp but still not penetrate completely into a surface groove or valley, in other words, the dimension of the features that can be measured is determined by the stylus radius. Moreover, if the tip is too sharp, it is much easier to scratch/damage the measured surface and wear the tip. The selection of a stylus mainly depends on the degree of spatial resolution desired. For a rough surface whose roughness is of the order of microns, there is no need to use a very small stylus. In fact, a stylus with a 10 micron tip radius would ensure reliable measurements and longer tip life.

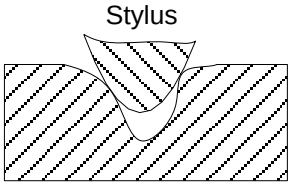


Figure 2.8: Stylus slope effect.

In addition, the process of stylus measurement can be considered as a form of mechanical filtering. Spatial features smaller than tip radius cannot be measured and consequently this means that the user will not gain any information that is below the spatial resolution of the stylus.

In practice for areal measurements, a scanning routine which specifies scanning parameters such as length, speed and sampling rate needs to be applied to achieve optimal surface measurement. The selection of these scan parameters must be carried out very carefully, as poor selection of the scan parameters will result in an unacceptable measurement. For example, a slow scan speed or too small a sampling spacing will

dramatically affect throughput, whereas a fast scan speed may lead to the intermittent stylus contact with the surface and excessive measurement noise.

2.3.2 Optical Methods

Optical measurement is a measurement technique which relies on the use of light to obtain surface topographical information. Depending on the usage of light, there are different kinds of optical methods. Some mimic the eye by using the light reflected off an area of the surface, whereas some mimic the stylus by focusing the light to one specific point [7].

Optical methods were first reported by Schmaltz to get the roughness of surfaces in 1927 [54]. From this time on, optical methodologies have been steadily developed as tools for surface metrology. At present, optical techniques have become the most common method for surface measurement metrology largely owing to the fast acquisition speeds and non contact nature of the measurement. Optical methods encompass most typically optical profilometers, confocal microscopes, and interferometers [55]. In the context of the present thesis the myriad of techniques will not be presented but the method most widely used to measure nanoscale roughness will be reviewed to illustrate its importance.

Interferometer methods utilize the coherent property of the light interference between two identical wavelength light beams from the same light source [56]. Figure 2.9 shows the classic Michelson interferometer set up. The light from a light source is split into two parts by the beam splitter. One part of the light reaches the test surface and reflects back to the detector, while the other reaches a reference surface (usually a flat mirror) and then travels back to the detector. After traversing these different path lengths, the light beams are recombined to produce interference fringes. The detector is normally a camera which contains an optical chip or CCD (Charge Couple Device) that has a light-sensitive pixel array. This chip can convert the light-intensity value for each pixel into an electronic signal with a corresponding value which will be stored in a computer. Following this the reference mirror is moved to change the length of the optical path traversed by the light beam scattered from the reference. The total movement should be restricted to several wavelengths of the light (phase shifts). Interference fringes are captured for several positions of the reference mirror. Changes in these fringes are monitored by the intensity

levels recorded by the CCD camera. Using a series of standard algorithms the intensity changes can be related to differences in the optical path length across the sample surface and the surface topography can be recovered [57].

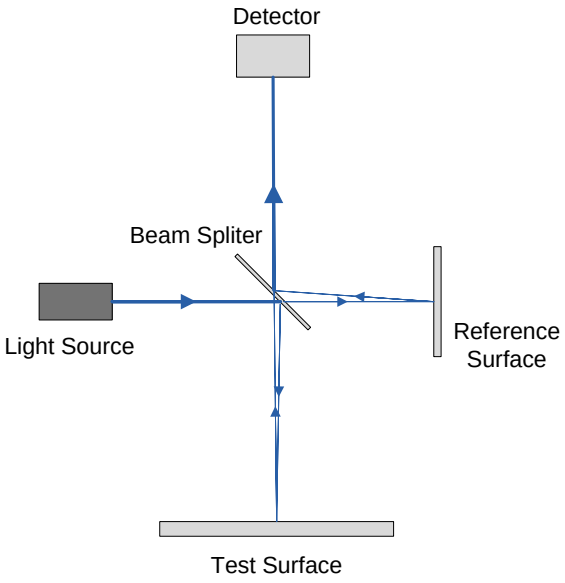


Figure 2.9: Path of light in Michelson interferometer [56].

Optical measurement can bring huge benefits as long as the capabilities and limitations are clearly understood and applied. Although optical measurement systems involve projecting light on to the test surface, they are designed to satisfy different measurement requirements and the cost of them varies. Hence, it is necessary to have a full knowledge of the scope of application and limitation of the optical measurement system that will be in use.

A competitive advantage of optical measurement is that it is non-invasive i.e. there is no need to contact the test surface [58]. Besides, there are many other advantages of this type optical measurement. For example, it can collect the information of all points within the measurement area simultaneously, so the surface can be measured quickly.

2.3.3 Other Methods

Scanning Probe Microscopy (SPM), including Atomic Force Microscopy (AFM) and Scanning Tunneling Microscopy (STM), can also be used for qualitative surface topography measurement. They are based on a powerful class of tools for nanometric

acquisition of topography data on very fine surfaces [55]. Besides, there are other possible methods for measuring surfaces but on the whole they are matched to a particular process or surface shape. One of them is the pneumatic technique which is fairly insensitive compared with other methods [7, 59], and difficult shapes would cause problems as the shape of the skirt has to match the general shape of the surface, and in addition the air has to be dry. However, it is cheap and robust. Another possibility is the use of capacitance based methods which are very sensitive and can be used effectively in certain applications [53]. Nevertheless, the signal from the transducer is susceptible to extraneous signals, so shielding has to be used for high sensitivity. Also the electrode shape should follow the general shape of the surface being measured. A further possible technique is to use ultrasound, it is possible to get phase as well as amplitude information from this technique [60]. However, it requires the very high frequencies to measure the roughness on fine surfaces and the attenuation rate is high due to the poor propagation of the ultrasonic wave. The directionality can also be poor.

2.3.4 Range and Resolution

The amplitude-wavelength (AW) map developed by Stedman is useful in describing the performance characteristics of a surface topography instrument, in terms of its ability to measure a sinusoidal surface, of a given wavelength and amplitude. An AW map shows the limits of measurement imposed by the instrument and sensor. For example, in the case of a mechanical stylus measurement, the operational region will be defined by the stylus tip radius and cone angle, the instruments' vertical and horizontal resolutions and ranges, as well as the datum accuracy. These limitations will produce polygons on AW maps that show the range and applicability of an instrument and its sensor [61]. AW maps are usually plotted with log-log scales since the boundaries become straight lines.

Figure 2.10 presents an amplitude-wavelength plot for different instruments for surface topography measurements. Each polygon in the figure indicates the working area of an instrument. The figure clearly shows that the specific working areas of the different instruments and defines the instruments' suitability for making a given measurement. The large working area of the stylus instruments illustrates its wide applicability. It should be

noted that the SPM systems have the highest resolution but limited range, while optical systems have a high resolution and a greater range than the scanning microscopes [62].

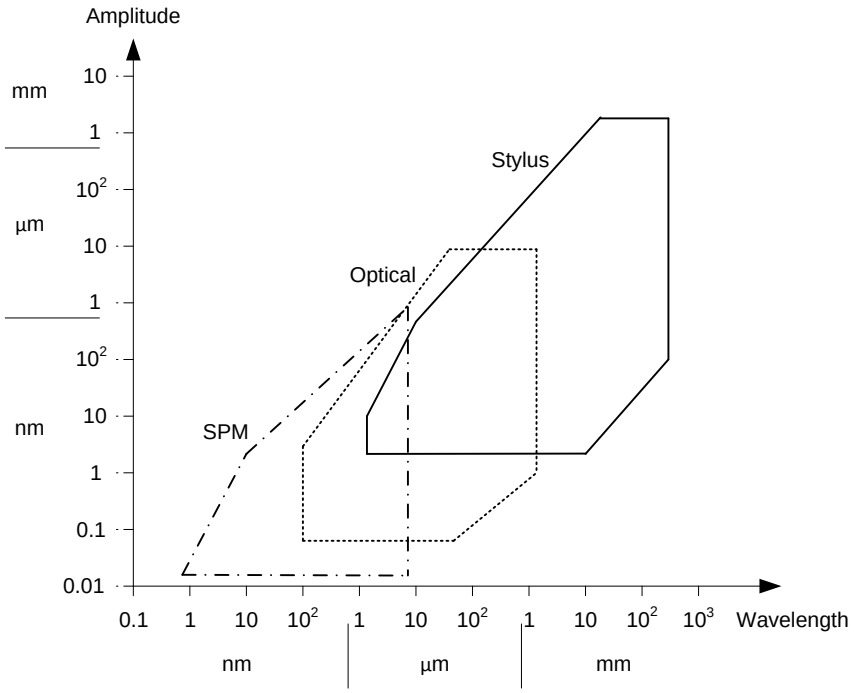


Figure 2.10: Amplitude-wavelength plot of different instruments for surface measurements [61].

In conclusion, there are many kinds of methods that have been designed to measure surfaces. Each method has its own advantages and disadvantages, it is therefore necessary to understand the most suitable technique before applying it to any given measurement task. Among these methods, stylus and optical are most widely used techniques for surface measurement. Almost every commercial surface instrument company produces instrument applying these two methods.

2.4 Fitting Techniques

The aim of fitting is to establish a base geometry from the collected data. Fitting is recognised as a necessary analysis step in coordinate metrology, as features of interest such as length, diameter, straightness etc cannot be determined directly from the data collected by the metrology instrument. Conversely, fitting is not commonly used in surface metrology [48]. However, there is no doubt that fitting is an important pre-processing step especially in areal surface metrology and is usually applied prior to

filtering. Normally it is hard to align the sample parallel an internal datum of the measurement instrument. Consequently fitting can involve removing any residual tilt after alignment by fitting lines (for profile measurement) or planes (for areal measurement). Therefore, surface fitting usually consists of the profile fitting and areal fitting, and it is to applied to fit the set of surface data points as closely as possible to a specified mathematical model [63]. Instead of providing information that can be used for process control and functional analysis, the created substitute geometry of data points mainly removes the expected form or effects of measurement and "prepares" the part of interest for further quantitative analysis.

It is well known that the measurement result may possess residual tilt even after careful alignment with the datum. In the field of surface metrology, LS (Least Squares) best fit lines and planes are the most common methods to remove this type of tilt. The following section presents a brief review of fitting of profile and areal data using LS methods.

2.4.1 Least Squares fitting

Fitting is the process of constructing a model function that has the best fit to a series of data points. It can be regarded as an over-determined problem as the unknown parameters in the model function are less than the data points [64, 65]. During the fitting process, fitting criteria which indicates how to establish the best fit geometry plays a critical role. Currently, there are many criteria used for fitting techniques [48], such as MZ (Minimum Zone), MI (Maximum Inscribed), MC (Minimum Circum scribed) and LS. In surface metrology, LS is the primary used method as it is fast and can produce a stable best-fit surface/profile.

The LS method uses a standard approach to the approximate solution of an over determined system. The objective of LS is to find the parameter values for the model function by minimizing the sum of squared deviations and it is defined as:

$$S = \sum_{i=i}^{n} \left(z_i - f\left(\mathbf{x}_i, \boldsymbol{\alpha}\right) \right)^2 \qquad (2.2)$$

Here z_i is the actual value of the dependent variable. The model function is defined as:

$$z = f\left(\mathbf{x}, \boldsymbol{\alpha}\right) \qquad (2.3)$$

The vector $\mathbf{x}$ includes all independent variables and the vector $\boldsymbol{\alpha}$ holds the m adjustable parameters. The minimum of S is found by setting the gradient to zero.

$$\frac{\partial S}{\partial \boldsymbol{\alpha}} = 0 \qquad (2.4)$$

The problem is now converted into a solution of a series of equations. According to the relationship between the deviations and unknown parameters, LS falls into two categories: linear least squares and non-linear least squares. The linear LS problem has a closed form solution which is any formula that can be evaluated in a finite number of standard operations, while non-linear least squares problem does not have a closed form solution. Therefore, the non-linear LS problem is usually solved by refinement iterations using methods such as the Gauss-Newton algorithm.

2.4.2 Profile fitting

Presently, profile analysis is still the most common way to evaluate surfaces as in industry profile contact analysis still predominates. However, areal optical analysis methods are becoming widespread because of their natural advantages of speed and non contact. In some cases, one profile is sufficient to represent the features of interested of the test surface. To obtain better analysis results, profile fitting is an important step. The aim of profile fitting is to remove or reduce the effects caused by measurement process and original form of the surface, as in most instances it is the roughness that is the subject of the analysis.

Surface profile data is usually stored as a set of sequenced height data z_i, and each z_i is associated with an incremental distance x_i to the first sampling point. The fitting task is to approximate these n points (x_i, z_i) to a predetermined function by using the fitting process. The predetermined curve/function is commonly known as the ideal function which indicates the approximate shape of original profile. Most of the common profile function equations are listed in Table 2.1.

Table 2.1: Common model functions for profile fitting.

Profile	Equation
Straight Line	$z = \alpha_0 + \alpha_1 x$

Parabolic Curve	$z = \alpha_0 + \alpha_1 x + \alpha_2 x^2$
Cubic Curve	$z = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \alpha_3 x^3$
Polynomial Curve(m-th)	$z = \sum_{i=0}^{m} \alpha_i x^i$
Exponential Curve	$z = ab^x + C$
Logarithmic Curve	$z = \frac{1}{ab^x + C}$

x and z are referred to as independent and dependant variables respectively, while others are referred to functional parameters to be determined. The polynomial equations are mostly used within the profile fitting [66]. Suppose the order of the original profile is m, there would need to be m+1 parameters determined within the relevant polynomial equation.

The sum of deviations is given by

$$S = \sum_{i=1}^{n} \left(z_i - f\left(x_i\right) \right)^2 = \sum_{i=1}^{n} \left(z_i - \sum_{j=0}^{m} \alpha_j {x_i}^j \right)^2 \tag{2.5}$$

S is then partially differentiated with respect to polynomial parameters where they are set to zero,

$$\frac{\partial S}{\partial \alpha_j} = -2 \sum_{i=1}^{n} \left(z_i - \sum_{k=0}^{m} \alpha_k {x_i}^k \right) {x_i}^j = 0, (j = 0,1,\cdots,m) \tag{2.6}$$

Equation (2.6) is rearranged and then written in matrix format,

$$X^T X A = X^T Z \tag{2.7}$$

Where

$$X = \begin{bmatrix} 1 & x_1 & {x_1}^2 & \cdots & {x_1}^m \\ 1 & x_2 & {x_2}^2 & \cdots & {x_2}^m \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & x_n & {x_n}^2 & \cdots & {x_n}^m \end{bmatrix} \quad A = \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_m \end{bmatrix} \quad Z = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_n \end{bmatrix} \tag{2.8}$$

The equation is easily solved by employing the Gauss-Jordan elimination method.

$$A = \left(X^T X \right)^{-1} X^T Z \tag{2.9}$$

Apart from a polynomial curve, other curves can also be used as model functions as long as they are the predetermined form of the measured profile. Even if the form of the profile is indeterminate, any of these model functions can be fitted and assessed post fitting to establish which is the most suitable.

2.4.3 Areal fitting

Areal analysis is becoming popular in surface metrology, and it is widely accepted as a better method to characterise a surface than profile analysis [67, 68]. Areal data is considered to be a more natural way to represent a surface as it contains more information with regard to intrinsic features. The principle of areal fitting is the same as profile fitting, namely, to establish the best-fit form for measured areal data. It is well known that the purpose of surface measurement is to acquire height information pertaining to the measuring point rather with respect to position information. Surface areal data is a form of grid data (though not real 3D data). The height value z_i is stored so that it is associated with a relative coordinate (x_i, y_i). Hence the measured data points can be represented as a 3D coordinate (x_i, y_i, z_i). Similar to profile fitting, areal fitting is achieved by approximating the data to a predetermined function of surface form.

Although many equations describing surfaces can be used as the ideal equations for the areal fitting process, polynomial equations are most commonly adopted because of the less than obvious shape of the measured surface. Other equations for surfaces can also be used for areal fitting and the process principle is nearly the same. In the following section, polynomial fitting of surface data (grid data) will be introduced, and the LS method is used to establish the best fit surface [69].

For a $M * N$ grid of data points, there are T ($T = M * N$) points in total. The n-th order polynomial surface can be expressed as

$$f(x, y) = \sum_{i=0}^{n} \sum_{j=0}^{n-i} \alpha_{ij} x^i y^j \tag{2.10}$$

The sum of deviations is given by

$$S=\sum_{l=1}^{N}\sum_{k=1}^{M}\left(z_{kl}-f\left(x_k,y_l\right)\right)^2=\sum_{l=1}^{N}\sum_{k=1}^{M}\left(z_{kl}-\sum_{i=0}^{n}\sum_{j=0}^{n-i}\alpha_{ij}{x_k}^i{y_l}^j\right)^2 \qquad (2.11)$$

For simplicity, suppose

$$w_p=z_{kl},\ u_p=x_k,\ v_p=y_l, \qquad \left(p=k+\left(l-1\right)M\right) \qquad (2.12)$$

S is then partially differentiated with respect to polynomial parameters and they are set to zero,

$$\frac{\partial S}{\partial\alpha_{kl}}=-2\sum_{p=1}^{M\times N}{u_p}^k{v_p}^l\left(w_p-\sum_{i=0}^{n}\sum_{j=0}^{n-i}\alpha_{ij}{u_p}^i{v_p}^j\right)\ \left(k=0,1,\cdots n;l=0,1,\cdots n-k\right) \qquad (2.13)$$

Equation (2.13) can be rearranged, and then written in matrix format,

$$U^TUA=U^TW \qquad (2.14)$$

Where

$$U=\begin{bmatrix}1 & v_1 & {v_1}^2 & \cdots & {v_1}^n & u_1 & u_1v_1 & \cdots & u_1{v_1}^{n-1} & \cdots & {u_1}^n\\ 1 & v_2 & {v_2}^2 & \cdots & {v_2}^n & u_2 & u_2v_2 & \cdots & u_2{v_2}^{n-1} & \cdots & {u_2}^n\\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \cdots & \vdots & \cdots & \vdots\\ 1 & v_T & {v_T}^2 & \cdots & {v_T}^n & u_T & u_Tv_T & \cdots & u_T{v_T}^{n-1} & \cdots & {u_T}^n\end{bmatrix}\ A=\begin{bmatrix}\alpha_{00}\\ \alpha_{01}\\ \vdots\\ \alpha_{0n}\\ \alpha_{10}\\ \vdots\\ \alpha_{1(n-1)}\\ \vdots\\ \alpha_{n0}\end{bmatrix}\ W=\begin{bmatrix}w_1\\ w_2\\ \vdots\\ w_T\end{bmatrix} \qquad (2.15)$$

The solution of equation can be obtained by employing the Gauss-Jordan elimination method,

$$A=\left(U^TU\right)^{-1}U^TW \qquad (2.16)$$

Polynomial fitting, whether for the profile or areal, is the most common method used for surface fitting. It is an indispensable tool in any surface characterisation system as correct alignment methods cannot be guaranteed during a measurement procedure. Although other fitting methods are not used as much as polynomial fitting, they are also important

for some special cases especially when the ideal form of a surface is clear and its equation is not polynomial e.g. a parabolic surface.

2.5 Filtering Techniques

The functional properties of engineering surfaces are relevant to manufacturing processes used to create them and each process can be considered to leave a “finger print” on the surface in the form of a unique topography [2]. The microscopic features produced by the manufacturing process play an important role, and they are crucial factors in determining the functional performance of surfaces. However, these features are not obvious and commonly cannot be “picked out” directly from the measurement data. Filtering techniques can solve this problem by partitioning a surface into different wavelength bandwidths. The underlying assumption is that surfaces consist of a series surface waves of varying wavelengths, and that certain wavelength bandwidths are related to certain aspects of the functionality of surfaces [48]. In comparison to fitting process, filtering is much more significant in the functionally significant wavelengths, and is a necessary step within any surface analysis procedure.

In the early development of surface analysis, surface data (profile data) was filtered in a graphical manner [70]. The measured profile was divided into several segments of equal length, and then one mean line was extracted from each segment to capture the slope of the profile. After connecting these mean lines together, the roughness profile was obtained by considering the deviations of the points from the mean line. The graphical method of filtering was cumbersome and time consuming. Although the method cannot partition the original profile precisely using a specific criterion, it can approximately extract the slope and roughness. Nowadays, it has been recognized that surface textures consist of fine texture called roughness, superimposed on more general curvature called waviness and long range deviations called form (as shown in Figure 2.11) [51]. On the basis of signal processing theory, filtering techniques are utilised to separate surface textures such as roughness, waviness, and form effectively.

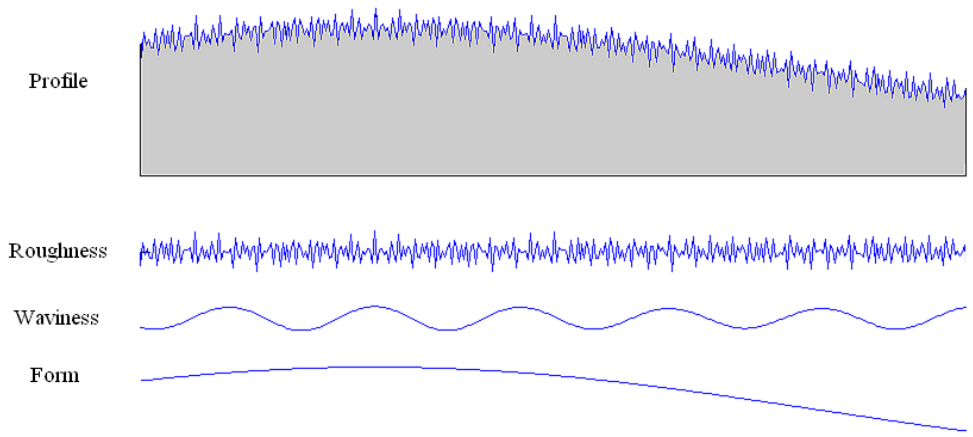


Figure 2.11: Roughness, waviness and form of a profile [51].

2.5.1 The Process of Filtering

According to signal processing theory, any signals can be viewed as comprising of sinusoidal functions of different amplitudes and wavelength. Surfaces are similar as signals, and surface textures can be considered as consisting of sinusoidal components within a range of frequency bandwidths [67]. There are two techniques available to separate surface textures from a surface: one is applied in the space domain and the other is applied in the frequency domain. As shown in Figure 2.12, path A is in the space domain, while path B is in the frequency (wavelength) domain.

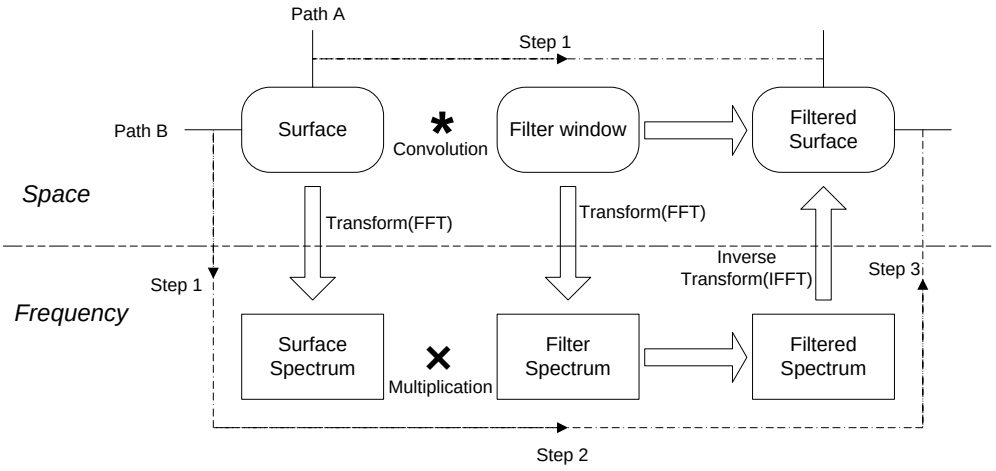


Figure 2.12: Filtering process in two domains—Space and Frequency [51].

Both methodologies are theoretically equivalent. It is more visually appealing to use path A because the surface exists in space rather than frequency. The unique step in path A is a convolution operation between the original surface and a filter window or weighting function of the filtering method. For each of filtered points, the space domain method requires a lot of multiplications and additions. In contrast, path B is more effective in computational terms but requires more steps. The first step is the transformation of original surface data from the space domain to the frequency domain, and then multiplication, the second step is carried out in the frequency domain. Finally, an inverse transform is required to obtain the filtered surface. Although there are three basic steps in path B, the calculation speed is fast as the transformations by the FFT routine is fast compared with a convolution operation [51]. In addition, it is relatively easy to change the attributes of the filter when the filtering operation is carried out in the frequency domain. Thus most filtering techniques are realised by using path B.

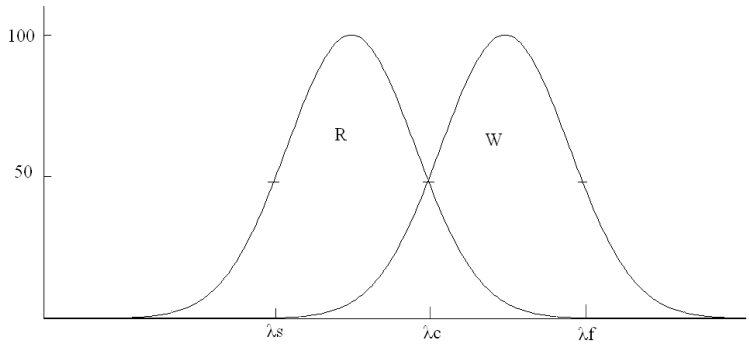


Figure 2.13: Cutoffs of surface filtering [51].

Surface textures are classified as roughness, waviness and form, and they consist of certain defined wavelength bandwidths [51]. As shown in Figure 2.13, there are three cutoffs which are required to be determined for the filtering process.

- λ_s is defined as the wavelength band between the roughness and even shorter wave components (often measurement noise),
- λ_c is defined as the wavelength band between waviness and roughness,
- λ_f is defined as the wavelength band between waviness and longer wave components such as form.

Currently, ISO 4288 and ISO 25178-3 provide guidelines for the selection of cutoff ratios, stylus tip sizes, and traverse lengths. In practice, it is suggested that users follow the standard guides to choose values for cutoffs. However, the selection of cutoffs might be determined according to the application and the intended texture of surfaces for some special cases. For example, in contact measurement, the measurement itself applies a filter with a λ_s cutoff , and the λ_s value is associated with the stylus size. In this case there is no need to set λ_s again within the filtering process in most cases, therefore, a roughness wavelength cuttoff λ_c is sufficient to extract roughness.

2.5.2 Filtering Methods

2.5.2.1 Electrical Filtering

Electrical filtering is an electrical network which transmits alternating currents of desired frequencies while substantially attenuating all other frequencies. The first electrical analogue filter to be used for surface metrology analysis was the single RC (Resistor-Capacitor) network, and it was subsequently replaced with 2RC (Two Resistor-Capacitor) filter [49]. The output of this network is not only a response of the input at any instant of time, but also those of prior input values with given reduced weights.

Although the 2RC filter is analogue and works in time domain rather than space domain and the implementations of this filter was through the use of electrical components, the hardware implementation method was later replaced by digital processing equivalents and it can be conveniently implemented in software. This digital filter became widely accepted as a “standard filter” and was introduced in many international standards [48].

2.5.2.2 Digital Filtering

In 1965, Whitehouse and Reason simulated the 2RC filter digitally [70]. The 2RC filter was described as a weighting function according to the cutoff. The weighting function was convolved with original surface (profile) to produce a running average mean line which was identical to the 2RC electrical filter mean line. Furthermore, they designed a phase correct digital filter for 2RC filter which could correct for the undesired phase shift in its output [71].

Today the use of digital filtering in surface metrology is widespread. Generally, there are three inter related steps involved in using it: specification, design and implementation. Subsequently, the Gaussian filter was introduced, and a digital filter was designed and implemented for it. With the improvement of computing capabilities, digital filtering techniques have become more and more useful within the surface analysis process [48].

2.5.2.3 Gaussian Filtering

Gaussian filtering utilises a filter whose impulse response is a Gaussian function, and it is described in ISO 11562 [43]. It is probably the most common filter in use today and it is ideally suited for smoothing surfaces with rich textures.

For surface profiles, the weighting function which determines the transmission properties of the filter is mathematically described by [72]

$$S(x) = \frac{1}{\alpha\lambda_c} \exp\left(-\pi \left(\frac{x}{\alpha\lambda_c} \right)^2 \right) \qquad (2.17)$$

Where a=04697, x is the position from the origin of the weighting function and λ_c is the long wavelength cutoff.

The Gaussian function is non zero for any x and theoretically infinite in width. However, it decays rapidly, so it is reasonable to truncate it and implement the filter directly for narrow windows. In practice, the width of the weighting function is selected to be the same as the cutoff λ_c [51]， consequently the mean line is exactly λ_c shorter than original profile as the distortion occurs at the ends of the profile and is usually removed.

2.5.2.4 Envelope Filtering

Envelope filtering initially proposed by Von Weingraber is different from other filtering techniques [48]. It generates the reference line (plane) by simulating the process of rolling a ball over the profile (surface) instead of using an averaging process. The deviations from the envelope reference line (plane) are assumed to be the texture or roughness. This filtering technique is termed the Envelope system, while others mentioned above are covered by the M system (averaging approach).

Although the M system has been generally accepted as the standard method of filtering, the E system is still indispensable in certain instances as it emerged from a functional standpoint. For example, it is sometimes believed to be a better representation of the surface where the relative motion between two contacting surfaces is critical. However, despite this the M system is usually the primary filtering choice. Fortunately, the E system has re-emerged and found new relevance within the realm of morphological filtering which is essentially a superset and offers more tools and capabilities [73]. Morphological filtering has already been an integrated part of the ISO filtering toolbox [74, 75].

2.5.2.5 Other Filters

In recent years, there have been some significant advances in filtering techniques. Although the Gaussian filter is widely used, it still suffers from some shortcomings. Some advanced filtering techniques are emerged to overcome these drawbacks. For example, the Spline and Gaussian regression filters overcome the problem of edge distortion and improve the performance in the presence of large form, while the robust Spline and robust Gaussian regression filters sort out the problem that the waviness profile would be distorted where a large peak or valley occurs [47, 67]. In addition, wavelet filtering is an effective method that can partition a surface into multi wavelength bands.

Diverse filtering techniques can be used for different applications. In practise, the selection of the suitable filtering techniques is significant for the extraction of surface textures. It is believed that there will be more advances in filtering techniques to satisfy the requirements of new applications in the future such as structured and tessellated surfaces.

2.6 Parameterisation

Parameterisation is a useful way to quantify the texture of a surface, and realise quality control for engineering processes. Due to the diversity and complexity of surface microstructures, it is impossible to use one parameter to give a complete presentation of surface texture [67]. A given parameter is usually designed to indicate surface features just from one aspect. At present, all surfaces can be roughly divided into two groups:

stochastic surfaces which have a random texture without any observable structures, and non-stochastic surfaces which have clear or regular structures such as structured surfaces [76]. Therefore, there are two primary methods used to parameterise these surface features respectively: statistical and geometrical methods. Normally, the statistical method aim to provide an "average" property description of the surface, which is more related to manufacturing processes, while the geometrical method is more appropriate from the functional point of view.

2.6.1 Statistical Methods

Statistical methods were first brought in to quantify surface textures, and are still the most widely used today. Currently, most of the surface parameters designed for profile and areal characterisation can be regarded as statistical [50]. These parameters were initially defined by industry for specific needs, and now they are defined in ISO standards such as ISO 4287 and ISO 25178-2 [11, 12]. Table 2.2 lists the parameters for profile surface, while Table 2.3 presents the parameters for areal surface.

Table 2.2: Standard surface profile parameters.

Family	Name	Description
Amplitude	Pp, Rp, Wp	Maximum profile peak height
	Pv, Rv, Wv	Maximum profile valley depth
	Pz, Rz, Wz	Maximum height of profile
	Pc, Rc, Wc	Mean height of profile elements
	Pt, Rt, Wt	Total height of profile
	Pa, Ra, Wa	Arithmetical mean deviation of the assessed profile
	Pq, Rq, Wq	Root mean square deviation of the assessed profile
	Psk, Rsk, Wsk	Skewness of the assessed profile
	Pku, Rku, Wku	Kurtosis of the assessed profile
Spacing	Psm, Rsm, Wsm	Mean width of the profile elements
Hybrid	Pdq, Rdq, Wdq	Root mean square slope of the assessed profile
Curves and related	Pmr(c), Rmr(c), Wmr(c)	Material ratio of the profile
	Pdc, Rdc, Wdc	Profile section height difference
	Pmr, Rmr, Wmr	Relative material ratio

Key:P is for profile, R is for roughness and W is for waviness

Table 2.3: Standard surface areal parameters.

Family	Name	Description
Amplitude	Sq	Root-mean-square deviation of the surface
	Ssk	Skewness of Topography height distribution
	Sku	Kurtosis of Topography height distribution
	Sp	Highest Peak from the mean surface
	Sv	Lowest Valley from the mean surface
	Sz	Height between the lowest and highest points
	Sa	Arithmetical average of the surface
Spatial	Sds	Density of summits of the surface
	Str	Texture aspect ratio of the surface
	Sal	Fastest decay autocorrelation length
Hybrid	Sdq	Root-mean-square slope of the surface
	Ssc	Average summit curvature of the surface
	Sdr	Developed surface area ratio
Curves	Vmp	Peak material volume of the surface
	Vmc	Core material volume of the surface
	Vvc	Core void volume of the surface
	Vvv	Valley void volume of the surface
SK Family	Spk	Reduced peak height
	Sk	Core roughness depth
	Svk	Reduced valley height
	Smr1	Peak material component
	Smr2	Valley material component
Others	Std	Texture direction of the surface
	S5z	Ten point height of the surface

Statistical parameters are usually relatively easy to calculate as most of them can be expressed by mathematical formulas. However, their value does not correspond directly to a physical property of the surface data. Additionally they cannot reflect local features which lie within the surface topography.

2.6.2 Geometrical Methods

In contrast to statistical parameters, the underlying rationale behind geometrical methods is based on functional requirements. In practise, geometrical methods do not focus on each data point directly, but try to extract observable geometrical features and then describe their properties and spatial distribution. The aim of these methods is to reduce the complex surface data sets to sets of objects which are analysed further. Hence, they are more natural and easy to understand. Although the extraction of texture features is very natural, the definition of suitable algorithms is very problematic and has dragged on for more than a decade. Now ISO 25178-2 has defined feature characterisation which can be regarded as a geometrical method [12]. Surface texture features which include areal features (hills & dales), line features (course& ridge) and point features (peaks, pits & saddle points) are extracted by surface segmentation algorithms. All the feature parameters defined in ISO 25178-2 are listed in Table 2.4.

Table 2.4: Feature parameters defined in ISO 25178-2.

Name	Definition	Explanation
Sds	FC; H; Wolfprune:X% ;All; Count; Density	Density of peaks
Spc	FC; P; Wolfprune:X%; All; Curvature; Mean	Arithmetic mean peak curvature
Sda(c)	FC; D; Wolfprune:X%; Closed:c; Area; Mean	Closed dale area
Sha(c)	FC; H; Wolfprune:X%; Closed:c; Area; Mean	Closed hill area
Sdv(c)	FC; D; Wolfprune:X%; Closed:c; VolE; Mean	Closed dale volume
Shv(c)	FC; H; Wolfprune:X%; Closed:c; VolE; Mean	Closed hill volume
S5p	FC; H; Wolfprune:X%; Top:5; lpvh; Mean	Five point peak height
S5v	FC; D; Wolfprune:X%; Bot:5; lpvh; Mean	Five point pit height
S10z	S5p+S5v	Ten point height of surface

The calculation of feature parameters is far more complicated than the calculation of statistical parameters. However, geometrical methods are irreplaceable for the characterisation of non-stochastic surfaces. On the basis of segmentation algorithms, some geometrical features (lines, circles) on the surface can also be extracted. In some applications, the dimension and distribution of these geometrical features are considered to be more significant.

2.7 Summary

This chapter gives a brief introduction to conventional surface characterisation techniques. Surface verification operators which consist of a series of operations are introduced and recommended for standardisation of the process of surface evaluation. Generally, in order to get the analysis result, the surfaces are processed by these operations such as Measurement, Data access, F-operator, L-filter and Parameters calculation. A number of techniques pertaining to operations are also introduced. Although there are many surface characterisation systems, the surface verification operators are not fully supported. In practice, they are normally generated by metrologists during the course of surface characterisation. As a consequence, various verification operators are likely to be created for the same surface characterisation by different metrologists. The evaluation results from different operator are greatly affected by the subjective determination, and they are impossible to be compared.

Surface verification operators are essential to be supported in a characterisation system as they can reduce the influence caused by human factors. Therefore it is necessary to develop a novel characterisation system, which can not only provide the standard verification operator but also entitle users to define their own verification operators. The verification results will be related to certain verification operator, and it is meaningful to be compared with those that are characterised by the same verification operator. In this thesis, a flexible characterisation system will be proposed. As all system functional modules are separated from the system framework and they can be reconfigured dynamically, users are able to combine them to form verification operators as they expect at the runtime. Meanwhile, the proposed characterisation system can also provide users some typical verification operators such as the standard verification operator mentioned in section 2.2.

Chapter 3 Design and Implementation of Flexible System Architecture

3.1 Introduction

Surface characterisation systems play a significant role in the field of surface metrology, and they are developed to assist in surface analysis and are based on advanced computer technology. All the numerical processes associated with surface metrology can be facilitated using such systems. At present, surface characterisation systems are commonly developed by instrument companies and some institutes, most are implemented as stand-alone systems without adequate consideration of further functional expansion. Consequently, the maintenance work and upgrading can become very difficult as time passes. This is due to the fact that any minor changes to certain elements within the system may affect on other parts and lead to other unexpected issues [18]. In addition, these surface characterisation systems are implemented for a single platform with certain programming languages. It is almost impossible to reuse their code or transport them to a new development environment and as consequence there is poor compatibility.

The NIST online system is developed as a web application which can be used on any operation system [10]. However, there is no interface for users to realise the system extension. This means all the functions can only be supplied by the system itself. "SPIP^{TM}" of Image Metrology is easier to be extended as users are allowed to develop their functional modules as plug-ins [77]. This software system is not designed only for surface metrology; therefore its available surface analysis functions are not abundant, even the standard parameters cannot be calculated. "Talymap" is a comprehensive surface analysis software system which provides plenty of functions for surface characterisation [8]. Users can also develop their functional modules for their own purpose. The drawback of its extension method is that users have to configure all text files manually, thus it is error-prone and may causes some conflicts with other parts of the system. These three systems are all developed with specific programming languages, and it is impossible to reuse their functions in other systems developed with different programming language.

It is well known that surface metrology is a rapidly developing discipline. Although many analysis algorithms and methods have already been specified in ISO standards, there are still some drawbacks to the present characterisation techniques with many being only appropriate for some surfaces under certain conditions. In other words, it is necessary to improve existing algorithms and create new algorithms to meet emerging requirements. Hence, as a software system in a research field, surface characterisation systems have to be updated and extended with the developing innovation in surface analysis and characterisation techniques [78]. In contrast to the algorithms of a surface characterisation system, the system architecture is of equal importance. It is clearly advantageous to develop a flexible system architecture that can bring huge benefits for surface characterisation systems by easily facilitating additional functionality.

Surface characterisation systems are usually supplied together with surface measurement instruments. After the measurement, surface characterisation can be finished directly with the associated characterisation system of instrument. It seems a perfect solution for surface assessment. However, there are many potential problems:

- The analysis results cannot be compared with those exported from other instruments due to the differences and incompatibilities among surface characterisation systems.
- Any errors of algorithms or functions in a surface characterisation system cannot be resolved remotely by its distributor.
- It will cost too much to realise the update of a surface characterisation system which is in use.

This chapter proposes a flexible software architecture for a surface characterisation system. The whole system is not implemented as a stand-alone chunk, instead, it is assembled by different functional components using the analogy of LEGO bricks [79]. Using this approach, system maintenance and extension becomes much easier. As modifications or changes can be completed inside the component itself without affecting other parts of the system, the deficient or redundant component can be easily removed from the system and new components can be amended without difficulty. On the other hand, functional components themselves can be reused directly by other systems without

any changes. Each component is implemented as an executable block, and it can not only be used in a surface characterisation system but also other software systems to which the interactive interfaces are transparent.

To implement a complete flexible system, object-oriented development method is insufficient. As the successor of object-oriented software development, component based development is adopted here to realise such a flexible surface characterisation system. This chapter mainly focus on the design and implementation of a system architecture using the component based development method, and the construction of a novel flexible surface characterisation system SurfStand.

3.2 Component-Based Development

3.2.1 Component

There are several definitions of the component based approach. A component is defined as a "physical and replaceable part of a system that conforms to and provides the realisation of a set of interfaces" [80]. This definition is broad and considers the component to be an organizational concept that embodies a set of functionalities that can be reused as a unit. The emphasis in this definition is on "reuse".

Microsoft Corp. has a slightly different definition. A component is defined as a "software package which offers services through interfaces" [81]. The emphasis in this definition is on "service provider". The service provider approach considers a component to be a piece of software that provides a set of services to its users. Both the "reuse" and "service provider" perspectives of a component introduce the important distinction between its public interface and its implementation in a particular environment.

This distinction addresses the issue of how the component should be designed in order to be an independent and replaceable piece of software with minimal impact on the users. In other words, components are reusable pieces of software code that serve as building blocks within an application framework [82]. The component is a language-neutral, independently implemented package of software services, delivered in an encapsulated and replaceable container, accessed via one or more published interfaces as shown in Figure 3.1. It is not platform constrained or application bound [83].

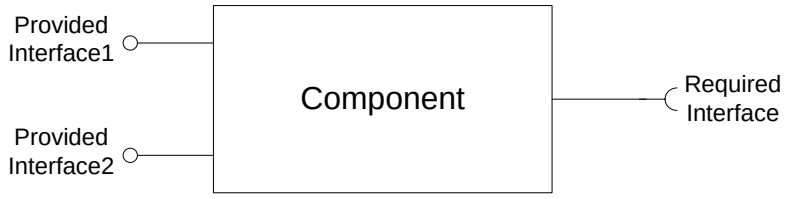


Figure 3.1: A component with provided and required interfaces.

3.2.2 Component-Based Software Development

Component-oriented software development can greatly improve the development efficiency and ease the extensibility and maintenance of large engineering software [84-86]. Component-based software applications are composed from diverse software components. Developers and sometimes end-users compose applications from often stand-alone components in flexible ways to achieve a desired set of functions. Two key aims of component technologies are to increase reusability of software in diverse situations without code modifications, and to enable end users to extend and reconfigure their applications via plug and play of components [87, 88].

The framework for CBA consists of three major parts [23]:

1) The overall system includes the CBAs facilities, services, the application components, and component managers. All the functions are separated from the overall system, and implemented in the components. The system is more like a container where each component can run; it takes charge of the construction, invocation and destruction of components. The event and message map are processed by the system.

2) The component is an isolated element, a member of an external distributed composite system, and it interacts with the system framework through a set of standard interfaces. Components should be designed to be independent units, much like Lego building blocks. They can be easily added to the system, and existing components may be detached and plugged into other systems. The services offered by components are made public by publishing their interfaces and contracts.

3) The interface is provided for components to enable asynchronous, dynamic, and anonymous communications. It provides proper connectivity between components and ensures communication between components. This is crucial to implement the dynamic connection of components rather than a statically chained function call. The interface is not only the bridge that connects the components and client; it also illustrates the functions, while the component is the function implementation. The connector is transparent to the client who does not need knowledge of the implementation of components.

The infrastructure for CBD needs three main elements: uniform design notation, standard interfaces, and repositories [23]. The uniform design notation ensures a consistent architectural diagram to describe a component's functions and properties. This is critical to design collaboration between components and to ease communication between developers. A standard interface for components allows applications in any language on any platform to access its features. This is achieved by an application in binding to the component model or IDL (Interface Definition Language). IDL is a specification language used to describe a software component's interface. IDL describes an interface in a language-neutral way, enabling communication between software components that do not share the same programming language. Repositories provide the runtime environment for components. Although a component is executable, it cannot run individually. The construction, operation and deconstruction are also managed by the repositories.

3.2.3 Current Technology for Component-Based Architecture

Component-based software development has already been supported by commercial component frameworks such as COM [27], CORBA [28] and Java/RMI [29].

- COM is a binary interface standard for software componentry introduced by Microsoft Corporation. The essence of COM is a language-neutral way of implementing objects that can be used in environments different from the one they are created in, even across machine boundaries. Since the specification is at the binary level, any interface must follow a standard memory layout which is the same as the C++ virtual function table. In addition it allows the integration of binary components written in different programming languages. Although the

interface standard has been implemented on several platforms, COM is primarily used in Microsoft Windows.

- CORBA is a standard defined by the OMG (Object Management Group) which comprises over 700 companies and organizations [89]. It enables software components written in multiple programming language and running on multiple computers to work together. The ORB (Object Request Broker) is the distributed service that implements the request to a CORBA object. It locates the object, communicates the request to the object and returns the results, when available, back to the client. Exactly the same request mechanism is used by the client regardless of where the object is located and in which programming language the object is implemented.

- Java/RMI is a Java based distributed object framework that relies on a protocol called the Java JRMP (Java Remote Method Protocol). RMI facilitates object function calls between JVMs (Java Virtual Machine). Unlike many remote procedure call based mechanisms which require parameters to be either primitive data types or structures composed of primitive data types, entire objects, even new objects whose class has never been encountered by the remote virtual machine, can be passed and returned as parameters [90]. Since the Java object is specific to Java, both the server object and client object have to be written in Java.

The choice of the suitable component-oriented technology for the surface characterisation software system is simply dependent on its application platform [30]. COM is suited for software solutions developed for the windows operating systems, CORBA for mission-critical and high-availability applications on mainframe and UNIX platforms, Java/RMI is best for internet and e-commerce applications that need to be ported across a large number of platforms.

The surface characterisation system, which is the subject of present thesis, is assumed to be a stand-alone application executed on the windows operating system. Hence, COM is the primary component-oriented technology to be adopted. The system components will be implemented in the C++ programming language using the ATL (Active Template Library) [91].

3.2.4 Advantages of Component-Based Development

There are many advantages to using component based software development to implement the characterisation system.

1) Reduction in development time and cost: The component, as an independent module, that is designed and implemented separately. The loosely coupled relationship between the components and mainframe allows the parallel development of a system. There is no doubt that the system development cycle will be shortened. In addition, many system functions can be achieved by reusing existing components rather than designing entirely new ones.

2) Reduction in maintenance costs: Because of the encapsulation of the component, it is possible to add new components and replace or remove the existing ones without affecting the system as a whole. Whatever the changes in the components, it is never necessary to recompile and reconfigure the system.

3) Enhanced extensibility and diversity: Instead of being combined into the system statically, the component can be connected to system framework dynamically. It means that the component is absolutely “plug and play”, and there is no need to take system architecture into account. Thus, the users can customize and update the system as they see fit.

3.3 System Architecture Design

3.3.1 Separation of Analysis Functions and System Framework

Generally whether a software system is suitable for a certain application is determined by its functions. System functions are of vital importance and always the concern of the users, while the infrastructure is of little interest to the end user. In a surface characterisation system, analysis functions such as fitting, filtering and parameters calculation are more emphasized in comparison to the design of system architecture. Therefore, most of present surface characterisation systems are equipped with abundant analysis algorithms but in poor system structures. Functions are tightly coupled with other functions or system framework, and it is extremely difficult to maintain the whole

system owning to these complex dependent relationships between functions and system framework.

In fact, system architecture is much more crucial for research software systems because most attributes such as the stability, maintainability and extensibility of a software system are closely related to its architecture rather than system functions [92]. However, it is impossible to design system architecture without considering relevant system functions, since system functions are always embedded in system architecture and bound to the system framework. Hence, to design and implement a flexible system architecture is to separate system functions which are prone to change from the system framework physically [83]. As mentioned in the last section, interfaces play the role of linking the system functions and framework. This following section will elaborate on the step by step implementation. For simplicity, a software system with only a few system functions is taken as an example.

3.3.2 Integrative Structure

The basic structure of system architecture can be thought as an integrative structure, as illustrated in Figure 3.2. The system function is implemented as a part of the system framework, and it can be invoked directly as all implemented details are exposed [93]. Both the framework and function are coupled with each other. The system cannot work unless every part of system is completed. Despite this, this structure is still used as it is very easy to implement as the developers do not have to pay much attention to the communication between the framework and functions. There is no need to write any code to organise system functions, instead they are part of the system framework.

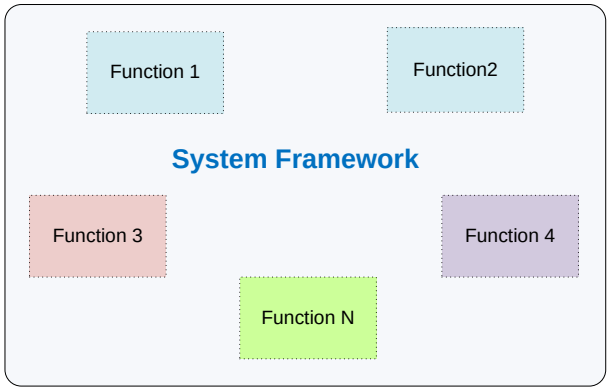


Figure 3.2: Integrative structure of software systems.

Integrative structure is suitable for some small scale software systems whose functions are relatively fixed without too many changes.

3.3.3 Semi-detached Structure

Although integrative structure is very simple and easy to realise, it is not popular for software development. The primary reason for this is that there can be problems when the number of system functions increases significantly. For example, when a bug is found in one system function, modifications may have to be completed within the corresponding code segment [94]. It can be hard to locate the code which needs to be corrected and additional bugs in other parts of the system may be generated during the process of revising earlier bugs.

To improve the integrative structure, one obvious scheme is to divide a software system into many independent parts. Each part is developed individually and assembled together to build the final system. This can then be regarded as a new system structure which could then be thought of as a semi-detached structure [95]. As shown in Figure 3.3, the system function is separated from the system framework. Thus its development and maintenance is independent and can be easily reused by other software systems. The system is a great improvement on most of present software systems are developed with semi-detached structure or similar structures.

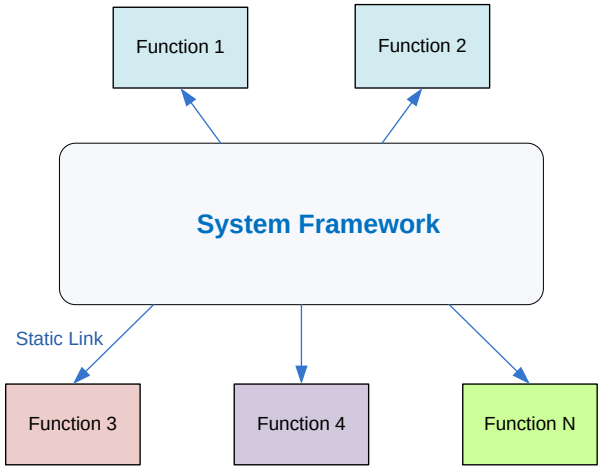


Figure 3.3: Semi-detached structure of software systems.

Although system functions are not part of system framework, the connections between them are indispensable. All system functions ought to be visible to the system framework,

while system framework is responsible for organising the system functions and invoking them when required. Consequently, developers of the system framework have to write extra code to deal with the system function elements.

3.3.4 Flexible Structure

In the semi-detached structure, the system framework and functions are separated from each other and implemented individually. However, the system framework has to know which functions are included in the whole system and provides associated codes to deal with comunication. When the number of system functions increases, these codes in the system framework will increase accordingly and become harder to maintain. In addition, the building of whole system cannot be completed until all parts of the system have been implemented. Moreover, the whole system has to be rebuilt and redeployed when modifications occur no matter which system framework or system functions are used [20].

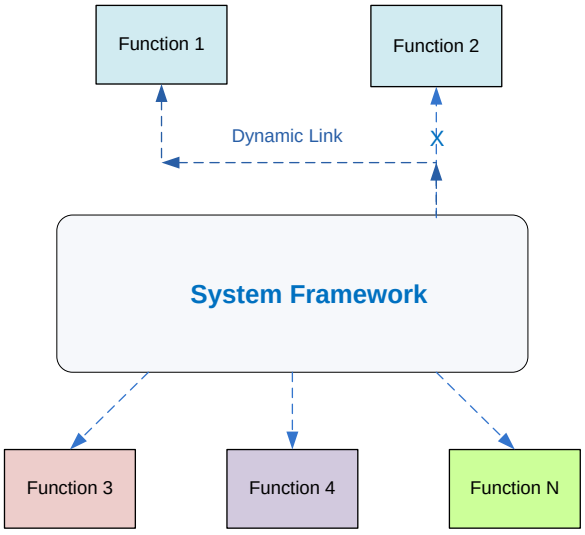


Figure 3.4: Flexible structure of software systems.

Usually, system functions need to be defined before the development of a software system. However, system requirements may change or develop over time even after the software system has been released for use. It is impossible to ensure that all the required system functions have been completed during the period of development. Some new system functions may need to be supplemented later even when the software system is in use. How to add these new system functions with minimum modifications on existing system is the key to realising the flexible structure [24, 96, 97]. Although system

functions themselves cannot be predicted, it is not difficult to find some features (the processing of data, parameters and output data) within them that are similar to existing functions. One feasible way to realise this is that the system framework connects with system functions according to their features as shown in Figure 3.4. System functions with the same features are treated in the same way and there is no need to know what the concrete functions are.

All system functions are divided into different types according to their interaction features, and one virtual system function alternate is abstracted from each type of system functions. In practise, this virtual alternate is implemented as an interface which is regarded as a bridge between system framework and real system functions. As long as a new system function implements the interface, it can be thought of as a corresponding type of system function and be invoked through the interface [84]. Therefore, system functions are not visible to the system framework anymore and they are absolutely separated from the system framework. In other words, the system framework is realised based on these abstracted interfaces rather than concrete system functions. Thus more code for communication and organisation need to be written in the framework. Obviously, the development of a system framework relies on predefined interfaces of the system functions, and every system function parts needs to implement these interfaces. Even if the system framework and/or system function parts are developed separately, the whole system is established by assembling them together without any other further changes.

In the flexible structure, interfaces acts as a common protocol that every part of the system has to comply with, and they indicate all the interactions among those independent parts. System functions can be recognised by the system framework via their interfaces [98]. Hence, the system becomes much easier to maintain and extend. Any modifications within one part will not affect other parts of the software system. Moreover, it is possible to add new function elements without any change to the original system even when it is running.

3.3.5 Proposed Surface Characterisation System Architecture

At present, most surface characterisation systems are developed with semi-detached or similar structures. System functions are developed first and then embedded to system

framework. They are all essential parts for the normal performance of the system and mostly unable to be used in other places due to the confinement of the development environment such as programming language and platform. In the meantime, maintenance will be costly for such systems. For example, the system maintainers have to be very familiar with every part of the system otherwise they will have no idea of how to deal with a problem that emerges when the system is in use. Therefore, the flexible structure is more suitable to establish a surface characterisation system.

Proposed SurfStand System Architecture

According to the principle that the system should be constructed instead of be created from scratch. System functions should be implemented as components, and the global system consists of these components which are standalone and executable entities. Generally, data importing, fitting and filtering (or any other analysis), parameters calculation and analysis reporting are essential tasks for a surface characterisation system. They are the representation of surface verification operations that are used within the surface characterisation process. The surface data flow within an ideal surface characterisation system is shown in Figure 3.5.

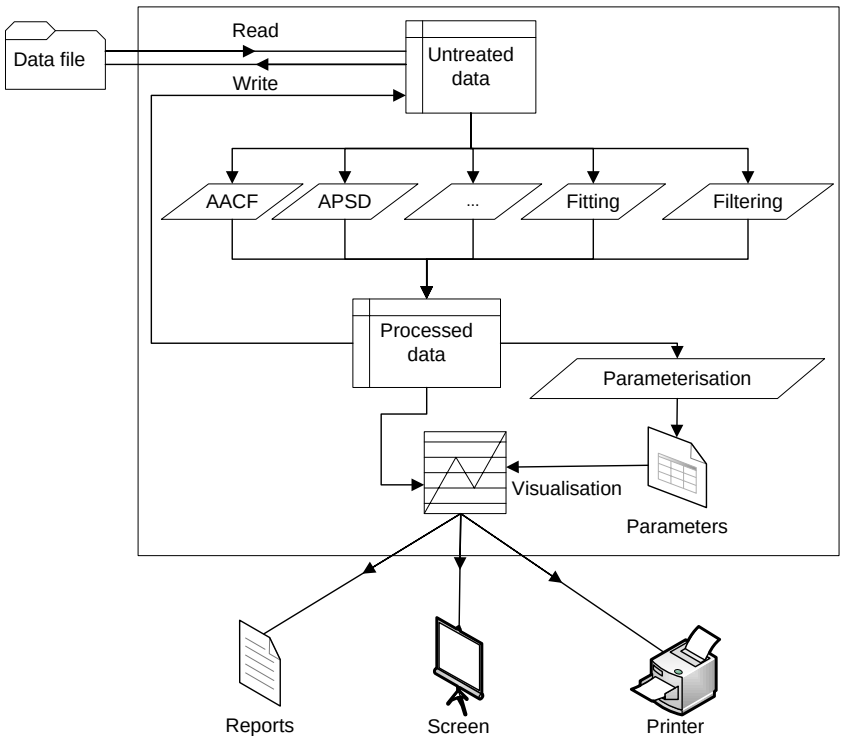


Figure 3.5: Data flow within a surface characterisation system.

After the measurement, surface data is initially stored in a data file. When evaluating a surface, surface data must be imported to the surface characterisation system from a specific file. The surface data was stored in system memory as “untreated data”. Once imported, users can select certain operations to process the untreated data according to the specific requirements. As most of the output of these operations are still surface data (though in a modified form) with the same format as the “untreated data”, these operations steps can be employed repeatedly. Following surface data treatment, the “processed data” which is expected to include the intended features such as surface roughness can be displayed or parameterised. Finally, the analysis results can be exported to screen, file reports, printer, etc.

Based on the common data process flow of a surface characterisation system, the system functions are classified as three main types: surface data accessing, processing and displaying. Data accessing refers to the exchange of the surface data between surface characterisation system and data files that are exported from instrument for storing measurement data. As different instrument companies usually specify their own file formats, it is required to parse these file formats in order to realise the data accessing function for the surface characterisation system. Data processing is the core function for every characterisation system, and it includes most of algorithms that can be applied to surface data. Data displaying is a necessary function which is used to represent surface data or analysis results for users. The more friendly the displaying components are designed, the better user experience will be.

As mentioned previously, either the system framework or any one system function ought to be a standalone chunk without any real connections with others, while the interface is the bridge between them. It is essential to define the interface before any development of those concrete components. In Figure 3.6, SurfStand SDK (Software Development Kit) is a fundamental software package which plays the core role of separating system functional components from the “SurfStand framework”. It includes the definition of all interfaces that will be used during the system connection, thus it is the essential package for the development of a system framework and system functional components. In addition, the user customisation mechanism is also based on the software package [99, 100]. Users can

develop their own functional components by realising those predefined interfaces. Some critical code fragments of Surfstand SDK are listed in Appendix A.

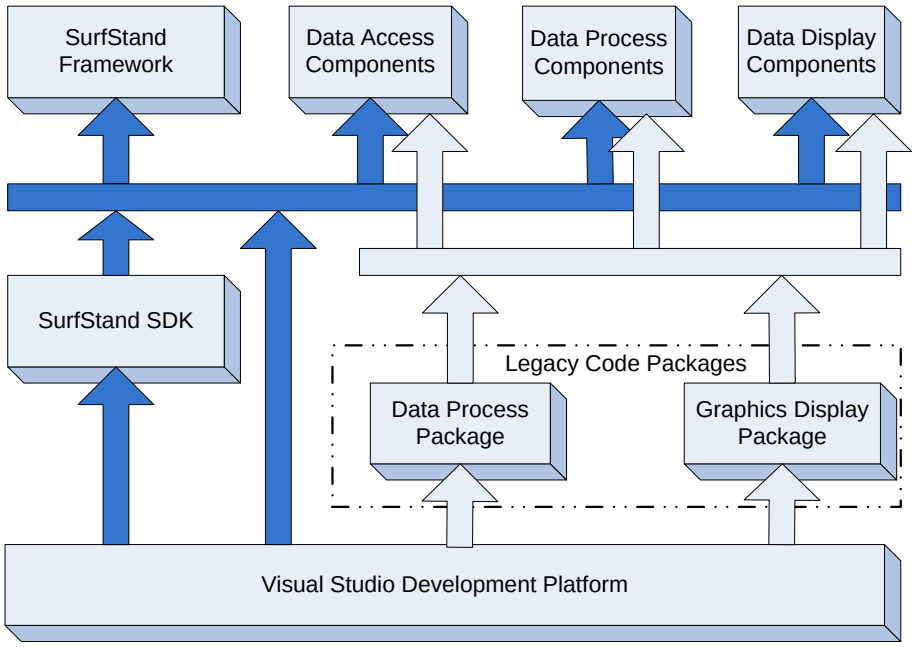


Figure 3.6: Development hierarchy chart for SurfStand.

In one word, both the system framework and functional components are developed on the basis of SurfStand SDK. The system framework requires these interfaces to complete the invocation as all functional components are not available during the developing process. All functional components are expected to derive from the predefined interfaces. Otherwise they are unrecognisable and cannot be configured in the final system.

Figure 3.7 illustrates the whole system architecture and shows the organisation of each part of the system. The system framework, as the interface of the entire system, takes charge of dealing with user requests. Within the framework, there are two important elements to enable dynamical loading of functional components, i.e. the process selector and the configuration database. The process selector is used to choose relevant functional components and carry out the specific actions by invoking those components. However, as many functional components may realise the same interface, how can the process selector find out which functional component is the correct one? The answer to this question is in terms of the configuration database which keeps the records of all useful

information with respect to available functional components when they are first added in the whole system. Therefore, every system functional component must be configured, otherwise they cannot be invoked at the appropriate time. After locating the correct functional component, the system framework can create a call out and manage this via its interfaces. It is apparent that the concrete component is absolutely invisible from the system framework and invoked dynamically. The only requirement is to configure these components correctly before invoking them [96]. In this way, it is extremely convenient to add, remove or replace any functional components even when the system is running, and users can configure their own surface characterisation system to their own design.

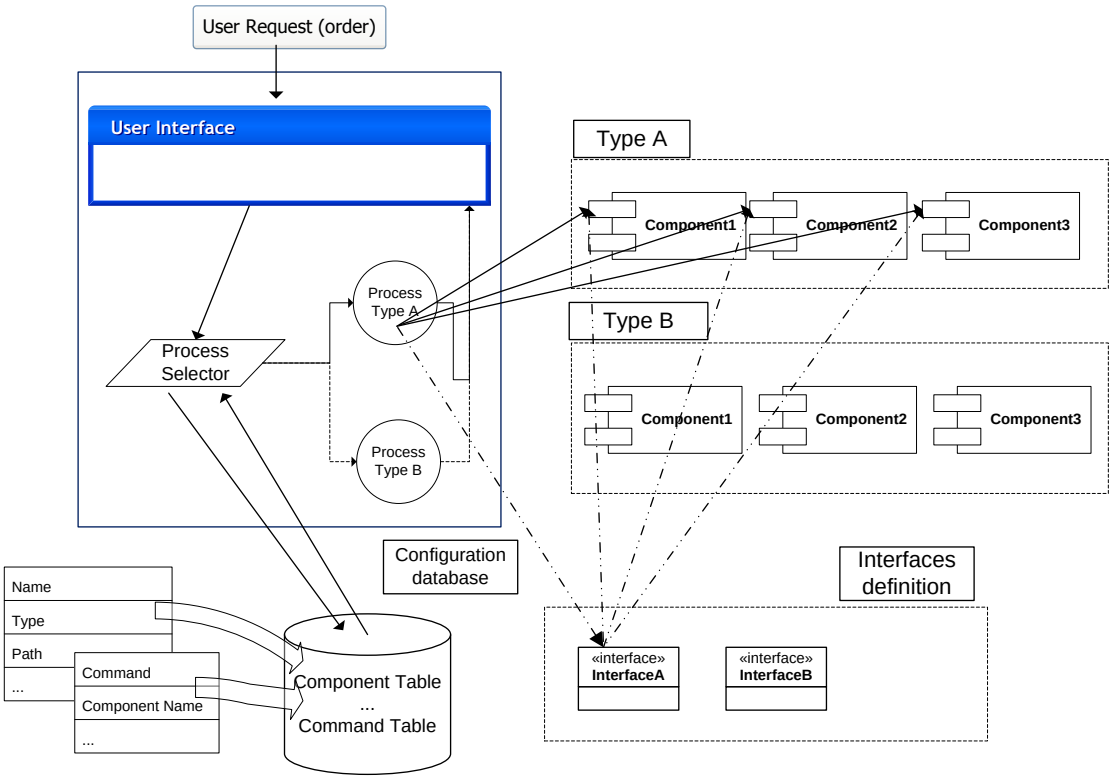


Figure 3.7: System architecture of SurfStand.

As mentioned previously, the invocation of system functional components is performed according to their interfaces which are determined by their types. In the proposed system architecture for SurfStand, there are various types of system functional components, and they are invoked in different ways within the system framework. Hence, it is necessary to classify system functions and design their corresponding interfaces before the development. The system functions categorisation is referred to Section 3.4. System

functional components of the same type have to implement the same interface so that they can be invoked in the same way. Among these interfaces, a common one for all functional components is to register some basic component attributes, such as Name, Type, Path and so forth to the configuration database. This happens when adding a new system functional component to the system. Similarly, there is also a dual interface which could be used to deregister functional components.

To sum up the proposed system framework can invoke a functional component via its interfaces at any time. The components are absolutely separated from each other. Obviously, it is allowable to change the quantity of system functional components without altering the system architecture and affecting other parts. It also implies that there is no need to rebuild and redeployment whole system when adding or removing system functions and it is desirable for the proposed expandable system. Meanwhile, the whole system is divided into many small individual parts which are much easier to maintain than a single large software system.

3.4 Surfstand System Functional Component Categorisation

In the proposed surface characterisation system, there are three major types of system functions: data accessing, data processing and data displaying. They comprise the foundation of various data analysis chains. However, they have different I/O properties and cannot be treated in a unified way. It is required to classify them into different types, so that system framework can use a unique method to invoke the functional components with the same type. Furthermore, the categorisation of system functional components is the prerequisite of the interface design, because interfaces are the only bridge between them and the system framework.

3.4.1 Data Access Component

Data access components implement system I/O functions. These components acquire the measurement data from instruments in a direct or indirect way and provide the raw data for the system. If a data access component directly acquires measurement data from a particular instrument, it means that this component is a customized component which is usually bound to the instrument and cannot be used by other instruments. For the sake of good reusability, the indirect way to acquire measurement data is encouraged as it is

platform independent. The measurement data file realises the connection between instruments and data access components. As illustrated in Figure 3.8, data access components are responsible for the data transfer between surface data files and the system framework.

No matter which kind of techniques is used to measure a surface, the result is usually stored in a file with specific format. The standard file format SDF [41, 101] for areal measurement data and SMD [40] for profile data are recommended by ISO. However, not all instruments are compatible with these standard file formats, and some instrument manufacturers create their own file format for data storing. For example, SUR is used as Mountains map surface format and Taylor Hobson surface format, while OPD is Wyko/Bruker surface format. As there are so many kinds of file format for measurement data, it is impossible to parse all of them. Furthermore some new file formats may be utilised in the future.

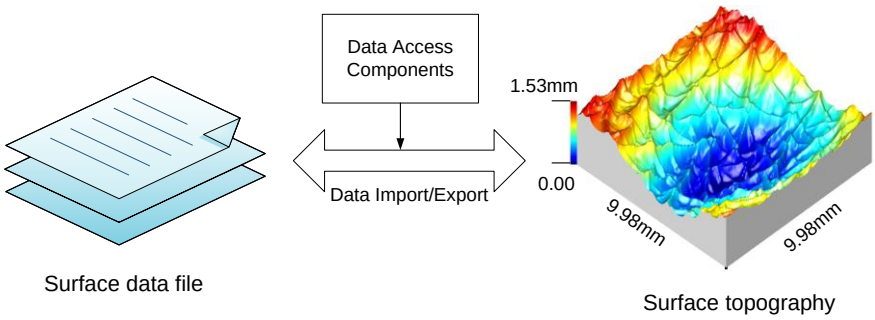


Figure 3.8: Function chart of Data Access Components.

3.4.2 Data Processing Component

Data processing components are the core part of the proposed surface characterisation system, and they are responsible for the realisation of manipulations applied to the surface data. These manipulations include not only standard surface operations defined in GPS standards but also other operations which are widely used during the process of surface characterisation such as some basic data transformation and arithmetic. Moreover, some novel algorithms or methods may be developed for surface characterisation in the future, and they may also be defined as data manipulation [5]. In Surfstand, each data manipulation will be encapsulated as an independent data process component which

could be invoked by the system framework dynamically. Thus the capacity of surface analysis and characterisation is closely related to the number of data process components.

As discussed in Chapter 2, surface characterisation is usually completed with a surface verification operator which consists of a series of sequential operations, and there are different types of operations within a complete surface characterisation procedure. Data process components can be naturally categorised according to their effects on surface data. For instance, the filtering component is the collection of all data process components which realise certain filtering techniques such as Gaussian, Spline, Wavelet, and so on. Figure 3.9 is an example of Robust Gaussian filtering component which extracts the high frequency band of original surface data.

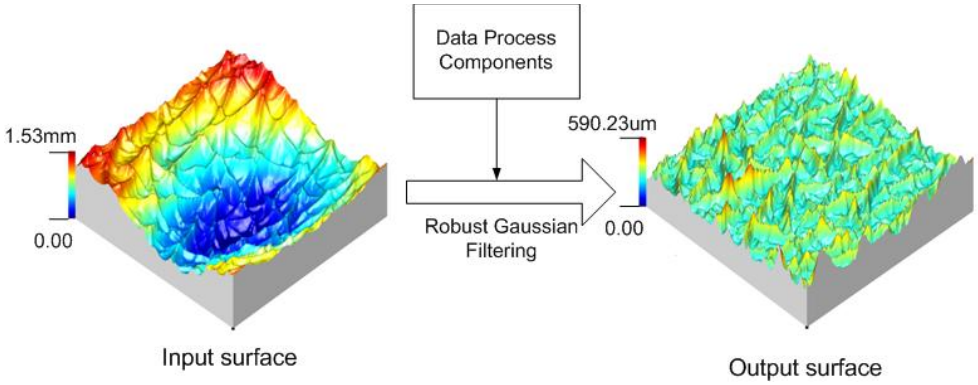


Figure 3.9: Function chat of Data Process Components.

Most of data process components have the same I/O property as the filtering component, namely, import one surface data and export one surface data. Thus it is easy to construct an operation chain, which is also regarded as a surface operator [2]. However, some data process components are different, their outputs are not surface data anymore. For example, the output of parameter calculation components is the parameter list rather than surface data. Therefore, further categorisation of data process components is required to enable the dynamic connection between them and the system framework. This will be elaborated in Chapter 5.

3.4.3 Data Display Component

Although the data display component seems to have no direct relationship with surface characterisation processes, it is helpful to have an intuitive sense of surface data and

explicit understanding of each analysis procedure. Both the input data and output data of the process component needs to be displayed by the display component. A variety of ways can be used to express a surface data. For example, a data table can list all the height values of a surface from the quantitative aspect, while a graphical view which often gives an end user a better idea of surface feature can supply a qualitative overview of the topography.

A display component only has a surface data as the input and is essentially a display control. When the system framework needs a display component to present surface data, one of these display components can be created and then be embed to the system framework. An integral surface data set usually includes a discrete data set which is a matrix storing the height and position values and data instructions concerning with the specification of intervals, offsets, units and so on. The presentation of these height values of a surface is not simple. One simple way is to plot these discrete points and connect the adjacent point, and form a grid graphic of the matrix data. However, this grid graphic cannot ideally display the surface features especially when it is be viewed from the top. Fortunately, 3D graphical display is no longer a barrier as the development of computer graphics technology has facilitated this process. Both OpenGL [102, 103] and DirectX can provide a perfect view of real objects. Figure 3.10 is the interface of a 3D display component which is implemented by utilising OpenGL technology.

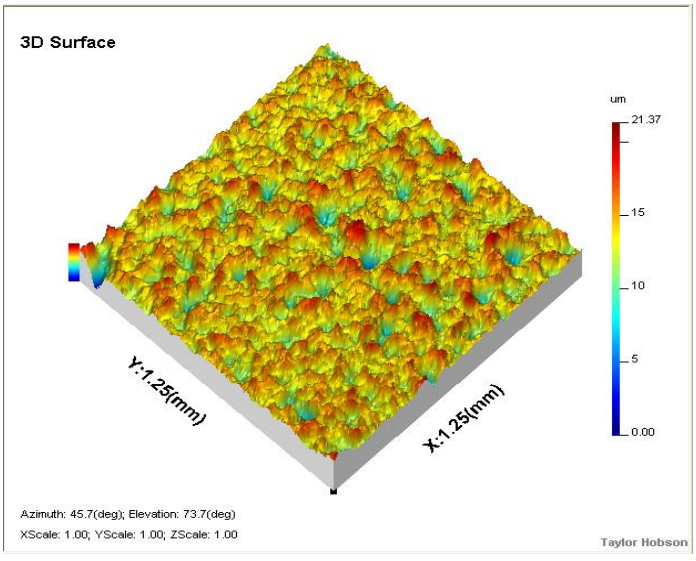


Figure 3.10: Data Display Components—3D Display Component.

3.5 System Development Life Cycle

SDLC (System Development Life Cycle) is a conceptual model which specifies how the activities of the development process are organized in the overall system development effort. As mentioned above, every part of the surface characterisation system can be developed individually, thus their life cycles are independent and have no relationships with other parts. Diverse SDLC methodologies have been developed to guide the processes involved, including the waterfall model, spiral model, incremental model and so on. Each model follows a particular life cycle in order to ensure success in the process of software development. In practise, it is possible to adopt different models to develop system parts according to their features.

3.5.1 Waterfall Model

The waterfall model is the earliest method of structured system development. The first formal description of the waterfall model is proposed by Royce in 1970 [104], although Royce did not use the term "waterfall". The waterfall model is a sequential software development process which is divided into separate process phases, and it emphasizes completing a phase of the development before proceeding to the next phase. All these phases are cascaded to each other so that second phase is started as and when a defined set of goals are achieved for first phase. Therefore, the waterfall model should only be used when the requirement is well understood and unlikely to change radically during the development [105]. It is appropriate for the development of some deterministic function components.

Although it has come under attack in recent years for being too rigid and unrealistic when it comes to quickly meeting the customer's requirement, the waterfall model is still widely used. The phases in waterfall model are: Requirement Specifications phase, Software Design, Implementation and Testing & Maintenance phase.

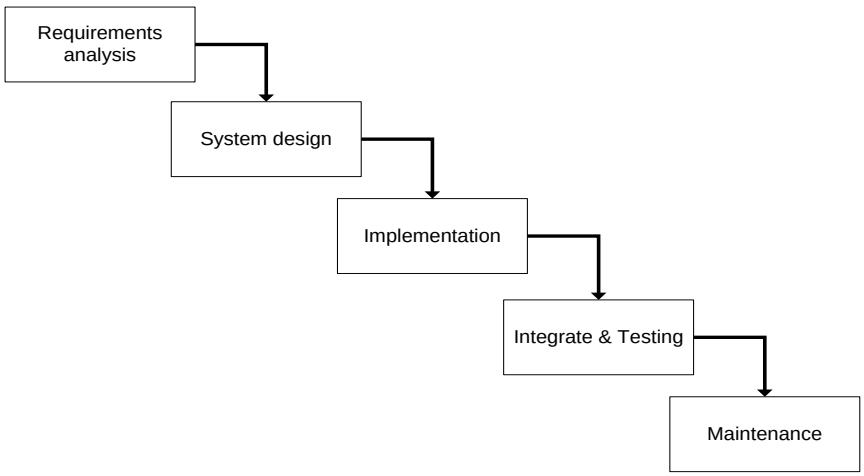


Figure 3.11: Software development model—Waterfall model.

As shown in Figure 3.11, the stages of the waterfall model are listed below:

Requirement Analysis & Definition: All possible requirements of the system to be developed are captured in this phase. The requirements are set of functionalities and constraints that the end-user (who will be using the system) expects from the system. The requirements are gathered from the end-user by consultation, these requirements are analysed for their validity and the possibility of incorporating the requirements in the system to be development is also studied. Finally, a Requirement Specification document is created which serves the purpose of guidance for the next phase of the model.

System & Software Design: Before starting actual coding, it is highly important to understand what the developer going to create and what it should look like. The requirement specifications from first phase are studied in this phase and system design is prepared. System Design helps in specifying hardware and system requirements and helps in defining overall system architecture. The system design specifications serve as input for the next phase of the model.

Implementation & Unit Testing: On receiving system design documents, the work is divided into modules/units and actual coding is started. The system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality; this is referred to as Unit Testing. Unit testing mainly verifies if the modules/units meet their specifications.

Integration & System Testing: As specified above, the system is first divided into units which are developed and tested for their functionalities. These units are integrated into a complete system during the Integration phase and tested to check if all modules/units coordinate between each other and the system as a whole behaves as per the specifications. After successfully testing the software, it is delivered to the customer.

Maintenance: This phase of the Waterfall Model is a virtually never-ending phase. Generally, problems with the system developed (which are not found during the development life cycle) come up after its practical use starts, so the issues related to the system are solved after deployment of the system. Not all the problems arise immediately but they arise from time to time and need to be solved; hence this process is referred as Maintenance.

3.5.2 Incremental Model

The incremental model performs the waterfall in overlapping sections (see Figure 3.12) attempting to compensate for the length of waterfall model projects by producing usable functionality earlier. This may involve a complete upfront set of requirements that are implemented in a series of small projects. As an alternative, a project using the incremental model may start with general objectives. Then some portion of these objectives is defined as requirements and is implemented, followed by the next portion of the objectives until all objectives are implemented. But, use of general objectives rather than complete requirements can be uncomfortable for project management. Because some modules will be completed long before others, well-defined interfaces are required. Also, formal reviews and audits are more difficult to implement on increments than on a complete system. Finally, there can be a tendency to push difficult problems to the future to demonstrate early success. If it is too risky to develop the whole system at once, then the incremental development should be considered [106].

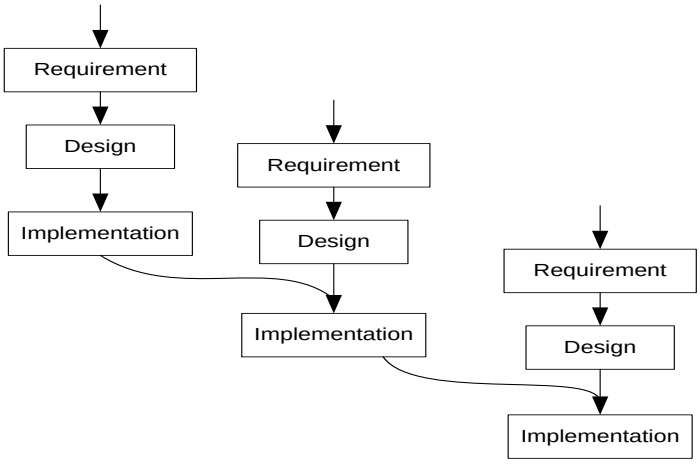


Figure 3.12: Software development model—Increment model.

3.5.3 Spiral Model

The spiral model of software development was originally proposed by Boehm [107]. As the name suggests, the activities in this model can be organized like a spiral. The spiral has many cycles. The radial dimension represents the cumulative cost incurred in accomplishing the steps completed and the angular dimension represents the progress made in completing each cycle of the spiral. The structure of the spiral model is shown in Figure 3.13 below. Each cycle in the spiral begins with the identification of objectives for that cycle and the different alternatives are possible for achieving the objectives and the imposed constraints.

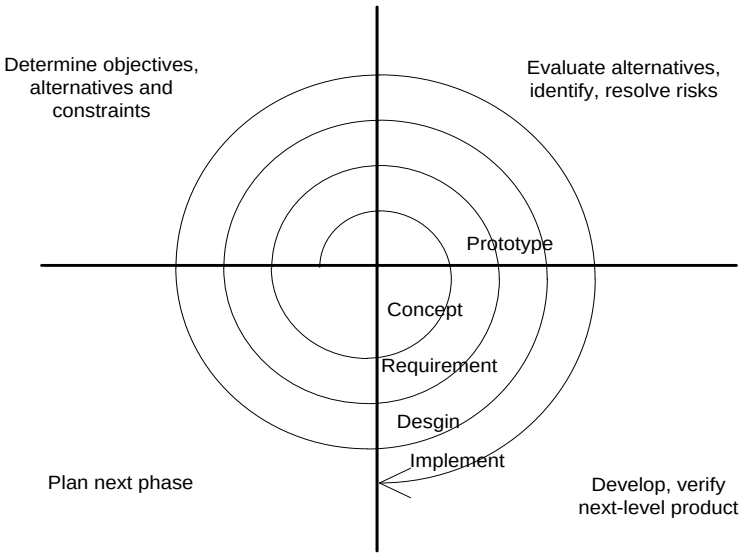


Figure 3.13: Software development model—Spiral model.

There are four phases in the Spiral Model which are: Objective setting, Risk Analysis, Development and Planning [105]. These four phases are iteratively followed one after the other in order to eliminate all the problems, which were faced in "The Waterfall Model". Iterating the phases helps in understating the problems associated with a phase and dealing with those problems when the same phase is repeated next time, planning and developing strategies to be followed while iterating through the phases. The phases in the Spiral Model are:

Objective setting: In this phase, the objectives, alternatives and constraints of the project are determined and are documented. The objectives and other specifications are fixed in order to decide which strategies/approaches to follow during the project life cycle.

Risk Analysis: This phase is the most important part of Spiral Model. In this phase all possible (and available) alternatives, which can help in developing a cost effective project are analyzed and strategies are decided on how they can be used. This phase has been added specially in order to identify and resolve all the possible risks in the project development. If risks indicate any kind of uncertainty in requirements, prototyping may be used to proceed with the available data and find out possible solution in order to deal with the potential changes in the requirements.

Development and validation: In this phase, the actual development of the project is carried out. The output of this phase is passed through all the phases iteratively in order to obtain improvements.

Customer/End User Evaluation and planning: In this phase, developed product is passed on to the customer/end user in order to receive comments and suggestions which can help in identifying and resolving potential problems/errors in the software developed. This phase is very much similar to a testing phase. A decision is made whether to continue with a further loop of spiral.

The Spiral model is most commonly used in very large software development projects. While the phases above bear similarities to the Waterfall model, the project will go through each of these four stages iteratively until the project is complete and the customer is satisfied. This allows for flexibility that is not afforded by the Waterfall model. Once the end user states that he is satisfied, the final system can be constructed

based on the very last prototype. Of course, the final system is thoroughly tested after this to make sure there are no final glitches. As with any system or piece of software, it would also be subject to routine maintenance.

3.5.4 Development Model of SurfStand

As outlined above, the new characterisation system SurfStand will be developed by component development techniques. Both the system framework and system functional component will be developed individually and they are executable chunks. This means the life cycles are independent and not confined to any other parts. Thus each part of the system can be developed using different development models.

3.5.4.1 Development Model for System Framework

The system framework is an indispensable part in the system, and it works as the main system skeleton. Apart from the real surface analysis functions, all other functions that are associated with the system are running inside the system framework. For example, users can start the system by executing the system framework part. Subsequently, all interactivities between users and system are available. As shown in Appendix B, the connection between system framework and functional components are realised through predefined interfaces. Although the system can still run smoothly, it is unable to carry out any analysis operations without any concrete functional components. Fortunately, users can configure system functions by adding or removing system functional components at any time after the launch of the system framework. In other words, users can change the configuration of system functions dynamically, and this activity would not change the phase of system framework in its life cycle.

Since the system framework is the critical part in the new surface characterisation system, it is necessary to improve and test it repeatedly to ensure its correctness and stability. Thus, the spiral model is the most suitable model for the development of the system framework. It includes four phases and they are iteratively followed one after other as shown in Figure 3.13. At the beginning of development, it is hard to set up all destinations, especially for this type of complex software system. Here the system framework is the most complex part in the surface characterisation system SurfStand, so it is better to achieve the goals and solve the problems step by step. Supposing every four

phases compose a development round which starts from objective setting, there will be a prototype system framework after each round. Developers can determine the objective of the next round by testing the prototype and comparing it with the intended one. Also the problems of the prototype will be solved out in next round. Therefore, the prototype of the system framework is expected to become better though a development round. With continuous improvement, the prototype may be released as the final system framework eventually.

Once the system framework is distributed, it will not be modified as frequently as in the initial stage of development. As long as there are no more significant user requirements which are relevant with the system framework rather than surface analysis functions, it is not necessary to start a new development round of system framework unless surface analysis models such as the analysis chain model are to be changed or there are intolerable problems that exist in the current system framework.

3.5.4.2 Development Model for System Functional Components

In the new surface characterisation system SurfStand, all the system functions associated with surface analysis processing are supposed to be separated from the system and developed as independent components. Although they are executable chunks, they cannot run without a compatible container (the system framework) in the new surface characterisation system. Therefore, the test of system functional components cannot be done before the completion of system framework. That could be thought as the only constraint which may affect the lifecycle of system functional components. However, there are many ways to simulate the test environment which is not required to be the final system framework. One straightforward way is to build a replacement of the system framework, which only realises the interactivities with system functional components and does not involve in other actions such as user interface. Fortunately, there is a better way to test COM objects. AXCTC (ActiveX Control Test Container) is a test tool distributed by Microsoft Company, and it is a container where the standard COM objects can run. Beside the final system test, all the other test activities of system functional components can be completed in AXCTC.

Since system functional components could be tested individually, it is possible to develop them with different models and they have their own lifecycle without any relationships with other parts. The selection of the development model is based on their importance, complexity and applicability. For example, some functional components are developed to assist the algorithm research, and this kind of components will not be released for end users. Thus these functional components are not so important that there is no need to concern their stabilities. Instead, the development cycle should be as short as possible to save more time for the research itself. Waterfall model is the most suitable development model for this case.

In practise, various development models may be eligible for the development of system functional components, and there are no rules to determine which development model is the most appropriate one for a specific system functional component. System functional components comprise data access components, data process components and data display components. Data access components are relatively not as complicated as the other two types, it is better to use concise development models which could reduce the development cycle. The Waterfall model is a commendable choice and it could be employed for the development of data access components. The primary reason is that the functional destination is clear and there are no complex algorithms or calculations inside.

However, most of the other two types of system functional components are not simple, and here waterfall model are not suitable. The requirement or destination is ambiguous as users are not quite clear what the component is supposed to be. Hence, those system functional components are required to be developed and tested time and time again until they are accepted by users. Moreover, this repeat development procedure also leads to more stable and robust system functional components. Incremental model and spiral model are both the development models with iterative process, and they will be primary employed in the development of data process and display components. The incremental model is more appropriate for components with multiple objectives, and each objective can be achieved step by step in an incremental way; while the spiral model is more appropriate for components with an extended objective, and it is easier to get close to the objective gradually. Certainly, this is not an absolute rule to determine the development

models, and developers of these components should take various factors into consideration and then make the decision.

Moreover, it should not be limited to these three common development models and other models can also be used for the development of system functional components when most appropriate. The waterfall, incremental and spiral models are the most common and classical development models, and they will be used for the development of “SurfStand”. Other development models such as the prototype model, V-model, rapid application model [105], etc. are not elaborated here, but they are all available for the development of system components.

3.6 Summary

This chapter emphasises on the design and implementation of the flexible system architecture. Component based development methods are introduced, and COM is adopted to be the primary component development technology. The flexible system architecture is designed by decoupling and categorising functional components. Finally, the development models are also discussed and discussed in term of the development of the SurfStand system framework and functional components.

Chapter 4 Implementation of Data Access Components

4.1 Conventional Surface Data File Format

Since the emergence of the need to evaluate a surface in a quantitative way, measurement instruments have been developed to collect the altitude information at specified positions of the surface. These altitude values compose the measurement result, which is known as surface data. Surface data, as the representation of original surface, is the basis of subsequent analysis and evaluation. Obviously it is necessary to store the surface data, otherwise the surface must be measured every time an analysis is required. In the early days, most surface instruments stored surface data in their own database and there are no interfaces to export measurement data. Users could only carry out analysis and evaluation using the embedded characterisation system of instrument. That was quite inconvenient as surface data cannot be transferred between characterisation systems and systems were likely to become out-dated quickly in terms of technology and standards development.

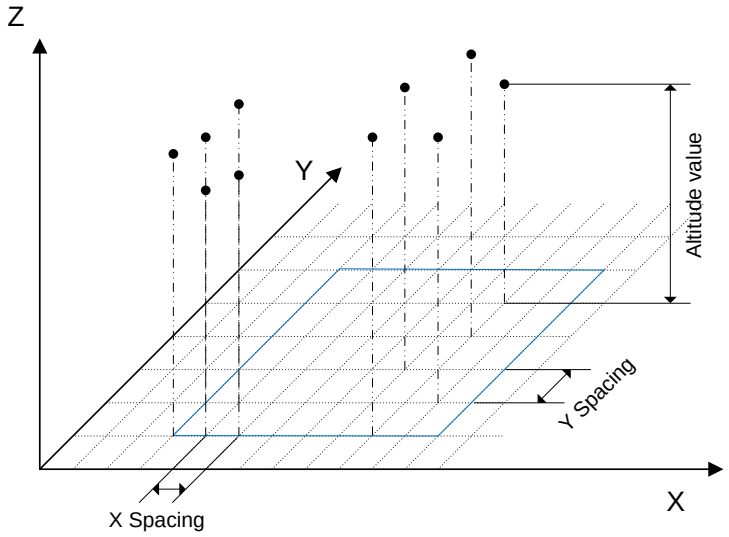


Figure 4.1: Surface data pattern—Grid Data.

As the measurement occurs at the specified position of the test surface, the altitude values are not chaotic and unordered [69]. Consequently the data needs to be stored in surface data files with specific sequence. Besides, the relevant positional information concerning altitude values is also needed to be stored. Therefore, surface data is composed of the altitude values and its relevant measurement position, and it is essential to specify the file

format before storing the surface data. For example, surface areal data is organised as grid data with equal intervals in two coordinate and altitude values in the third coordinate as shown in Figure 4.1. The actual position can be defined as long as the spacing of X and Y coordinates are known. Storage space can be saved by recording spacing information instead of the actual measurement position. This method is widely used for surface files [108].

Generally, surface files comprise two main parts: one is the header, and the other is the grid data. The header part stores information on how to reconstruct the measured surface, while the data part consists of altitude values for each sampling point. Although these two parts are compulsory for every surface file, the contents and details inside can be quite different, since they are determined by their file formats. As the aim of surface files is to store the measurement data, their formats are normally specified by instrument companies. This explains why there are so many file formats used for storing surface data each having different specification and definition. Table 4.1 lists some of the more popular surface file formats [109].

Table 4.1: Common file formats for surface data storage.

**.dat is also used by some other companies with different format specifications*

File format	Manufacturer	Data type	Encoding Type
SDF (.sdf)	ISO	Areal	ASCII, Binary
SUR (.sur)	Digital Surf	Areal	Binary
MAP (.map)	Taylor Hobson	Areal	ASCII, Binary
OPD (.opd)	Bruker Corporation	Areal	Binary
DAT (.dat)*	Zygo Corporation	Areal	Binary
SMD (.smd)	ISO	Profile	ASCII
PRF (.prf)	Taylor Hobson	Profile	ASCII
PRO (.pro)	Digital Surf	Profile	Binary

As a result of commercial confidentiality, many file formats provided by instrument manufacturers are not open to users. Surface files with these formats can only be accessed by the specific software developed by the instrument companies themselves. Consequently, in order to improve the commonality and versatility of surface files, it is necessary to specify standard surface file formats for surface measurement data. At

present, SMD and SDF file formats have been defined in ISO 5436-2 and ISO 25178-71 respectively. The former is used for profile data, the latter is for areal data. As the standard file format, they are supported by more and more surface characterisation systems. The use of standard file formats is strongly recommended; nevertheless, many popular file formats as specified by several instrument manufacturers are still widely used. No matter what format it is, a surface file should include several key elements as listed in Table 4.2.

Table 4.2: Essential elements for every surface data file.

Element	Description
Number of Points	The number of total sampling data points (It is determined by the number of points in each trace and that of traces for areal data)
Spacing	Sample spacing for each axis
Unit	Length unit of each axis (Standard unit is 'm' if it is not specified)
Data type	The type of computer data which is used to store Altitude values
Data area	A set of altitude values which is stored sequentially

One dominating distinction among different file formats is the choice of how to organise constituents in a surface file. The main constituent elements have already been presented in table, and there may be other constituents in certain file formats pertaining to particular instruments or other information associated with the specimen. In different file format specifications, the essential constituent may exist in different positions and occupy different memory space. For example, the altitude values can be saved as different data types by which the size and format of one value is determined. As the result, a surface file cannot be parsed without knowing the specification of its file format. However, since the altitude data is the primary constituent in a surface file, some file formats can be reconfigured easily because of its data is stored in ASCII (American Standard Code for Information Interchange) code. This configuration works fairly well but is not completely reliable, and it cannot be realised simply by programming as the size of each data set is uncertain and the unit is not defined. In summary, the specification of file format is the prerequisite to access a surface file, and it is also the guarantee of accessing a surface file correctly.

4.2 Component Design

4.2.1 Surface Data Structure Design for the Proposed Surface Characterisation System

Surface data is a complex data type that does not exist in any computer programming language. Instead it is composed of many individual data types. For convenience, it is necessary to define a structure for storing the whole surface data. As mentioned in last section, surface data from different surface files may be varying. Especially, the file header is composed of differing elements according to its file format specification. For example, some file formats define a “created time” to record the time when the surface is measured and stored in the file, while other file formats may not contain this element. The decision whether such an element needs be recorded in the structure is essentially a trade off between the memory space and information continuity. Within a surface file, the elements of the header occupy very little memory space by compared with the altitude values. Therefore, memory space is not a concern and the structure may contain as many elements as much as practical or pertinent to the end use of the data.

According to common surface file formats, the data structure is designed as the compound of many primitive data types such as float, double, char and so on. By composing those elements together with certain orders, the data structure is designed with the elements listed in Table 4.3.

In practise, the C++ language is the main programming language that used to create COM objectives, and the actual data structure is defined as shown in Appendix C (C.1, C.2).

The order of these elements is arranged according to memory alignment rules. Data alignment is to put the data at a memory offset equal to some multiple of the word size, and this will increase the system performance as a result of the way that the CPU (Central Processing Unit) handles memory. To align the memory data, there may be some meaningless bytes between the end of the last data structure and the beginning of the next which is called data structure padding. Hence, when designing a data structure, it is necessary to consider this effect and arrange the order more reasonably. Otherwise, the

same information may occupy more memory space than necessary when the system executes.

Table 4.3 Internal surface data structure elements.

Field	Element	Data Type	Description
Data Information	Manufacturer	Char Array	Name of manufacturer
	CreateDate	Char Array	Create Date(YYYY/MM/DD)
	ModifyDate	Char Array	Last modified date (yyyy/mm/dd)
	NumPoints	long	The number of points per profile
	NumProfiles	long	The number of profiles
	XScale	float	Scaling factor of X axis
	YScale	float	Scaling factor of Y axis
	ZScale	float	Scaling factor of Z axis
	XUnit	Char Array	The unit of X axis
	YUnit	Char Array	The unit of Y axis
	ZUnit	Char Array	The unit of Z axis
	XOffset	float	Offset in X axis direction
	YOffset	float	Offset in Y axis direction
	ZOffset	float	Offset in Z axis direction
	IsMetricUnit	bool	Unit system (True: Metric; False: Imperial)
	Description	Char Array	Description for surface data such as process records
Data Matrix	NumColumns	long	The number of points in each row
	NumRows	long	The number of rows
	Data	Float Array	Altitude values of all points

Although the definition given in Table 4.3 contains almost every element that appears in a surface data file, there are still some elements which are not concerned here. However, not all elements in the surface file are necessary to restore the measured surface. A surface can be recovered as long as the essential altitude values, scale parameters and units are given. When accessing a surface file, there are three scenarios:

1) The element (such as Element A in Figure 4.2) existing in a surface file can be found in data structure.

2) The element (such as Element B in Figure 4.2) existing in a surface file cannot be found in data structure.

3) The element (such as Element C in Figure 4.2) defined in data structure does not exist in the surface file.

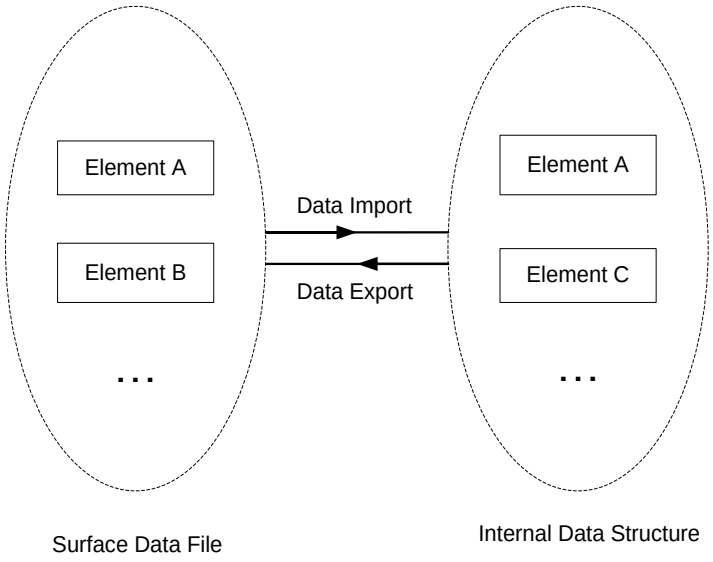


Figure 4.2: Data transfer between surface data file and internal data structure.

Also illustrated in Figure 4.2 is the fact that there are two directions of surface data transfer between surface files and the system internal data structure. In order to import surface data from any surface file or export surface data to a file seamlessly, three rules have to be satisfied for the previous scenarios:

1) Elements which exist in both sides are copied directly, and essential type conversions are sometimes required.

2) Elements which only exist in the transfer source are ignored.

3) Elements which only exist in the transfer destination are set with default values.

The first scenario is the most common and it is the simplest way to conduct the data transfer. There are no difficulties in obtaining the element from surface file and storing it in the data structure, and vice verse. In Scenario 2, the element will be ignored when importing the surface data from the file as there is no memory space reserved for it. Conversely, an element is set with a default value according to its specified type. For

instance, if the type is ‘float’, the value is set to be ‘0’. This default setting makes the data transfer smoothly and has no influence on the surface data because the element in Scenario 2 is not an essential part of surface data. Scenario 3 is similar as Scenario 2. The difference is that the element exists in the data structure instead of surface file. Thus the element can be ignored when exporting surface data to files, while it is set with default value when importing surface data.

Type conversion always occurs during the process of data transfer, because the specified types of elements may be different between surface file and data structures. It is well known that numerical values stored in computers can be expressed in different ways with different degree of precision. Therefore, the type conversion from high precision to low precision may lead to the loss of precision. To keep the precision of surface data in original surface files, every element in the data structure adopts the highest precision type as shown in Table 4.3. As long as the precision of the data structure is high enough, there should be no issues caused by data transfer from surface files to the data structure. On the contrary, the potential precision lose caused by data transfer from the data structure to surface files is not so critical and can be ignored, because the exported surface data will not used for calculation directly and the relative error is slight. Sometimes, the low precision types are not adequate to realise some algorithms and may lead to unexpected results. Hence, high precision types are widely selected in the data structure and are used in the present system.

Once the rules of surface data transfer between surface files and the data structure have been specified, there should be no problems for data exchange between surface files and the characterisation system as long as the surface file formats are known. Therefore, no matter what types of file format surface data are stored in, it can be converted to be the same data type which is defined by the data structure. For the present system the user defined data type is the unified surface data format in the whole characterisation system, and all functional components will use this structure to exchange surface data.

4.2.2 Surface Data Class Design for the Proposed Surface Characterisation System

Based on the surface data type defined in last section, the surface data class is designed to combine the data and the relevant data operations together. The surface data type is defined separately instead of being defined in the surface data class directly. What is of most relevance is that the surface data structure itself is a complex data type and it contains many elements. Assuming that all elements and the data operations are put together, then this is confusing and apt to bring out some unpredictable mistakes. Moreover, the object of these operations is the surface data rather than its elements. Setting them at the same level is much more reasonable, and then the levels of structure are distinct and clear.

Generally, each type of numerical data has its own operations, and surface data is no exception. Although surface data is not a pure numerical data type, its main constituent is definitely of a numerical type. Some basic operations for numerical data are still suitable for surface data such as addition and subtraction. However, these operations usually have their own constrains, and they cannot be applied on different surface data directly. In this case the requirements to check whether the surface data satisfies relevant operation constrains. For instance, surface data with different spacing or units cannot be added together, otherwise the result is meaningless and its spacing or units are unable to be set.

Beside those basic operations, there are many other operations that are eligible for surface data. As surface data mainly consists of a set of height values, there are some operations such as finding the extreme value, reversing each value, truncating height values with a threshold value, etc which are easy to implement. Regarded as the primitive operations for surface data, they occur in many algorithms and are widely used in the analysis process. The code fragments for the definitions of surface data classes are given by Appendix C (C.3), which includes all those common operations.

Many operations defined in the surface data class are conductive to the implementation of algorithms. For example, finding the extreme value of the surface data is such an operation which is often used in many algorithms. Therefore, there is no need to

implement this operation any more when developing a functional component. Instead, as an internal operation of the surface data class, they can be called when required.

4.2.3 Interface Design for the Data Access Components

Before designing the interface of data access components, it is necessary to take into account common attributes of all functional components in the surface characterisation system. In order to make functional components known by the system framework, every functional component has to be registered to the system database. On the contrary, deregistering a functional component from system database is imperative when it is not an effective part of the system any more. Hence, the root interface is designed as:

```
//Interface of all external function components
    public interface SSObject
    {
        bool OnRegister();
        bool UnRegister();
    }
```

SSObject is the foundational interface for the whole system, and every functional component has to realise this interface otherwise it cannot be added to system environment and be executed correctly. In other words, the interface of data access component has to inherit the root interface *SSObject*. It is well known that every data access component has to realise its own interface, consequently, *SSObject* will be realised as well because of the inheritance relationship.

Data access components are designed to exchange surface data between the surface characterisation system and external data sources. As mentioned earlier, surface data sets are normally stored in surface files after being measured, and they are widely accepted in various surface characterisation system. Therefore, surface files are treated as the only data source in this section. In the future, to expand the data source of the characterisation system, new data access interfaces can be added and implemented. For example, the surface data can be imported to the characterisation system directly without the procedure of storing in surface files. However, this is only a supplementary for the data source. Accessing surface data from surface files is much more critical for the proposed characterisation system in that it means the system is not bound to one or several kinds of particular measurement instruments.

It is impossible to enumerate all the file formats in use at present and some new file formats may be defined in the future. However, no matter what kind of file formats are used to store the surface data, the operations of the surface file are nearly the same and they all have similar attributes. Generally speaking, reading and writing surface data are the foundation of all operations for a file. Any more complicated operation on surface files will comprise a reading and writing operation. Therefore, reading and writing data are two main operations for a data access component. The data access component interface is defined as:

```
    //Interface of data access components
    public interface SSFileIO:SSObject
    {
        bool ReadDataFile(String fileName, int bRestoreBD, ref SSData
pdata);
        bool SaveDataFile(String fileName, SSData pData);
    }
```

ReadDataFile() and *SaveDataFile()* are both virtual methods which should be implemented as long as the interface *SSFileIO* is implemented. Table 4.4 list the instruction of these two methods.

Table 4.4: Arguments description of methods of interface SSFileIO.

Method	Argument	Description
ReadDataFile	fileName	The file name
	bRestoreBD	Whether the surface data is restored when there is bad or missing data
	pData	The data which is used to keep the data from the surface file.
SaveDataFile	fileName	The file name which indicates the destination of the surface data
	pData	The surface data which is saved

4.3 Standard Surface Data Format: SDF

There are many surface metrology instruments available, and often they have their own surface data file formats for the storage of surface data on the instrument platform. This situation has lead to an expensive file conversion procedure of converting one data file format to another if cross platform characterisation is needed. To overcome the drawbacks of the conversion of diversity of data file formats, a unified definition for a flexible data file format has been developed.

ISO 25178-71 defines the standard surface file format SDF. Similar to many standard file formats that already exists for interchange of other popular computer based data, such as PCX for image files, DXF for drawing files and ASCII for text files, the SDF format facilitates interchange of surface topographic data between different surface characterisation software packages. There are two forms of data representation: ASCII and binary [69]. In practice, the ASCII representation is commonly used for data transfer between computer platforms because there is no standard method of internal binary representation between platforms. Binary representation is expected for intra-platform use.

4.3.1 SDF File Format

The SDF file format is divided into three sections: header, data area and trailer

Table 4.5: File header description of SDF surface file.

Information Field	ASCII Record Name	Binary Data Type	Length(Bytes)
Version Number		unsigned char	8
Manufacture's ID	ManufacID	unsigned char	10
Creation Date and Time	CreateDate	unsigned char	12
Last Modification Date and Time	ModDate	unsigned char	12
Number of Points per Profile	NumPoints	unsigned int	2
Number of Profiles	NumProfiles	unsigned int	2
X-Scale	Xscale	double	8
Y-Scale	Yscale	double	8
Z-Scale	Zscale	double	8
Z-Resolution	Zresolution	double	8
Compression Type	Compression	unsigned char	1
Data Type	DataType	unsigned char	1
Checksum Type	CheckType	unsigned char	1
		TOTAL	81

1) Header

The header is an important part of an SDF file. It contains general information about the measurement system such as the manufacturer's ID, and the information that is necessary for the reconstruction of original surface such as the number of points per trace. The header occupies 81 bytes in total. Table 4.5 lists the types and lengths of its elements[41].

- Version Number

This field defines not only the version number but also the type of the data file representation. The first character of the version number is set to be 'a|A' or 'b|B'. 'a|A' stands for ASCII representation, while 'b|B' stands for binary representation. For example, "bISO-1.0" means the SDF file is stored in binary format, and the format version is 1.0. Future evolutions of this format will modify the number such as "1.1" or "2.0".

- Manufacture's ID

This field is used to record the source of the data, and it might be a hardware or software identifier.

- Creation Date and Time

This is a 12 character field that store the date and time in the format 'DDMMYYYYHHMM'. Obviously, it is used to record the data and time when the surface file is created. For instance, "121020120935" is interpreted as 12 October 2012 at 9:35 am.

- Last Modification Data and Time

This field indicates the data and time when the surface data is last modified. The format is the same as the 'creation Data and Time'.

- Number of Points per Profile

This field records the number of point of each profile, and it is not allowed to exceed one Word (unsigned int) of storage (65535).

- Number of Profiles

This field records the number of profiles. Similar as the ‘number of Points per profile’, the maximum value is also 65535. When it is equal to 1, the surface data is stored as a profile.

- X-scale, Y-scale and Z-scale

These three fields record the scaling factors of x, y and z axis respectively. They provide scaling to the standard unit of the meter. Thus an X-scale value of 1.00E-6 represents a sample spacing of 1um.

- Z-resolution

When dealing with digital data it is important to recognise the problem of quantisation rounding. After certain processing operations, the data type may be changed or re-scaled. The value of Z resolution will be changed to realise the re-quantisation of original data. It should be set to a negative number when it is unknown as some other file format may not include this value.

- Compression type

This field defines the compression type used for the data. Table 4.6 lists the supported compression types.

Table 4.6: Compression types list.

Compression Type Number	Compression Type	ASCII Compression Type
0	None	None
1	Run Length Limited	RLL

- Data type

This field specifies the data type used to store the surface data. It can be a single character value or an abbreviation of the base data type name as listed in Table 4.7

Table 4.7: Supported data types list of SDF file format.

Data type	Abbreviation	Data type code	length	Decimal value Range
unsigned char	UCHAR	0	1	0—255
unsigned int	UINTEGER	1	2	0—65535
unsigned long	ULONGINT	2	4	0—4294967295
float	FLOAT	3	4	-3.4E-38—3.4E+38
signed char	CHAR	4	1	-128—127
signed int	INTEGER	5	2	-32768—32767
signed long	LONGINT	6	4	-2147483648—2147483647
double	DOUBLE	7	8	-1.7E-308—1.7E+308

- CheckSum Type

This field contains the value of the checksum type used for maintaining data integrity. The supported checksum types are listed in Table 4.8.

Table 4.8: Checksum types list.

CheckSum Code	CheckSum Type	ASCII CheckSum Type
0	None	None
1	IntegerTrace	UIntTrace

2) Data area

The data area contains all the height values for measurement. It is the main part of a surface file, and usually occupies the most space of a surface file. The height values are stored successively in the order in which they are collected. In fact, the actual height values are obtained by scaling the stored values by the Z-Scale factor defined in the header.

Sometimes the height values at certain positions are invalid or cannot be measured especially with the optical instruments. These data points are referred to as ‘bad’ data or ‘missing’ data. Therefore, it is necessary to implement procedures which enable the analysis software to take appropriate action when such data is encountered. Normally, the identification of such data points may be achieved by setting them to a particular value within the data range which is not allowable for any valid data points. For example, the

maximum or minimum values for the particular data type used are both available to represent these data values. In the ISO 25178-7, the minimum value is recommended.

3) Trailer

The trailer is usually stored at the end of the file as a series of character values. It contains the historical information that is useful when associated with the surface file. Some measurement information which is not defined in header can be stored in the trailer, and the process information after the measurement such as filtering could also be stored. In a word, any information can be written in the trailer as long as it is believed to be useful.

4.3.2 Interface Implementation

The SDF file component is a form of data access component, thus the interface *SSFileIO* has to be implemented. Meanwhile, the parent interface *SSObject* is also required to be implemented. In other words, the SDF file component class has to override all virtual methods defined in Both *SSFileIO* and *SSObject*. The implementation of these two interfaces for the SDF file component is detailed below:

- Derived from *SSObject*:

As mentioned in Section 4.2.3, there are two methods in *SSObject*: *OnRegister()* and *UnRegister()*. They will be invoked when the component is added or removed. What needs to be done in the method *OnRegister()* is to add the component information to the components database of the system.

```
SSCommon.AddAssembly("sdf", "FileAccess.FileSdf",
System.Reflection.Assembly.GetExecutingAssembly().Location,
SSAssemblyType.FILEACCESS);
```

Similarly, it is necessary to remove the component information in *UnRegister()* correspondingly.

```
SSCommon.RemoveAssembly("sdf",SSAssemblyType.FILEACCESS);
```

- Derived from *SSFileIO*

ReadDataFile() is invoked when a SDF file is opened, while *SaveDataFile()* would be invoked when surface data needs to be saved. These two methods involve a quantity of

codes to parse the file format. The critical activity is to transfer the data information between the SDF file and the inner surface data structure.

4.4 Summary

This chapter gives a brief introduction to conventional surface data file format. The internal data structure of the characterisation system is designed and the rules of surface data exchange between surface files and the characterisation system are specified. Additionally the interface for data access components is defined. Finally, the implementation of one data access component for SDF file format is elaborated.

Chapter 5 Design and Implementation of Data Process Components of the Developed System

5.1 Categorisation of Data Process Components

Surface analysis and characterisation are a sequential set of procedures comprising several operations. According to the GPS standards, this analysis procedure is termed the surface verification operator. Every verification operator consists of a series of operations, such as fitting, filtering and parameter calculation. In a surface characterization system such as one proposed here, each operation is realised as a data process component. Therefore, data process components are no doubt the most significant and complex constituent part of every surface analysis and characterisation process. In addition, some data manipulation such as data transformation and data arithmetic procedures are also often encapsulated as data process components.

Data process components can be considered as “black boxes” which are invisible to end users. Users care about their I/O properties rather than their implementation details. From the perspective of a surface characterisation system, a data process component is available for data processing as long as its I/O requirements are satisfied. However, there are many surface data analysis and process operations, and their I/O properties are not always the same. To enable the seamless connective between data process components and the characterisations system, it is necessary to abstract the similarity of these operations and design standard interfaces for data process components. Hence, the categorisation of data process components is the prerequisite to define their interfaces. As outlined previously, their I/O properties would be the primary factors to be considered for categorising data process components.

It has been established that the aim of a data process component is to realise some form of analysis and operation based on the surface data. Naturally, surface data is the same input object of all data process components. Each data process component is required to import one surface data at least, otherwise, it is not considered to be a real data process component. Most data process components only need one surface data set (e.g. parameter calculation), while some others may need two or more sets of surface data (subtraction of

surfaces). The number of import surface data sets is dependent on the operation itself. Consequently, such number of the input data sets needed is an important factor that has a great influence on the categorisation of the data process component.

Besides the import aspects of the component, differences also exist at the export end of data process components. Normally, the output of a data process component is supposed to be a somewhat modified surface data set as well. For example, the outputs of filtering and fitting operations are still surface data sets with the same resolution and dimensions of the input data. However, as the analysis procedure, the outputs could be of a different form according to the procedure's algorithmic process. In certain instances these output data sets cannot be presented as standard display components e.g. power spectral density visualisation. Therefore, it is necessary to output a display control which can present the analysis results in the optimal way.

In addition, many data process components may have extra inputs in addition to the surface data set for them to run. For example, it is required to set up the cut-off for a filtering operation component or the polynomial order for a fitting operation component. Given that all these parameters are considered as the inputs for the data process component, it is impossible to abstract them as the same attributes for different data process component, thus there will be many types of data process components. Fortunately, there is one easy method to deal with these parameters. Supposing a data process component is a production machine, its input materials would be surface data while the output is surface data or display controls. When starting the production machine, it is required to set up any relevant parameters correctly. Certainly, different configurations of these parameters will lead to a product with different properties. Similarly, all the parameters for a data process component can be regarded as attributes of the component rather than its input. Every data process component can define its own attributes according to its internal analysis algorithms. Hence, the categorisation of data process components will not be affected by the parameters of a particular algorithm.

In short, the number of input surface data sets and the output result are the categorisation criteria of the data process components. According to the requirements of the proposed characterisation system, there are three fundamental types of data process components.

All surface verification operations can be represented by these fundamental components or their compositions.

1) Type A: One Data Set In and One Data Set Out

The operation with one input data set and one output data set is the main form within surface verification operators. The goal of characterising a surface is to obtain the intrinsic features of it, thus there is usually no need to compare with another surface during the analysis process. Usually the comparisons are made after obtaining specified features. Each operation could therefore be implemented as a "black box" by hiding intermediate data sets within the corresponding process. As long as operations are realised as this type of operation, they can be employed as a constituent part of any operator. Figure 5.1 illustrates a levelling operation, the input data of this operation can be any output data of other operations, and vice versa.

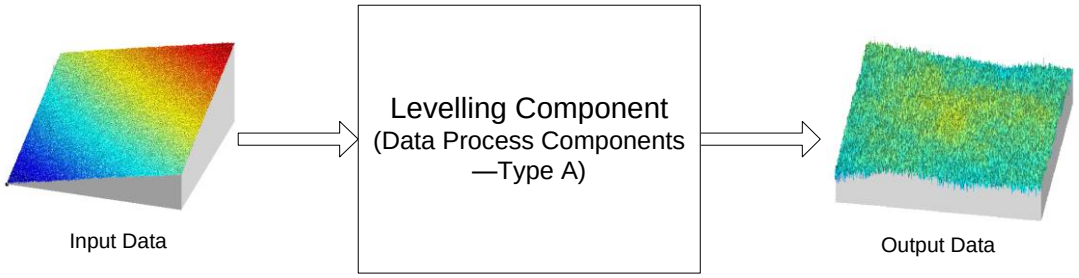


Figure 5.1: Example of Data Process Components—Type A.

As the input data includes not only the matrix data but also the data properties such as the data intervals and the operation is only deal with the matrix data, the output data has the default properties which are the same as the input data. As the most critical procedures during the surface characterisation, fitting and filtering are both considered to be this type of operation: fitting is usually the first step of the process, often rectifying the measurement setup errors. It is recognised as an F-operator which removes form from the primary surface [12]; filtering is an operation to extract the significant or important information from the measurement data for further analysis.

2) Type B: Two Surface Data Sets In and One Surface Data Set Out

In the field of surface metrology, comparison or arithmetic operations between two surfaces are quite common, although these operations are not parts of the standard surface

verification chain. For example, cross-correlation which can be used to quantitatively acquire the similarity between two surfaces is a typical operation in this category, and the arithmetic operations such as adding and subtracting surfaces are also regarded as this kind of operation.

This Type B operation has two operands, and it is the basic prototype for operations with multiple operands. Usually an operation with more than two operands can be separated into many sub-operations with two operands. For simplicity, suppose that there are three surfaces: A, B and C to be added together, the one operation (A+B+C) with three operands could be divided into two operations (A+B and C+(result of A+B)) with two operands.

However, there is one implicit constraint for most of these operations; namely, the input data A and data B should have the same resolution and spacing interval. Some particular operations which have specified the process algorithm can deal with different resolutions and intervals. Otherwise, two surfaces with varying resolution or interval cannot be process with this kind of operation. Figure 5.2 is an example of adding two surfaces together.

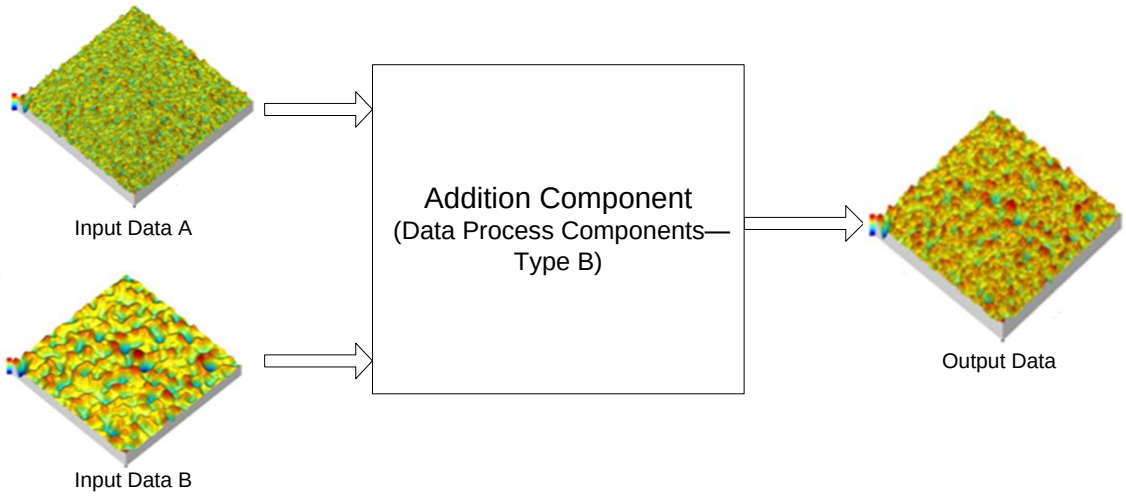


Figure 5.2: Example of Data Process Components—Type B.

3) Type C: One Data Set In and One Display Control Out

Sometimes the input and the output of an operation are not in the same format. In the surface characterisation system for example, surface data should be always treated as raw

data to be processed, but the output data is not always surface data. For example, the parameter calculation operation to compute various parameters is based on a surface data which has already been processed by previous operations, so the output of such an operation is an array of parameter values. Obviously, these parameter values are quite different from a standard surface data set, and thus they cannot be displayed by the general graphical data display component. The display of these analysis results should be provided by this operation component. An extra display control which is designed to display the analysis results should be returned from this component. In Figure 5.3, a parameters list control is implemented in the proposed system as the output of parameters calculation components. Thus, the outputs of these components are display controls.

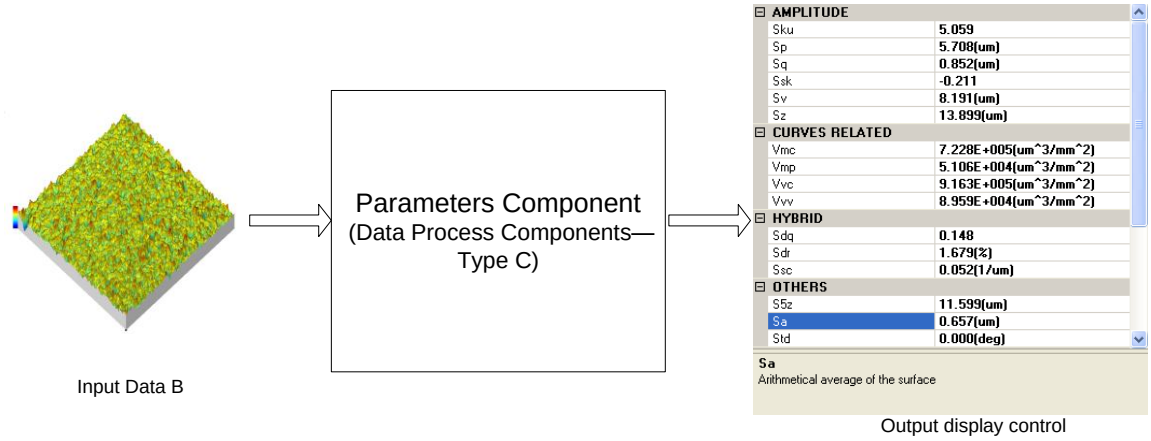


Figure 5.3: Example of Data Process Components—Type C.

Some particular analysis modules just do operations on the input data and do not output any surface data or parameter calculation. For example, the bearing curve is the cumulative probability density of the height of a surface and can be calculated by integrating the profile trace. The output of bearing curve analysis is a profile, which is not a surface data set. In order to plot such a profile, the component has to create a display control for the system framework. In this category, the analysis result of a component is not surface data which can be displayed in the system framework via a display component. So the analysis result should be output to a new control which is provided by a dedicated component and the system framework displays this new control.

5.2 Component Interface Design of the Proposed System

As a specific functional component type, data process components have to realise the *SSObject* interface as well as data access components.

Data process components are the most significant and complex functional components in any surface characterisation system including the proposed system. The interface design of data process components is often supposed to be much more difficult than other components. Since the interface can only reflect the data process components with the same attributes, it is impossible to design a unique interface that can be used by all data process components. Instead, it is necessary to design at least three main interfaces for the three categories of data process components as proposed in the previous section.

5.2.1 Interface of Verification Operations

Functional components are fundamental elements of the characterisation system, they provide different types of functions for surface analysis, and they play the role of function library. They are not the same as the actual analysis actions. Since this characterisation system is designed accord with GPS standards, most of analysis actions are standard verification operations. Besides, there are still other actions deriving from the software system requirements. For example, opening a surface file is an essential action before any data analysis is carried out, but clearly this is not included in any GPS standards as the standards only concern themselves the analysis techniques and methods. This is the essential distinction between theory and practice in terms of system development. In essence, opening a surface file is also a verification operation which derives from metrology practice. Therefore, an actual surface verification operator is certainly composed of many of these types of operations.

In the developed characterisation system, commands are designed to represent those verification operations. When the system receives a request from users, relevant commands will be formed and executed. Almost every command is related to at least one functional component by which the actual data processing work is completed. In fact, most commands only associate with one functional component except for compound commands. The compound commands consist of a series of sequential commands, and it is regarded as the representative of verification operators. As there are many kinds of

functional components, commands are also required to be classified according to the types of associated functional components. To invoke these commands consistently, an interface with common properties and methods is needed and has been developed here. No matter what command is created when the user request comes, the system can activate the command in a unique way. *ISSCommand* is such an interface for all commands, and the definition is listed in Appendix D. It contains four properties and two methods. The instructions have been listed in Table 5.1.

Table 5.1: Description of member variables and methods of interface ISSCommand.

Types	Name	Description
Variable	Tag	Link to the tag recorded in the user action list
	Name	The name of the command
	DefaultArgs	Specifies whether the default arguments are used.
	ArgList	Records all arguments of the command
Method	Execute()	Performs the command with appropriate arguments
	ModifyArgument()	Changes the arguments of the command

The implementations of different commands cannot be the same; consequently they are classified according to the functional component they are associated with. Since there are three main types of functional components in the whole system, the number of command types is no less than three. In fact, there are six types of commands in the developed characterisation system. As data process components are more complex than the other two types of functional components, three different types of command are defined according to the categorisation of the data process components. In addition, one more command is defined for compound commands which are the composition of a series of sequenced commands. Figure 5.4 shows the inheritance hierarchy structure of commands for the developed system.

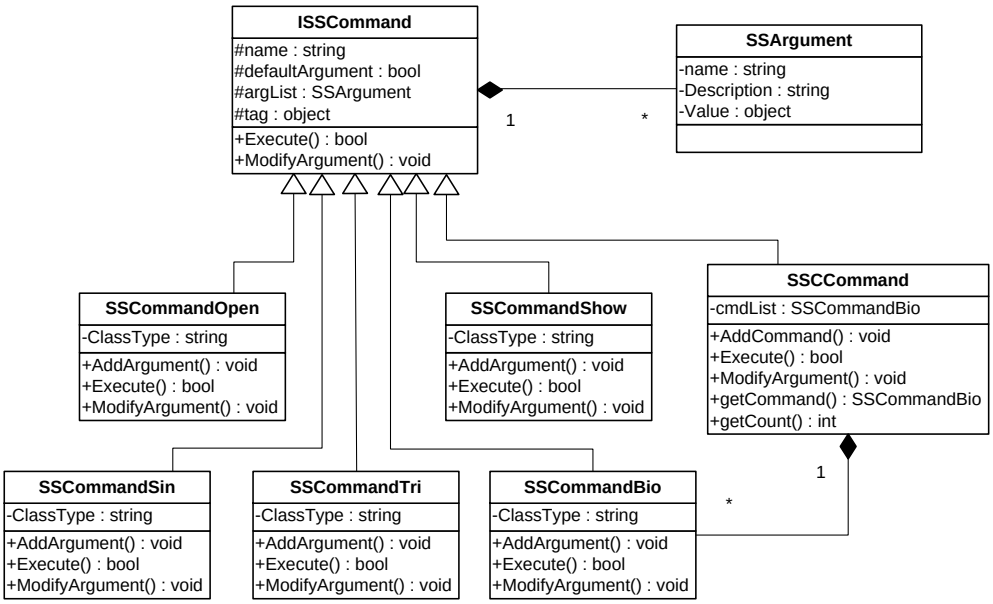


Figure 5.4: Internal command class inheritance hierarchy diagram of developed system.

SSCommandAccess is the command type which deals with data access components, and it will be created when users try to open or save a surface file. *SSCommandShow* is defined to meet the requirements of data display requests. Normally, this type of command will be automatically created after all other types of commands because display components are always used to present the results from others functional components. *SSCommandSin*, *SSCommandBio* and *SSCommandTri* are designed for commands that are associated with data process components. According to the I/O properties of data process components, these three types of commands have one, two or three surface data sets as their operands. They are related with Type C, Type A and Type B of the data process components respectively. *SSCCommand* is a compound command type, and it is not related to any functional component directly. Instead, it is a command list of *SSCommandBio* commands. Therefore, the SSCCommand is related to the implementation of verification operators in this characterisation system. The relationship between verification operators and operations is the same as that between simple commands and compound commands. Figure 5.5 shows the relationships between them.

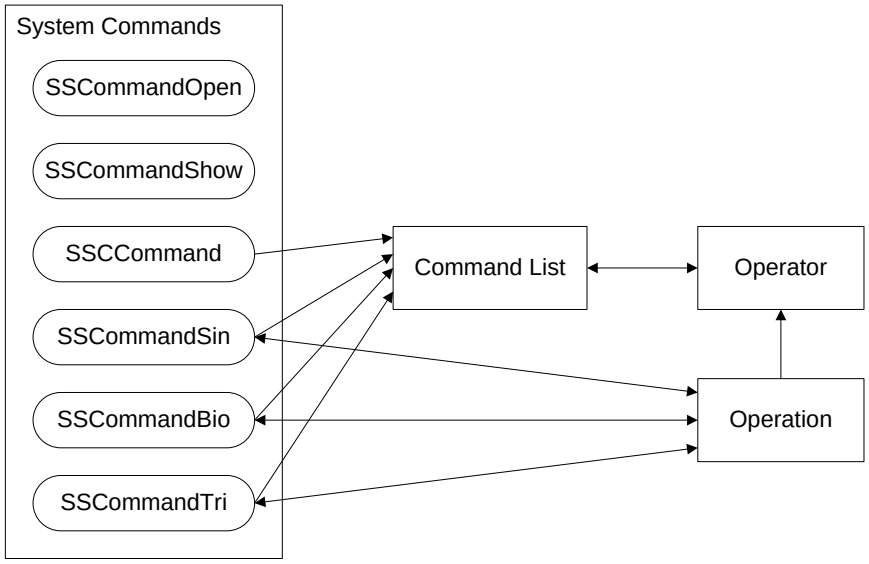


Figure 5.5: Relationship between commands and surface verification operators.

5.2.2 Interface for All Data Process Components

Although the interface for each type of data process components is essentially different, there are common attributes amongst them. In fact, the methods or operations of all data process components are very similar. The most important method is the execution of the algorithm, and this is similar to reading or writing surface data for data access components. Unfortunately, the execution method cannot be regarded as a common operation as its parameters will vary for different subtypes of data process components.

However, there is one method that is the same for all types of data process component. This is the parameter setting, and almost every data process component has its own parameters which are related to particular internal algorithms. Generally, these parameters have great impact on the analysis result. To ensure the execution of internal algorithms, it is necessary to set up all parameters before starting the execution. Sometimes, one data process component is expected to execute many times by changing one or more parameters setting to examine the effects caused by those parameters. Therefore, setting algorithm parameters is no doubt an important operation for a data process component.

Since the parameter setting is a common operation for all data process component, a lower layer interface needs to be developed prior to the interface design of every concrete

subtype. In practise, this interface is named as *SSAnalysis*, and it acts as the interlayer interface between the root interface *SSObject* and the interface of every subtype. It is defined as:

```
public interface SSAnalysis:SSObject
    {
        void ModifyArguments(ISSCommand command);
    }
```

ModifyArguments() is the only method in the interface *SSAnalysis*, and its aim is to invoke the parameter setting action instead of transferring parameters into the component. In operation, the parameters are kept inside the command rather than the functional component. Although two commands may invoke the same functional component, they are different because they transfer different parameters into the component. Figure 5.6 illustrates the relationship between commands and functional components. Therefore, the argument of the method *ModifyArgument()* is only the command which will invoke this functional component.

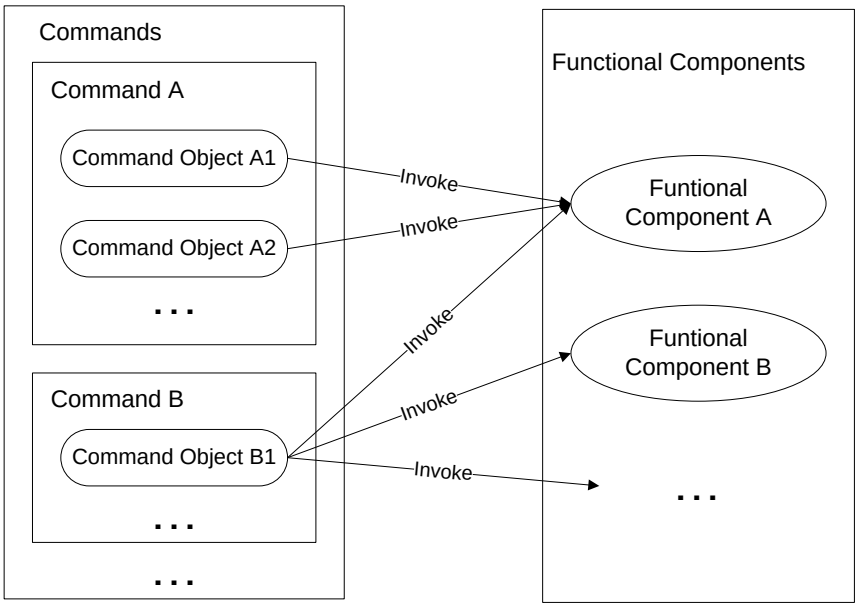


Figure 5.6:Relationship between commands and system functional components.

One possible way to set up parameters for a command is to create a parameter dialog inside the method *ModifyArgument()*. The dialog is defined in the functional component itself on which all parameters associated with the inner algorithms are selected or set. These parameters are then saved in the argument list of the command. For instance, the

code fragments of *ModifyArgument()* in the *FeatureParameters* component are listed in Appendix E.

5.2.3 Interface for Subtypes of Data Process Components

As the root interface of all data process components, *SSAnalysis* is supposed to be derived by every interface for subtypes. Figure 5.7 is the inheritance hierarchy diagram of the data process components.

SSAnalysisA, *SSAnalysisB* and *SSAnalysisC* are the interfaces for three subtypes of data process components respectively. All of them just have one method "Execute", which is to perform the analysis algorithm inside the functional components. With the additional method *ModifyArguments()* from the parent interface *SSAnalysis*, they are the same interface in essence. However, the arguments of "Execute" are different and they are dependent on I/O properties of data process components. This is the reason why three interfaces are designed. The definitions of these three interfaces are listed in Appendix F.

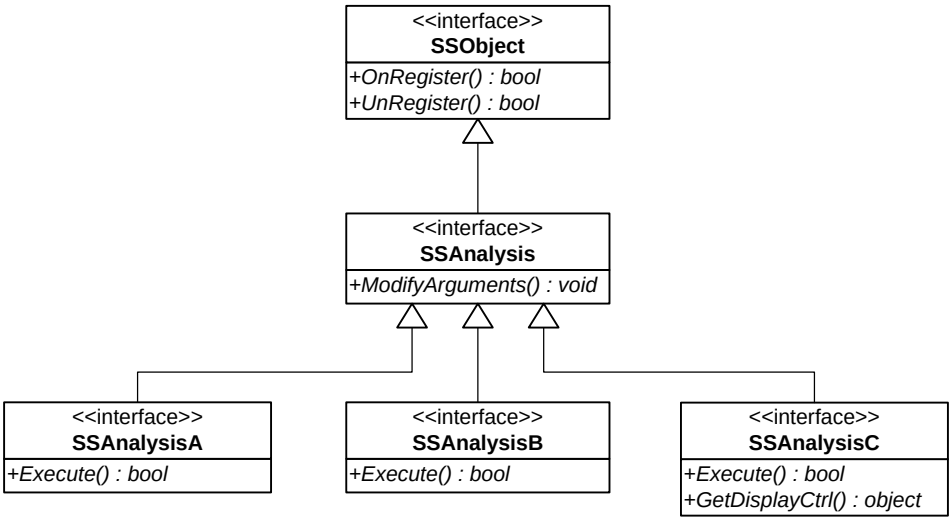


Figure 5.7: Class inheritance hierarchy diagram of data process components.

Every interface has its own *Execute()* method with different arguments. The arguments are composed of input data, output data and parameters for algorithms within the component. In addition, a flag argument *defaultArgs* helps indicate whether the default arguments are used for algorithms or not. All the instructions of these arguments are listed in Table 5.2.

Table 5.2: Description of arguments that occurs in SSAnalysisA, SSAnalysisB and SSAnalysisC.

Argument	Type	Description
pSrc	SSData	Input surface data
pSrc2	SSData	Another input surface data
pOut	SSData	Output surface data
defaultArgs	bool	Whether the default arguments are used
argList	List<SSArgument>	Arguments List(Dynamic linked list)

It is worth noting that there is one more method within *SSAnalysisC*. As the Type C data process component do not return any surface data as the analysis results, the display component is unable to be invoked to display its results. Hence, *GetDisplayCtrl()* is such a method which returns a specific control for the analysis results. Meanwhile, the command *SSCommandSin* invokes this type of function in a different way. It should use the returned display control rather than any standard data display control to present the analysis result.

5.3 Case Studies: Geometrical Analysis Component

Surface roughness analysis represents an attempt to quantify how surface topography plays an important role in determining how a real object/surface interacts with its environment. A series of parameters have been defined by ISO standards to represent the corresponding surface texture in a quantitative way. Although roughness analysis is accepted as an effective way to predict the performance of a surface, it is ineffective in evaluating a surface such as structured surfaces whose functions are highly dependent on the geometrical features of the structures rather than the surface roughness [4, 67]. Similarly, the deviations of these real geometrical features from their ideal form should be quantified for the purpose of quality evaluation.

At present, there are no ideal instruments for the measurement of these tiny geometrical features on a surface directly. However, they can be measured indirectly by using surface instruments, and then be extracted using mathematic algorithms. A pattern analysis method which is described in ISO 25178-2 [12] is proposed to extract boundaries of geometrical features. As these boundaries are sets of discrete points, it is not convenient to represent the characteristics of geometrical features directly. A further segmentation

based on these boundaries is imperative for the recognition, extraction and construction of the geometrical features. Realising the recognition and construction of predefined geometrical features based on these primitive segments is essential for the further characteristics analysis. Common geometrical features to be analysed include circle, slots, edges and combinations of these. There are two types of characteristics defined in GPS standards, intrinsic and situation characteristics [39].

5.3.1 Surface Segmentation

Surface segmentation is an essential procedure for the geometrical features analysis. The edge information of the microstructures on a surface cannot be acquired by measurement directly as a surface data is a continuous cloud of data points. Although many techniques have been implemented for edge detection, they are mainly used for image processing and analysis. In the field of surface metrology, a pattern analysis method [12, 110] based on morphological analysis has been developed for analysing microstructure surfaces. This segmentation method can be used to determine regions of a scale limited surface which define the scale limited features. Regions consisting of hills and dales on the scale limited surface are separated from each other. The boundaries between hills are defined as course lines, while those between dales are defined as ridge lines. It has been demonstrated that ridge and course lines are maximum uphill and downhill paths emanating from saddle points and terminating at peaks and pits [12]. Therefore, a scale limited surface can be organised by networks of critical points: peaks, pits and saddles points in addition to critical lines: ridge lines, course lines. Unfortunately the result of the segmentation is initially disappointing as the surface is separated into a large number of insignificant tiny segments. Thus it is necessary to merge these tiny segments to a few large significant segments. Height and area pruning methods applied to the surface networks with appropriate thresholds can be implemented to obtain a more reasonable segmentation result.

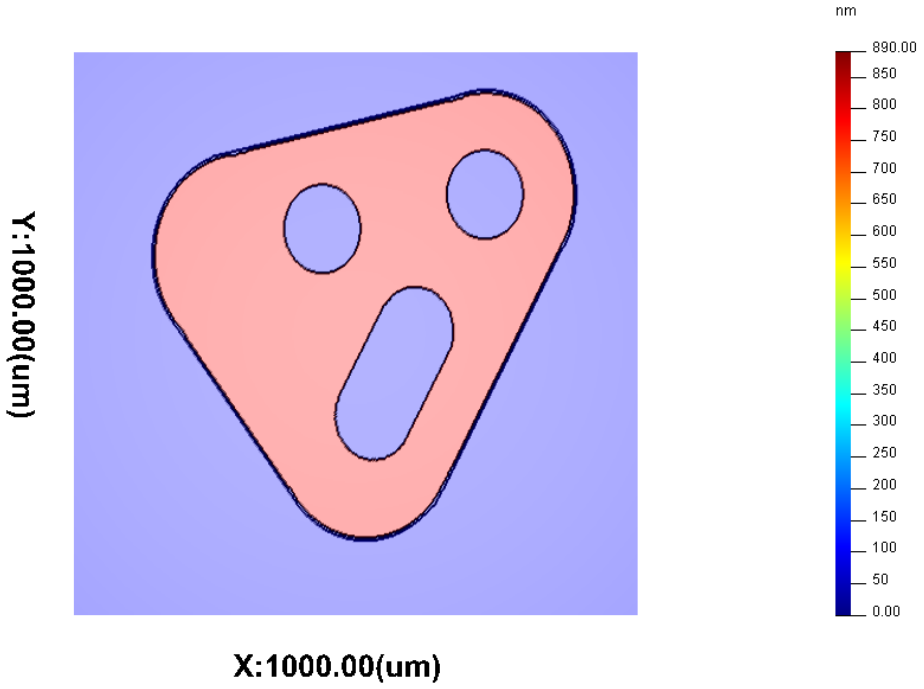


Figure 5.8 Demonstration of surface segmentation

As the magnitude of altitude differences between regions of microstructures and other regions is usually much greater than that of roughness, the boundaries can be extracted very effectively by further processing of the surface segmentation analysis. Figure 5.8 illustrates the result of surface segmentation of a surface data set, and it shows that the boundary curves of geometrical features have been extracted effectively. In fact, the reliability and accuracy of boundaries are not only determined by the threshold used for segmentation, but also the clarity of microstructures. If the microstructures on a surface suffer ambiguity that may be caused by manufacture or measurement, the boundaries of the feature obtained by surface segmentation are indicative of this.

5.3.2 Boundary curve approximation

The boundaries of geometrical features usually cannot be represented by only one segment, so the segmentation of boundary curves is necessary before feature recognition and shape analysis. Many techniques of planar curve segmentation have been proposed in the literature. For instance, Biswajit Sarkar et al.(2003) use genetic algorithms to realise the approximation of planar curves [111]. Wu Chih Hu (2005) has developed a

methodology for planar curve segmentation with line segments and conic arcs based on the types of breakpoints [112]. Here boundary curves are segmented into lines and circular arcs based on curvature variation, and then LS methods are applied for the fitting. Figure 5.9 is a boundary curve which is the result from the surface segmentation of the image shown in Figure 5.8.

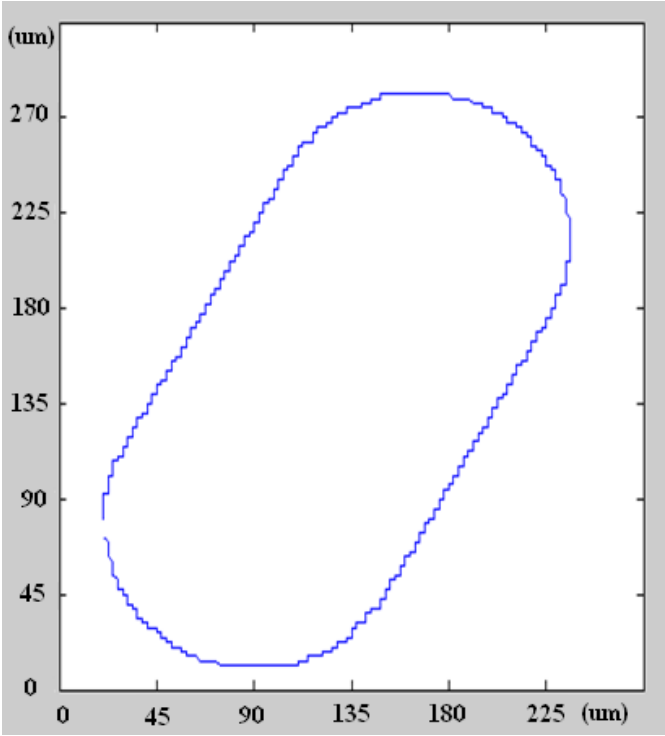


Figure 5.9: One boundary curve of a geometrical feature.

Curvature as a significant difference between lines and circular arcs is selected to establish the junction points which separate the whole boundary curve into primitive segments. To do this, the curvature of each point on the boundary needs to be calculated. Practically, curvature calculation in accordance with the curvature formula cannot be applied here directly, as the boundary curves are effectively non smooth curves and the noise has a great impact on the result. A more successful approach to obtain the curvature is to calculate the curve radius which is the reciprocal of curvature.

The calculation procedure of radius of curvature is described as follows. A boundary curve can be defined using the representation $(x(i), y(i))\ \ (0 \le i < \mathrm{N})$, where $x(i)$ and $y(i)$ are two functions of the index variable i, and N is the number of points. To

calculate the radius of curvature of point $P\left(x(i), y(i)\right)$, sufficient adjacent points should be taken into account. A possible choice is to take k points from each side of point P, so the total number of points used for circle fitting is $2k+1$. If the boundary curve is not closed, this number is less than $2k+1$ for those points at the end sides of the curve. The parameter k determines the magnitude of the variation of curvature among adjacent points. Figure 5.10 illustrates the curvature of all points on the selected boundary curve. After the curvature calculation, the junctions can be found out by taking the derivative of the curvature along the curve and locating the spikes on the derivative as shown in Figure 5.11.

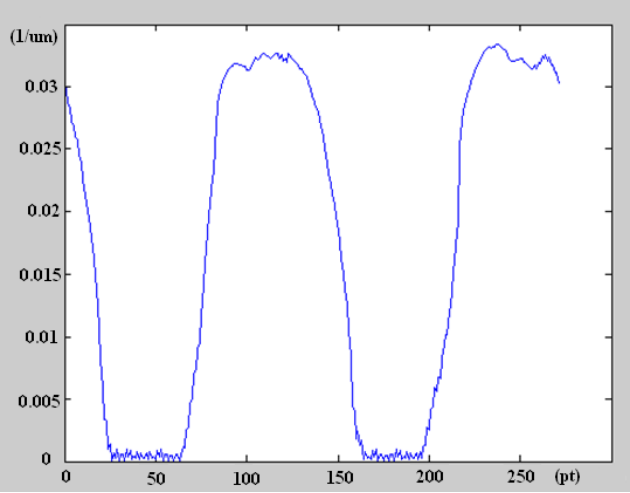


Figure 5.10: Curvature of each point on the boundary curve.

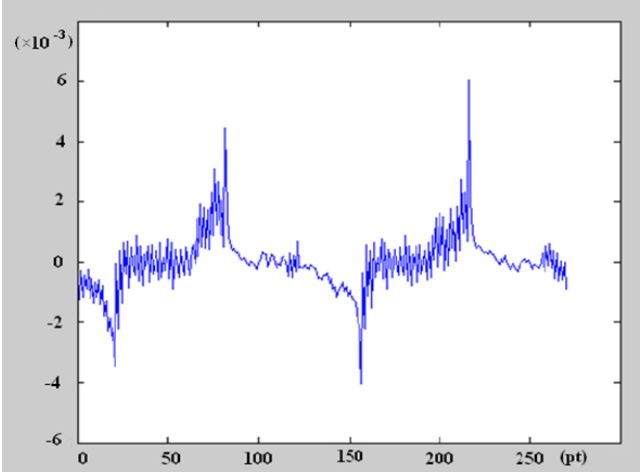


Figure 5.11: Derivative of curvature along the boundary curve.

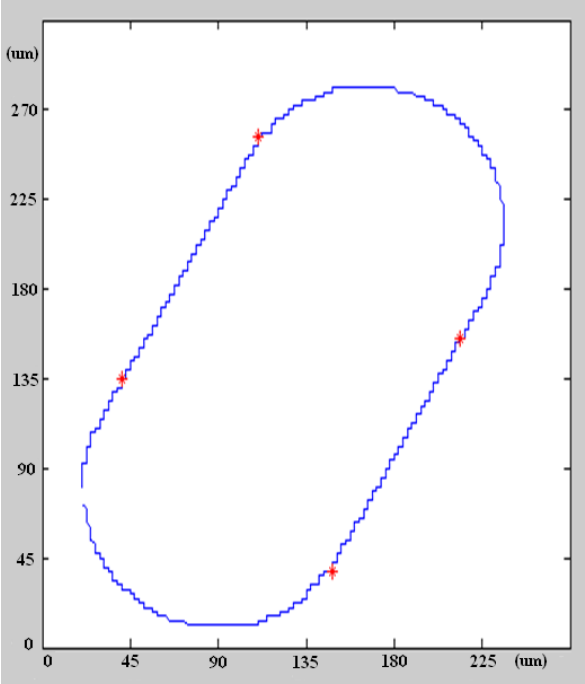


Figure 5.12: Extracted joint points of the boundary curve.

According to the calculated junction points as shown in Figure 5.12, a boundary curve can be divided into many short curve segments. A segment contains all the points between two adjacent junction points. However, before the approximation of a segment, its objective function is required. In other words, whether the segment is a line or circular arc needs to be determined in advance. The curvature of a point on a line, which is zero theoretically, should be very close to zero. A threshold can be chosen to simplify this determination. If the curvature is over this threshold, the segment is a circular arc; otherwise, it is defined as a line. Thus, all segments that can be divided by junction points can be fitted by using a least-squares method, and they can be used for the recognition and characteristics analysis of geometrical features.

5.3.3 Geometrical features construction

Various geometrical features can be constructed by combining lines and circular arcs with different quantities and sequences. For simplicity, circles, triangles, quadrilaterals, slots and keyholes are considered in this section. Suppose that C indicates a circular arc and L a line. These geometrical features could be expressed by a sequence of circular arcs and lines.

Table 5.3: Combination patterns for common geometrical features.

Geometrical features	Expression
Circle	C
Triangle	LLL
Quadrilateral	LLLL
Slot	CLCL(LCLC)
Keyhole	CLLL(LCLL,LLCL,LLLC)

Generally, each boundary curve is a combination of line segments and circular arcs after the processing of the boundary approximation. A boundary curve can be identified as a form of geometrical feature by comparing their segment expressions. The boundary curve showed in Figure 5.12 is recognised as a slot whose expression is 'CLCL'. Nevertheless, some boundary curves cannot be recognised as they are either ambiguous or beyond the scope of this example. The geometrical features of interest which have been predefined in Table 5.3 can be constructed from boundary curves. Furthermore, the characteristics of these geometrical features can be quantified for the surface evaluation and verification.

Characteristic analysis of geometrical features includes their intrinsic characteristics as well as the situation characteristics between two features. For instance, centre point and radius are intrinsic characteristics of a circle feature. However, the distance between two centre points can be thought of as a situation characteristic which reflects the relationship of two geometrical features. In ISO/TS 17450-1 situation characteristics, which are length and angle, can be separated into location characteristics and orientation characteristics [39]. In this component, both the intrinsic characteristics and situation characteristics are calculated and listed according to its feature type as shown in Figure 5.13.

ParaName	Type	Value	Valid	Nominal	Lower	Upper	Unit
S2_A1|Radius	Radius	67.8600...	F	67	-0.5	0.5	um
C4_C0|Radius	Radius	69.0999...	P	69	-0.5	0.5	um
C3_C0|Radius	Radius	69.0999...	P	69	-0.5	0.5	um
S2_L0|Length	Length	139.240...	F	130	-5	5	um
S2_A1|Angle	Angle	-183.77...	F	-190	-5	5	deg
S2_L2|Length	Length	144.119...	P	140	-5	5	um
S2_A3|Radius	Radius	67.7300...	F	67	-0.5	0.5	um

Figure 5.13: Characteristics list of geometrical features.

5.3.4 Interface Implementation for Geometrical Analysis Component

Initially, it is essential to determine what type this geometrical feature component belongs to. The analysis process is based on only one surface data set which is regarded as the input data for the component, while the analysis results are geometrical features (such as circles, slots) and their dimensions and size. Obviously, this component is a type C data process component according to the I/O properties. Therefore, the interface *SSAnalysisC* and all ancestor interfaces need to be implemented. This means that the component class has to override all the methods it derived from these interfaces.

- Derived from *SSObject*

There are two methods in *SSObject*: *OnRegister()* and *UnRegister()*. The former is called when adding the component to the system framework, while the latter is called when removing it. In the *OnRegister()* method, three processes are completed.

a) Add the component information to the system component list:

```
SSCommon.AddAssembly("GeometricalFeatures",
"DataAnalysis.GeometricalFeatures",
System.Reflection.Assembly.GetExecutingAssembly().Location,
SSAssemblyType.DATAANALYSIS);
```

b) Add a new command to the system command list, and the command is associated with this component. Meanwhile, the default arguments are also configured to a command.

```
SSCommandSin cmd = new SSCommandSin();
cmd.Name = "GeometricalFeatures";
cmd.ClassType = "GeometricalFeatures";

cmd.DefaultArgs = true;
```

c) Add a menu item for this component, users can invoke the command by clicking this menu item.

```
SSCommon.AddMenuItem("&DataView;GeometricalFeatures ",
"GeometricalFeatures");
```

Conversely, there are also three elements in the Unregister correspondingly.

a) remove a menu Item

```
SSCommon.RemoveMenuItem("&DataView;GeometricalFeatures");
```

b) Remove a command

```
SSCommon.RemoveCommand("GeometricalFeatures");
```

c) Remove component information

```
SSCommon.RemoveAssembly("GeometricalFeatures",SSAssemblyType.DATAANALYS
IS);
```

- Derived from *SSAnalysis*:

In the interface SSAnalysis, only one method needs to be overriden.

ModifyArguments is to update the parameters setting in the command. A parameters dialog is created, and it is initialised by the last set values saved in command. After modifying the parameters, all values are saved to the command as shown in following code fragment.

```
ParameterSetting ps = new ParameterSetting();

…//Read parameters values from the command;

ps.ShowDialog();

…//Save parameters values to the command;
```

- Derived from SSAnalysisC:

Unlike the other two types of data process component, there are two methods in *SSAnalysisC*. The method *Execute()* analyses the input surface data to obtain the results and send the result to the display control, while the *GetDisplayCtrl()* method returns the display control. These two methods are invoked by the command *GeometricalFeature* which is added to the system framework by this component. As the implementation of the algorithm is very complex and lengthy, the code will not be presented here.

By overriding these methods in the component class, they are invoked correctly at the correct time. The developers of this component do not need to worry about the communication between it and the system framework, as these methods are predefined to complete presupposed work.

5.4 Summary

This chapter emphasises the design and implementation of data process components in the developed system software framework. As the critical functional components for surface characterisation, data process components are much more complex and diversified than the other two types of components. Therefore, the categorisation of them is discussed first. Following this the interfaces of different types of data process components are defined respectively. As an example the design and implementation of the geometrical analysis component is finally elaborated.

Chapter 6 Implementation of Surface Visualisation Components

6.1 Computer Graphics

Computer graphics is one of the most exciting aspects of computing technology, and it is core to any work that uses computation to create or modify images. The computer is utilised to generate visual images synthetically and to integrate or alter visual and spatial information sampled from the real world [113]. The development of computer graphics has made computers easier to interact with and greatly enhances data facilitating deeper understanding and interpretation. Computer graphics techniques are primarily used to present to the end user surface data which is captured from the measurement instrument. Figure 6.1 presents an overview of the graphics system. The original data is imported by the Input Devices, and then be processed by the CPU and GPU (Graphics Processing Unit) in sequence. After the processing, the data is converted into the image which is saved in frame buffer, and then the image is transferred to output devices.

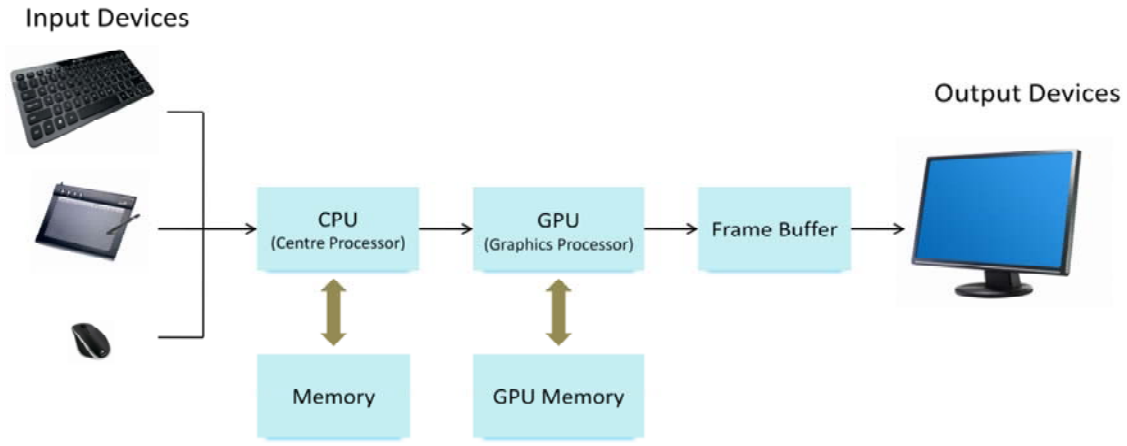


Figure 6.1: Computer graphics system principle.

The uptake of high level computer graphics is widespread, and it includes almost everything on computer that is not text or sound. As technology has improved, 3D computer graphics have become more common, but to date 2D computer graphics are still widely used. Currently, many powerful tools have been developed to achieve data visualisation. One common and convenient way is to program with a graphics API (Application Programming Interface) that supports the necessary modelling and does

most of detailed work of rending the scene. There are a number of graphics APIs available, both Direct3D and OpenGL are commonly used [114].

Direct3D and OpenGL are competing API both can be used in an application to render computer graphics, and they are both implemented within the display driver. Modern GPU may implement a particular version of one or both of these APIs. On the one hand, DirectX is a collection of APIs for handling tasks related to multimedia, especially game programming and video on Microsoft platforms. The name DirectX is coined as a shorthand term for all of the APIs that begin with Direct, such as Direct3D, DirectDraw, DirectMusic, DirectSound, and so forth. Direct3D is usually used by software applications for visualisation and graphics task. As the most widely publicised component of DirectX, the term Direct3D is often used interchangeably with DirectX. On the other hand, OpenGL is an open standard Application Programming Interface that provides a number of functions for rendering of 2D and 3D graphics, and is available on most modern operating systems including but not limited to Windows, Mac OS and Linux.

The differences between Direct3D and OpenGL are not only their platform and proprietary. In general, Direct3D is designed to virtualise 3D hardware interfaces. Direct3D frees the programmer from accommodating the graphics hardware. While OpenGL is designed to be a 3D hardware accelerated rendering system that may be emulated in software[115]. They are fundamentally designed as a product of two separate modes of thought. As such, there are functional differences in how the two APIs work. Direct3D expects the application to manage hardware resources that is done by implementations in OpenGL. Thus, OpenGL decreases difficulty in developing for the API, but increases the complexity of creating an implementation. With Direct3D, the implementation is simpler and the developers have the flexibility to allocate resources in the most efficient way possible for their applications.

As the present surface characterisation system is developed on Windows system, both Direct3D and OpenGL are eligible for the surface topography rendering. The developers of further data display components can select any one of them according to their

preference. In this chapter, the development of data display components with OpenGL is elaborated.

6.2 Computer Graphics Programming—OpenGL

OpenGL is a powerful software interface used to produce high-quality computer generated images and interactive applications using 2D and 3D objects and colour bitmaps and images [116]. In recent years, it has become a worldwide standard for 3D computer graphics programming. Although OpenGL was originally developed by SGI for its high-end IRIX workstations, it has grown in scope enormously and now its development is led by a group consisting of industry leaders such as 3DLabs, HP, Evans & Sutherland, IBM, Inter, Microsoft, NVIDIA and SGI itself. This group is called the ARB (OpenGL Architecture Review Board) [117]. In this section, some basic functions as applied to the present system are elaborated.

6.2.1 Modelling

OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms. It consists of about 150 distinct commands that are use to specify the objects and operations needed to produce interactive 3D applications. OpenGL does not provide high-level commands for describing models of 3D objects such as automobiles, parts of the body, airplanes, or molecules [103, 118]. With OpenGL, the desired model is built up from a small set of geometric primitives: points, lines, and polygons.

Primitives are defined by a group of one or more vertices. A vertex defines a point, an endpoint of a line, or a corner of a polygon where two edges meet. Data (consisting of vertex coordinates, colours, normal vectors, texture coordinates, and edge flags) is associated with a vertex, and each vertex and its associated data are processed independently, in order, and in the same way. Primitives are drawn subject to several selectable modes. These modes are independent of each other, namely, the setting of one mode would not affect that of others even when they may interact to determine the final image in the frame buffer. Figure 6.2 shows an example of drawing a polygon and five points.


```
glBegin (GL_POLYGON);
    glVertex2f(0.0, 0.0);
    glVertex2f(0.0, 2.0);
    glVertex2f(2.0, 2.0);
    glVertex2f(3.0, 1.0);
    glVertex2f(2.0, 0.0);
glEnd();
```

```
glBegin (GL_POINTS);
    glVertex2f(0.0, 0.0);
    glVertex2f(0.0, 2.0);
    glVertex2f(2.0, 2.0);
    glVertex2f(3.0, 1.0);
    glVertex2f(2.0, 0.0);
glEnd();
```

Figure 6.2: Comparison of drawing a pentagon and five points.

6.2.2 Transformation

6.2.2.1 The Viewing Transformation

Viewing transformation is analogous to positioning and aiming a camera, and it is specified with *gluLookAt()* [103, 118]. The arguments for this function indicate where the camera is placed, where it is aimed, and which direction is up. If the viewing transformation is not set apparently, the camera has a default position and orientation. The camera is placed at the origin, points down the negative z-axis, and the up vector is (0, 1, 0). It is the same meaning as call:

gluLookAt(0.0, 0.0, 0.0, 0.0, 0.0, -1.0, 0.0, 1.0, 0.0);

However, this may fail to specify the viewing transformation. The reason for this is that the final viewing transformation is not only dependent on the matrix specified by *gluLookAt(),* but also the current matrix. It is necessary to set the current matrix to the identity matrix with *glLoadIdentity().*

6.2.2.2 The Modelling Transformation

The modelling transformation is to position and orient the model rather than the camera. The model can be rotated, translated and scaled. These three operations are realized by *glRotate*(), glTranslate*()* and *glScale*()* respectively [103, 118, 119]. Usually some combinations of these are used to realize modelling transformation.

It is well known that both viewing transformation and modelling transformation are available to obtain the expected view of the mode. Sometimes it is easier to think about the effects of transformations one way rather than the other. However, it does not make sense to separate these effects, and they are combined together to take effect simultaneously.

6.2.2.3 The Projection Transformation

Specifying the projection transformation is similar to choosing a lens for a camera. The purpose of the projection transformation is to define a viewing volume, which determines how an object is projected onto the screen and defines which objects or portions of objects are clipped out of the final image. There are two types of projection: perspective and orthographic [118].

- Perspective projection

One important characteristic of perspective projection is foreshortening. It means that the farther an object is from the camera, the smaller it appears in the final image. This method of projection is commonly used for animation, visual simulation, and any other applications that strive for some degree of realism because it is similar to how a camera works. (As shown in

Figure 6.3)

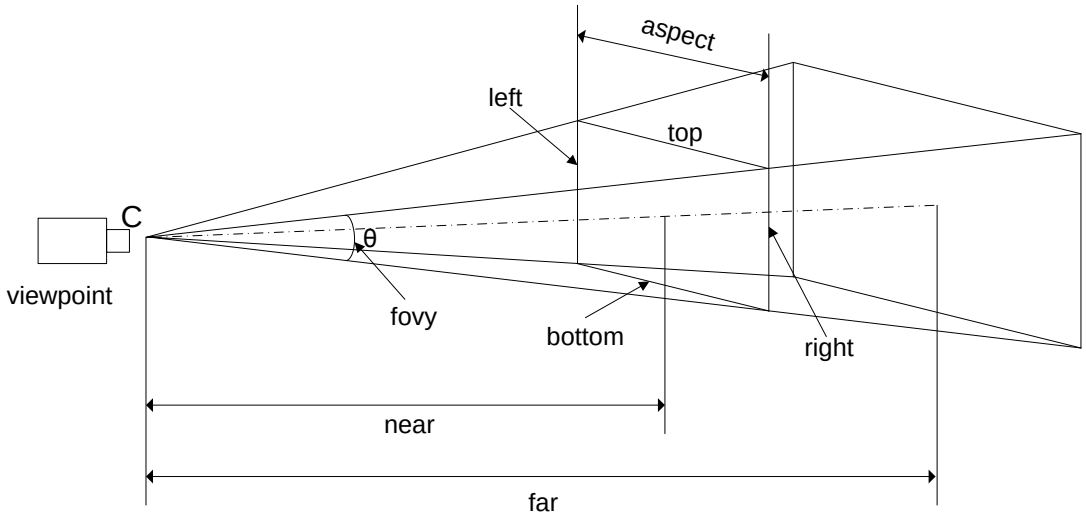


Figure 6.3: Perspective viewing volume.

- Orthographic projection

With orthographic projection, the viewing volume is a rectangular parallelepiped as shown in Figure 6.4. The size of the viewing volume does not change from one end to the other, thus an object would be the same size no matter how far it is from the camera. This type of projection is used for applications such as creating architectural blueprints and computer aided design. This is because the actual sizes of objects and angles between them are crucial and need to be maintained.

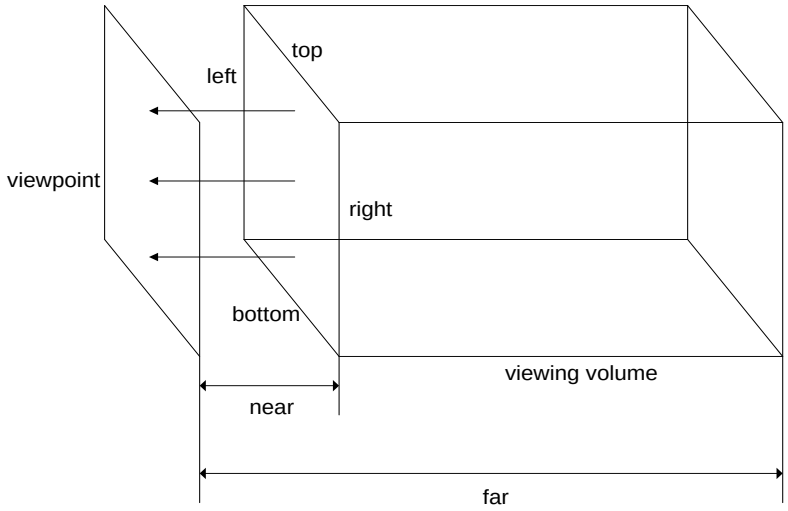


Figure 6.4: Orthographic viewing volume.

With no other transformations, the direction of projection is parallel to the z-axis and the viewpoint faces toward the negative z-axis. In other words, the values of far and near are used as negative z values if these planes are in front of the viewpoint, and positive z values if they are behind the viewpoint.

6.2.2.4 The Viewport Transformation

The viewport transformation is analogous to choosing the size of the developed photograph [119]. Therefore, the viewport is the rectangular region of the window where the final image is drawn. The function *glViewport()* is used to specify the viewport:

Void glViewport(GLint x, GLint y, GLsizei width, GLsizei height);

The (x,y) specifies the lower-left corner of the viewport, and width and height are the size of the viewport rectangle. The aspect ratio of a viewport should be the same as that of

viewing volume. Otherwise, the final image will be distorted when mapped to the viewport.

6.2.3 Colour

On a computer screen, the colour of each pixel is formed by emitting different amounts of red, green and blue light. In OpenGL, the colour information of each vertex is stored either in RGBA mode or Colour-index mode [118, 119].

- RGBA mode

In RGBA mode, the values of R, G and B are required to be specified. The R, G and B values can range from 0.0(none) to 1.0(maximum). For example, the set (0.0, 0.0, 1.0) represents the brightest possible blue. Any types of colour can be composed of appropriate R, G and B values. Function *glColor*()* is used to set a current colour.

- Colour-index mode

Unlike RGBA mode, there is only one single number to specify the colour. This value is an index which indicates an entry in a table that defines a particular set of R, G and B values. Such a table is called a colour map. However, colour maps are controlled by the window system, and there are no OpenGL functions to do this. There would also be a great deal of variation among the different graphics hardware platforms. Function *glIndex*()* is used to set a colour index as the current colour index.

6.2.4 Lighting

In the real world, the colour of an object depends on the distribution of photon energies that arrive and trigger cone cells. Those photons come from a light source or combination of sources, some of which are absorbed and some of which are reflected by the surface. OpenGL approximates light and lighting as if light can be broken into red, green and blue components. The light in a scene comes from several light sources that can be turned on and off individually. In the OpenGL model, the light source has an effect only when there are surfaces that absorb and reflect light. An object might emit its own light, scatter some incoming light in all directions, and reflect some portion of the incoming light in a preferential direction [103, 118].

6.2.5 Other Functions of OpenGL

Except for the aforementioned features, OpenGL has many other advanced features such as Blending, Antialiasing, Texture mapping, Frame buffer, and so forth [103, 118, 119]. With all these features, it is possible to make the final image appear close to the real world. In this section, other functions of OpenGL would not be elaborated as they are barely used in the present display components. It is sufficient to create quality surface topography images with previous features discussed in the above section.

6.3 Component Interface Design

The aim of data display components is to present surface data on the computer screen. Similar to the case that one idea can be expressed in different languages, data can be presented in different forms such as line chart, pie, histogram, and so forth. Hence, it is possible to present surface data in different ways which may be used for various purposes. For instance, a dynamic 3D view is more suitable to revealing the surface spatially, while a top view is better for comparing surfaces intuitively.

Surface data is a set of altitude values with certain ordinates. It is not easy to paint the surface with these discrete values. Although surface data for displaying is not processed any more, there are still some calculations such as coordinate transform, render, lighting, and so on which need to be executed. Therefore, it is not simple to implement a satisfactory and perfect data display component. However, no matter how complex the data display component is, the interface needs to be quite simple. Surface data transfer is the only interaction between the system framework and data display components. All the system framework needs to do is to send surface data to the data display component, and there is only one method in the interface. Following is the definition of this interface:

```
public interface SSDisplayIO:SSObject
{
    bool SetData(object pData, bool bCopy);
 }
```

The interface SSDisplayIO is derived from SSObject due to the fact that any data display components belongs to functional components. “SetData” is invoked by the system framework which transfers the processed surface data to the data display component. The

first argument is the surface data which would be presented, while the second argument is used to determine the way in which the surface data is transferred.

In general, the interface of the data display component is simpler than the other two types of functional components. It is not hard to implement the interface. Instead, developers can then invest more effort in creating the surface graphics.

6.4 Three-Dimensional Display Component of the developed Surface Characterisation System

Nowadays, surface areal analysis is becoming more and more significant, and it is widely used to evaluate surface quality. Accordantly, three-dimensional display is indispensable for a surface characterisation system as it can present a surface spatially. Users can develop intuitive knowledge about the surface by reproducing a surface in 3D on the screen, and the difference between pre and post processing can be presented naturally. Furthermore, users can observe more details from different perspective by rotating, translating and scaling the surface. This section details the implementation of the 3D display component with OpenGL technology.

6.4.1 Coordinate System

A surface is a three dimensional geometry, and it is presented on a computer screen which is absolutely a 2D coordinate system. The transform is achieved by creating the 3D Geometry pipeline, a collection of processes that convert 3D points from those that are most convenient for system developers into those that are most convenient for the display devices [114]. The pipeline is showed in Figure 6.5.

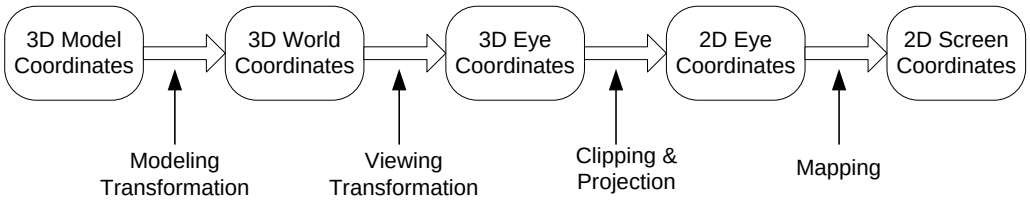


Figure 6.5: The geometry pipeline that achieves the coordinate transformation.

OpenGL has already realized this pipeline. Developers do not need to carry out the calculation themselves. Instead, they can use the functions introduced in Section 6.2 to complete these processes step by step. Thus the coordinate transforms become much

easier for developers, and they need only concern themselves with how to build up the surface model in 3D model coordinates. OpenGL uses a Right-Handed coordinate system by default as shown in Figure 6.6.

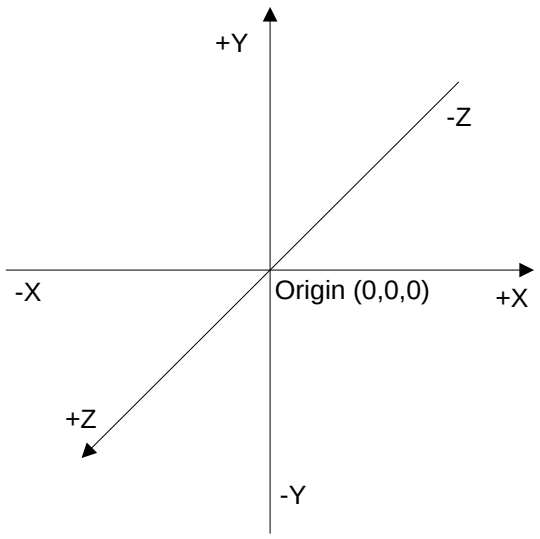


Figure 6.6: Default 3D model coordinate system of OpenGL.

6.4.2 Modelling

A surface is a physically continuous face with an infinite numbers of points. After the measurement, the number of points is reduced to a finite discrete value. Only a limited number of points are captured, and their height values are recorded in a surface file. Modelling a surface is to build up the surface prototype based on these values which are regarded as surface data.

Surface data is grid data with height values. It is not difficult to convert each height value to 3D points as the height value is related to a 2D coordinate which indicates the position where the height value is captured. The most critical issue is how to connect these points to form a complete surface. In this section, surface is divided into many triangles according to the position coordinates as shown in Figure 6.7. Hence, the surface can be created as long as each small triangle is filled with gradient colours[120].

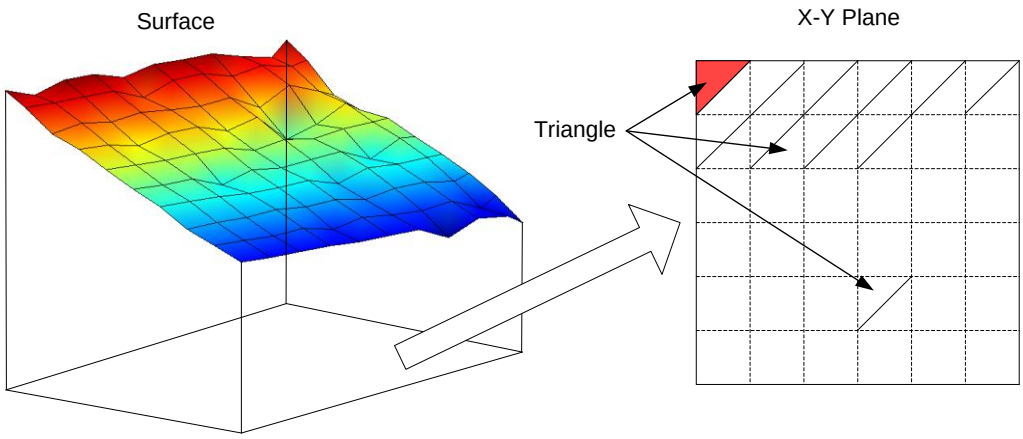


Figure 6.7: 3D modelling of surface data.

The above method of modelling is to build up the surface as naturally as possible. However, the requirement of the display may sometimes be different. For example, users may want to see the mesh view or point view. It is then necessary to model the surface in a different way. Certainly, developers can also create a new data display component to meet the new requirement.

6.4.3 Render

The colour is a significant property of each vertex in OpenGL. Specifying a colour value for a 3D point that is converted from its height value is necessary. Otherwise, all the points are plotted with the same colour, and the surface could not be viewed as expected. To make the 3D objects stereoscopic, the colour of a point is normally determined by its height information. Points with the same height value are plotted with the same colour. It is very convenient for understanding function for the surface data to be a collection of height values.

For surface analysis, it is insufficient to define the actual colour for a point based on height information. This transform could be realised by establishing a mapping relationship between the height and the colour. On the one hand, it is hard to define a unique range that is suitable for any surface data as the range of height values is indeterminate. Therefore, height values are required to be unitized first, and then the unit height value could be used for colour mapping. Also, one colour consisting of R, G and B values needs to be mapped with a numerical value. The best way is to set up a colour map which records all potential colour values. The more colour values used, the better colour

effects are produced. In addition, the colour values of the colour map should change gradually to make the 3D object looks smooth. At present, there are many colour maps designed for different colour effects [121]. Grey, Jet, cool, hot and Green are implemented in the presently developed 3D data display component, and users can select any one of them to render the surface.

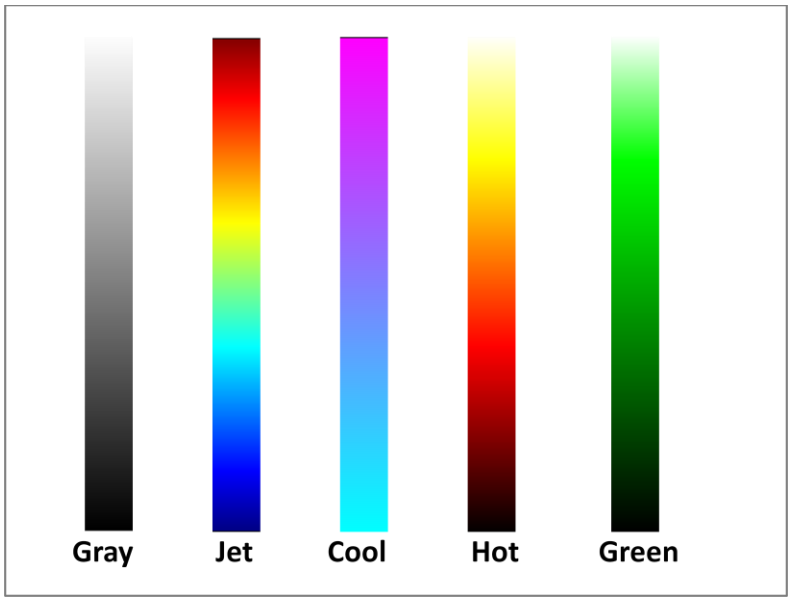


Figure 6.8: Supported colour maps for surface render.

As shown in Figure 6.8, they have different colour components in the colour map.

- Grey is a liner greyscale colour map;
- Jet ranges from blue to red, and passes through the colours cyan, yellow, and orange.
- Cool consists of colours that are shades of cyan and magenta;
- Hot varies smoothly from black through shades of red, orange, and yellow, to white;
- Green varies smoothly from black though shades of green to white.

6.4.4 Lighting and Object Materials

In OpenGL, the object material can be simulated to appreciate light and materials variations. As discussed in section 6.2, it is possible to make the surface more vivid as a

real surface by setting its materials properties when the lighting is enabled. In this data display component, many materials are available such as brass, ruby, jade, pearl and so forth [122]. As an example, the properties of brass are listed in Table 6.1.

Table 6.1: Property values for the brass material.

Property Name	Value
GL_AMBIENT	(0.0215f,0.1745f,0.0215f,1.0f)
GL_DIFFUSE	(0.07568f,0.61424f,0.07568f,1.0f)
GL_SPECULAR	(0.633f,0.727811f,0.633f,1.0f)
GL_SHININESS	0.6f

As long as the material properties of an object are specified, the surface looks like it is made from that kind of material. Figure 6.9 shows an example of surface data with brass as a material selected.

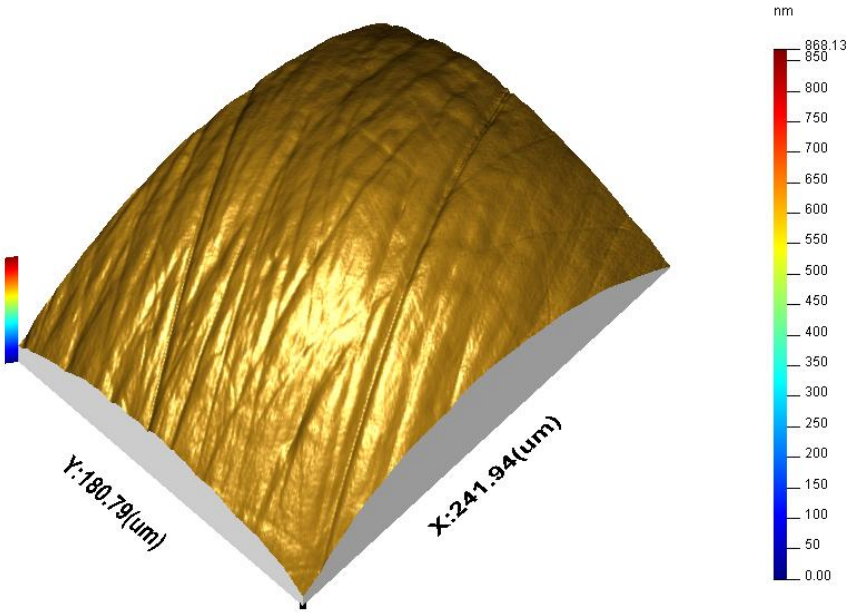


Figure 6.9: An example of materials rending—Brass.

6.4.5 Interface Implementation

As a data display component, the 3D display component has to be derived from the interface *SSDisplayIO*. Thus, it is required to implement two interfaces: *SSObject* and

SSDisplayIO. All the virtual methods defined in these two interfaces ought to be overriden:

- Derived from *SSObject*

There are two methods in *SSObject*: *OnRegister()* and *UnRegister()*. The *OnRegister()* method only needs to add the component information to system component list:

```
SSCommon.AddAssembly("3DDisplay", "DataDisplay.DataDisplay3D",
System.Reflection.Assembly.GetExecutingAssembly().Location,
SSAssemblyType.DATADISPLAY);
```

On the contrary, it is necessary to remove the component information in *Unregister()* correspondingly.

```
SSCommon.RemoveAssembly("3DDisplay", SSAssemblyType.DATADISPLAY);
```

- Derived from *SSDisplayIO*

As outlined, the interface *SSDisplayIO* is quite simple, and it only has one virtual method that needs to be implemented. There is not too much work in implementing the function *SetData()*. It only needs to confirm the data passed in is real 3D data and then transfer it to the actual painting control. All the real drawing and calculating codes are not presented here due to their enormous size.

6.5 Summary

This chapter places emphasis on the design and implementation of data display components. It firstly gives a brief introduction to computer graphics and its programming technique— OpenGL. Then the interface of the data display components is defined. Finally, the 3D display component is selected as a case study to illustrate the implementation of data display components with OpenGL technology.

Chapter 7 Test and Evaluation of the Proposed Surface Characterisation System

7.1 Introduction

System test and evaluation is essential as the process of validating and verifying whether the software system works as expected and meets the intended requirements. This process comprises a significant activity within the life cycle of a software system. According to differences of test objectives, the test of a system can be classified to many types such as unit tests, integration tests, system tests and acceptance tests [105, 123]. Different system development models will focus the test effort at different points in the development process. Generally they occur after the requirements have been defined and the coding process has been completed. This chapter will discuss the test of the new surface characterisation system and the evaluation of the system by comparing it with other systems. Testing the whole system involved many tests, for the sake of brevity a selection of case study tests are presented here.

7.2 Unit Tests

Unit tests usually refer to tests that verify the functionality of a specific section of code. In an object-oriented environment, it is usually at the class level. It means that this type of test happens whenever any tiny section of code is completed, and there is no need to elaborate them [124]. In this section, the unit test refers to the test at component level.

All the system functions related to surface analysis are implemented as an independent functional component, and each functional component is a standalone unit. Since they are developed individually, it is necessary to test them before adding them into the system framework. However, functional components cannot be executed in the operating system environment directly even if they are executable. They need a container in which all the functions can be invoked. There are many ways to simulate such a container. One possible way is to write the test code for each component respectively. The drawback of this method is that too much extra code has to be written, and this code would be useless when the test is completed. The simplest way is to add functional components to the system framework which could invoke their methods and modify their properties. The

prerequisite of using the method is the test can only be completed when the system framework is completed. In addition, Microsoft provides a tool AXCTC which is able to test ActiveX controls by changing their properties, invoking their methods, and firing their events [125]. Most of the data display components are tested with this tool.

7.2.1 Test Case 1: SDF File Component

Data access components are designed to access surface data files, and there are two main requirements for them: read and save surface data. This test case examines whether the SDF file component works as expected, namely, reading the SDF file to import surface data and saving a surface data to a file. The test is completed with the assistances of both the system framework and the 3D display component. To make the system framework be aware of SDF file component, it should be added in before the test.

1. Open a SDF file to check whether the surface data could be displayed.

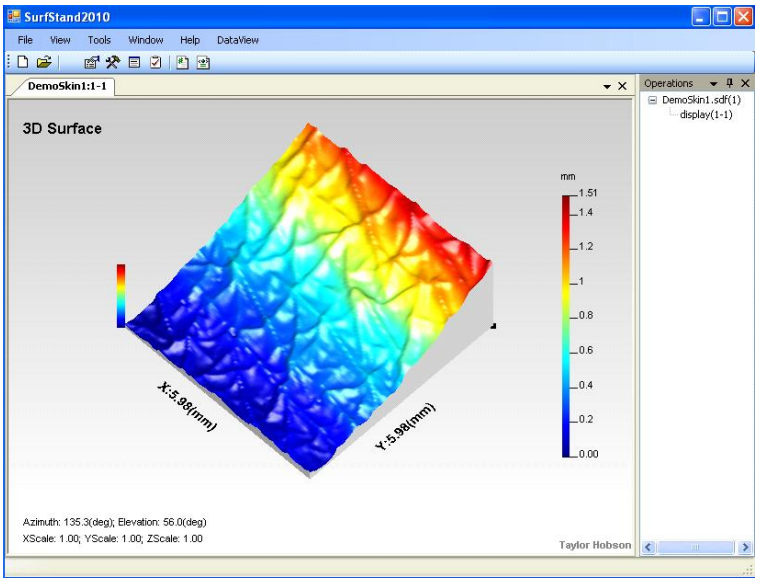


Figure 7.1: A SDF file opened with the SDF file component.

Actually, the surface data could be displayed correctly no matter which SDF file is opened. Figure 7.1 shows the output result when opening an SDF file.

2. Save current surface data to a SDF file, and opened the saved file to check whether the surface data is displayed as the same as the original one. Figure 7.2 show the test result.

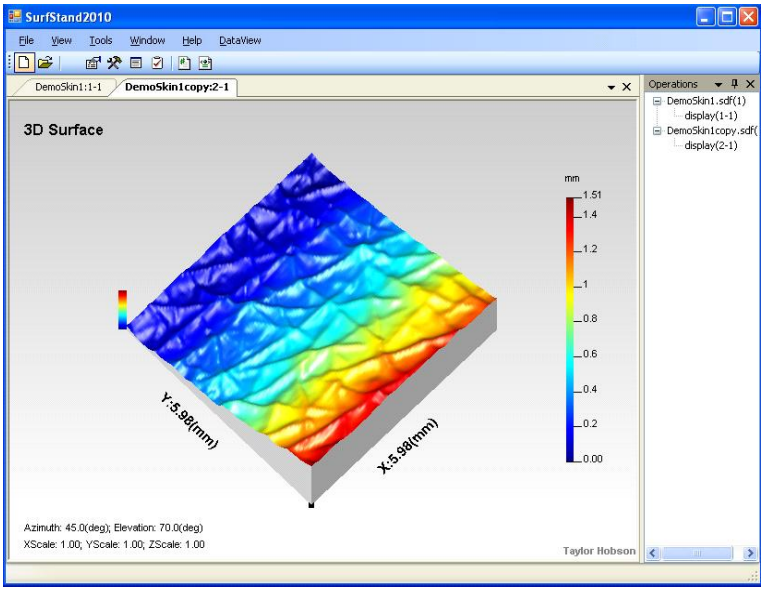


Figure 7.2 The surface data that is saved with the SDF file component.

3. Repeat the test step 1 and 2 on other SDF files to check whether every surface data could be displayed correctly.

During the test process, there is no an evidence of problems. Hence, the SDF file component is supposed to be correct, and it can be used to access SDF data files.

7.2.2 Test Case 2: 3D display component

Unlike other functional components, the 3D display component is an ActiveX control which could be tested with AXCTC. This case is designed to test whether the surface data could be presented by the component correctly, and all requirements of the 3D display component are satisfied. For example, the displayed surface should be presented correctly and rendered with corresponding colours.

1. Load the 3D display component to AXCTC, and then invoke the *setData()* function which accepts a file path as the input argument. Note that it is an override function which is different from the one introduced in Chapter 6.

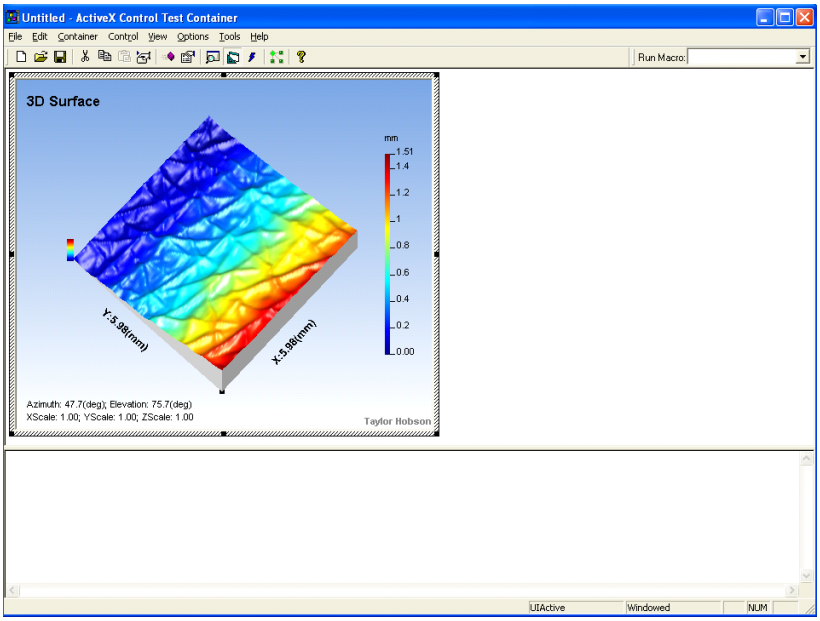


Figure 7.3: 3D display component loaded with ActiveX Control Test Container.

As shown in Figure 7.3, the surface data is correctly displayed on the 3D display control with the default component properties.

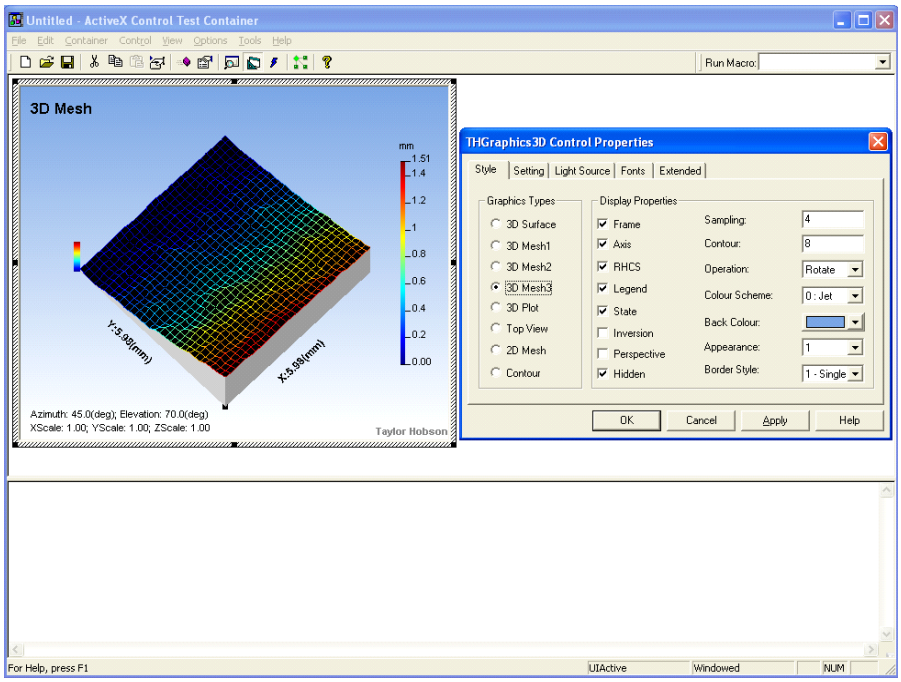


Figure 7.4: Surface data display with 3D mesh representation type.

2. Open the properties dialog, and then change the properties to check whether the surface data is still presented as expected. As there are many properties of the 3D display component, only three of them are selected as examples.

1) Change the graphics type from 3D surface to 3D mesh, the result is showed in Figure 7.4.

2) Change the colour map from Jet which is used by default to hot as shown in Figure 7.5.

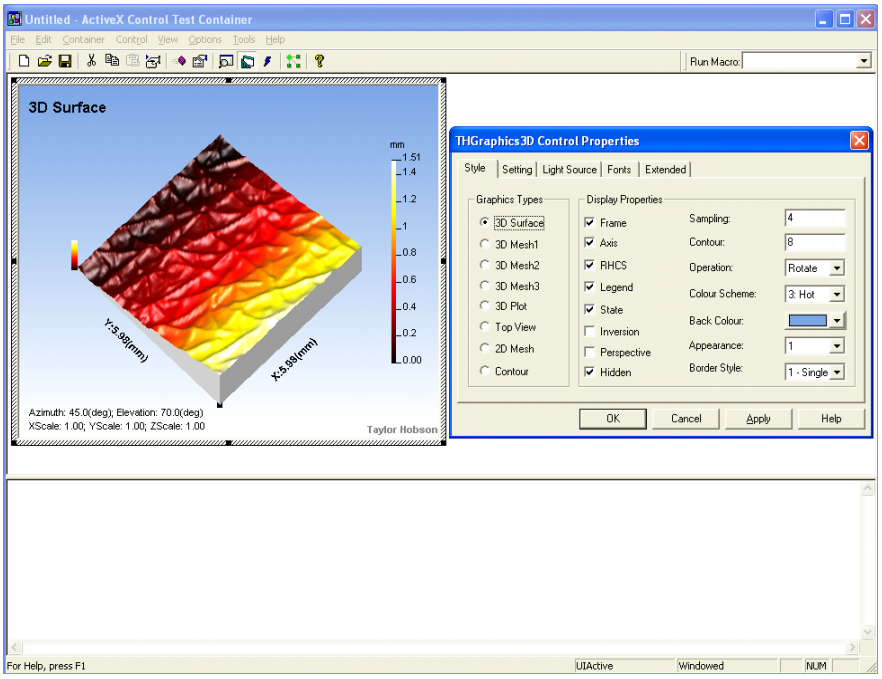


Figure 7.5: Surface data rendered with hot colour map.

3) Change the object material from colour scheme to yellow rubber as shown in Figure 7.6

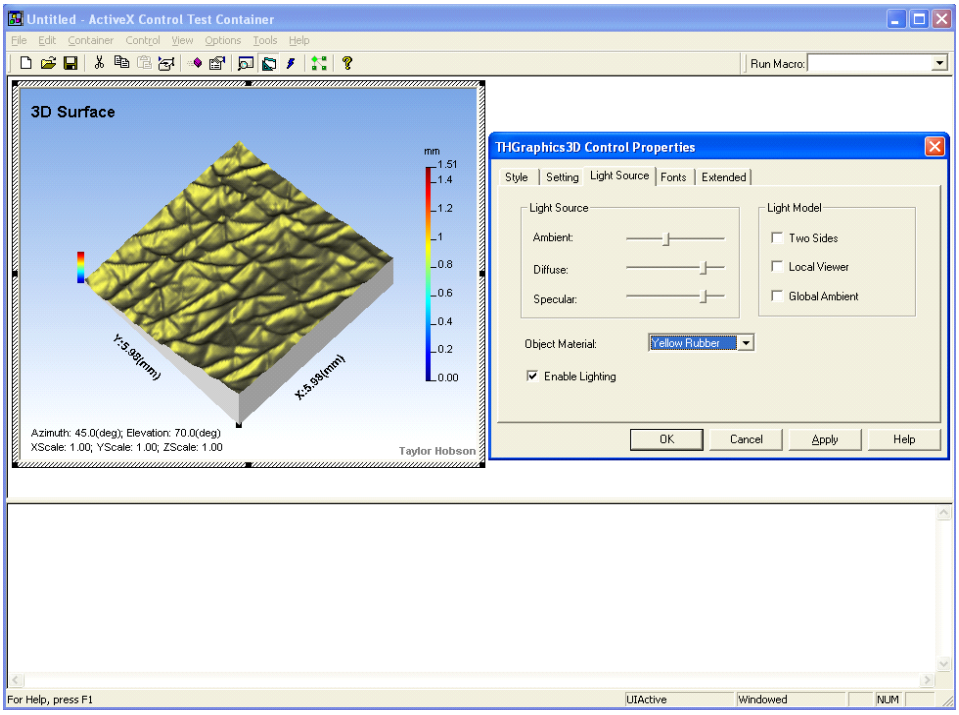


Figure 7.6: Surface data display with yellow rubber material.

The display results are the same as expected after changing properties of the 3D display component. Therefore, it could be thought as a qualified data display component.

7.3 Integration Tests

Integration tests work to expose defects in the interfaces and interaction between integrated components. During the unit test process, only the component functions are tested to check if they work as expected. However, it is necessary to test their interfaces in order to ensure that they are connected together and work correctly. The purpose of integration testing is to detect any inconsistencies between the software units that are integrated together [105, 123]. In the surface characterization system, all functional components are developed independently, thus they all need to be tested as to whether they can execute successfully after being added to the system framework. In other words, integration testing aims to test whether the component is compatible with the system framework.

Integration is essential for every functional component before it could be released for use. Three examples are selected to demonstrate the integration test for each type of functional components.

7.3.1 Test Case 1: SUR File Component

This test case is designed to check whether the SUR file component could be added and removed to the system successfully, and then SUR data files could be accessed by system or not.

1. Add the SUR file component to the system framework with the components configuration dialog as shown in Figure 7.7. The listbox on this dialog lists out all functional components which have already been added to the system framework.

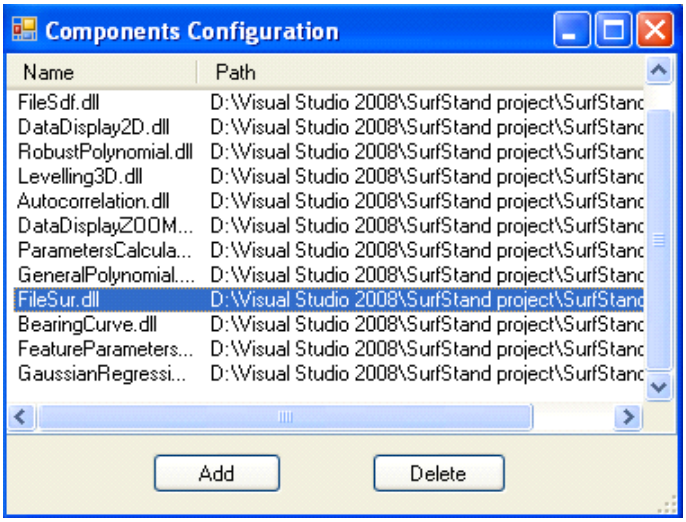


Figure 7.7: Components Configuration dialog.

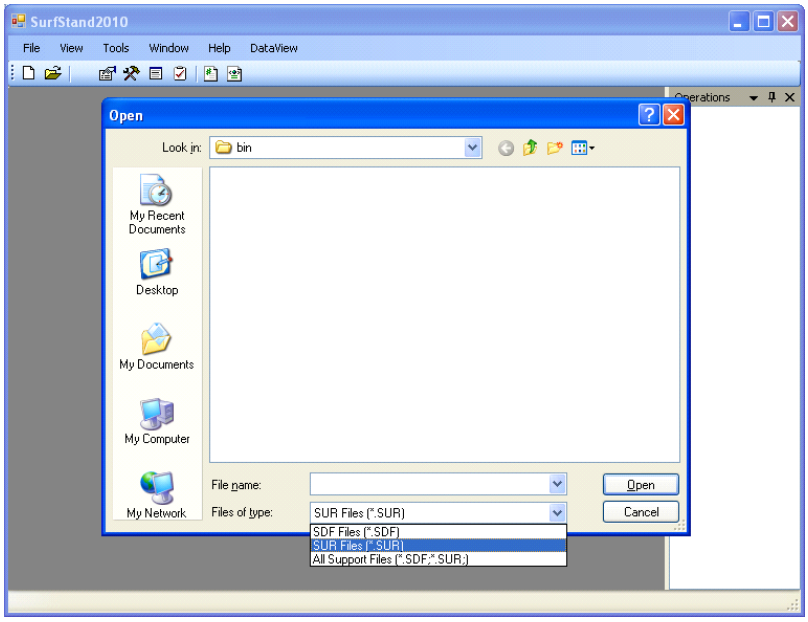


Figure 7.8: Open file dialog with supporting SDF and SUR file formats.

After adding the SUR File component ‘filesur.dll’, the SUR file becomes a support file format which can be found in the open file dialog, as shown in Figure 7.8.

2. Open a SUR file and then save it to check whether the function could be invoked successfully. As shown in Figure 7.9 the SUR file could be read and the surface data was display correctly.

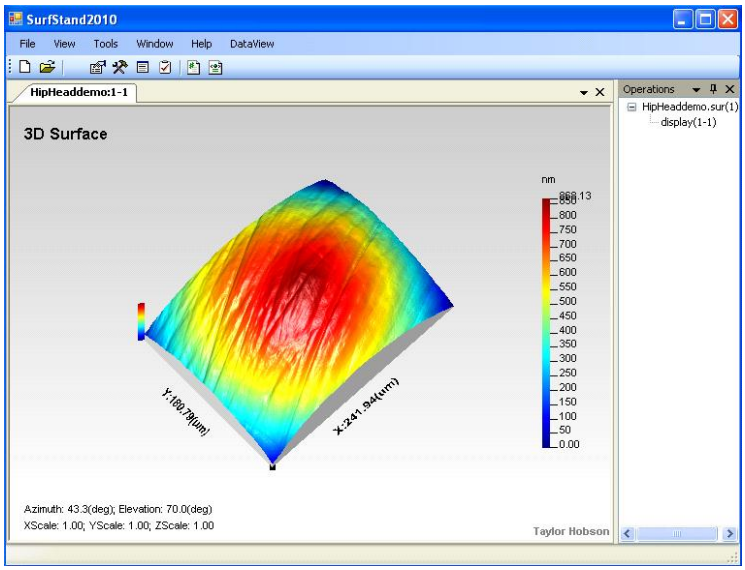


Figure 7.9: An example of surface data imported with SUR file component.

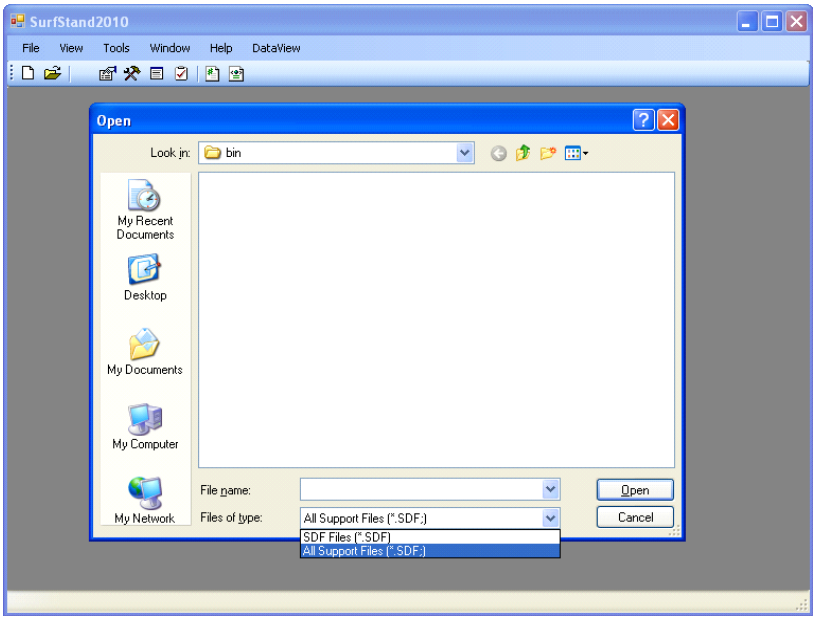


Figure 7.10: Open file dialog without supporting SUR file format.

3. Remove the SUR file component from the system framework to check whether the SUR file format is still supported by the system. As shown in Figure 7.10, there is no SUR file format in the open file dialog any more.

7.3.2 Test Case 2: Levelling Component

Similar to the test of the file access component, the aim of testing data process components is to check whether they could be added and removed successfully and work as expected.

1. Add the levelling component with the Component configuration dialog. Then the new menu item 'Levelling' states the component is added successfully (See Figure 7.11). Similarly, this menu item will disappear when removing the levelling component.

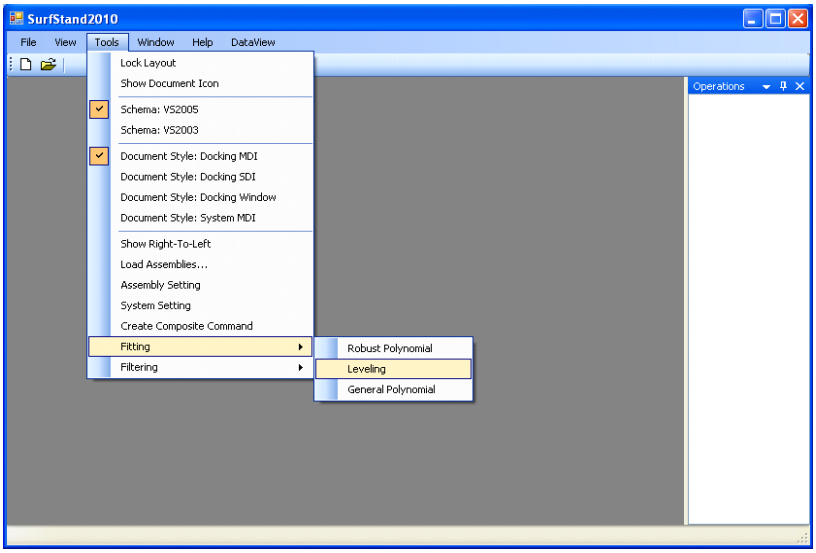


Figure 7.11: The emerging levelling menu item after adding the levelling component.

2. Open surface file, and execute the levelling function by clicking the menu item 'levelling'. Figure 7.1 is the original surface data. After levelling, the analysis result is display in a new 3D display component as shown in Figure 7.12.

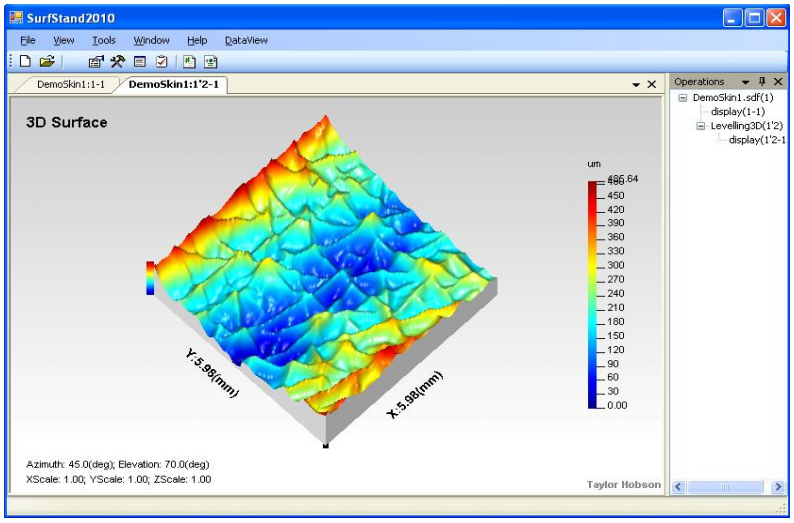


Figure 7.12: The surface data that has been processed with levelling component.

7.3.3 Test case 3: Topview component

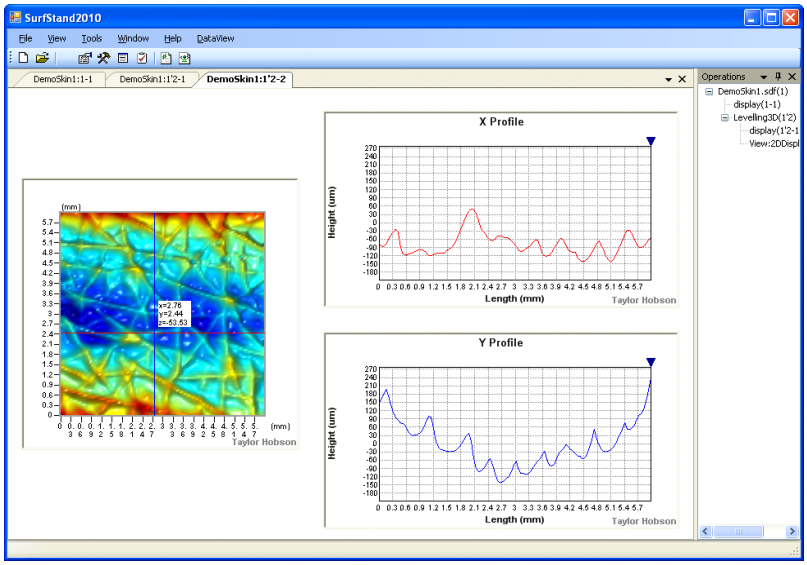


Figure 7.13: The surface data that is displayed with Topview Component.

The Topview component is to present a surface in a different way, and it can also display the profiles of the surface data. Since there is no obvious changes on the user interface after adding or removing a display component, an extra data process component ‘View in 2D’ is used to test the Topview component. After adding both of them, there is a menu item “View in 2D” occurs in the menu. Whether the Topview component works or not could be checked by clicking this menu item. Figure 7.13 presents the display result by invoking Topview component.

7.4 System Tests

System tests are performed on the whole system, and they serves to evaluate the system's compliance with its specified requirements [105]. The prerequisite for system testing is that all components should pass the unit test and integration test successfully. In this section, the primary work is to check whether the system could be configured dynamically and used to characterise a surface conforming to GPS standards. Similar to integration tests, two test cases are selected to demonstrate the system tests of the surface characterisation system.

7.4.1 Test Case 1 Components Configuration

All functional components are expected to have the ability to be added and removed dynamically. This case is designed to check whether system could load and remove a functional component even when the system is running.

1. Start the system, and then open a surface file. The surface data should be successfully displayed, as shown in Figure 7.1.
2. Open the 'components configuration' dialog. Reconfigure system functions by adding or removing functional components. For simplicity, the ‘Bearing curving analysis’ function is added to test whether the system function configuration is success.
3. Invoke the bearing curve analysis function by clicking the new menu item ‘Bearing curve’. The analysis result is displayed as in Figure 7.14.
4. Open the ‘component configuration’ dialog again, and then remove the functional component. The system is still running and does not crash. The analysis result of bearing curve is displayed the same as before the removal process.

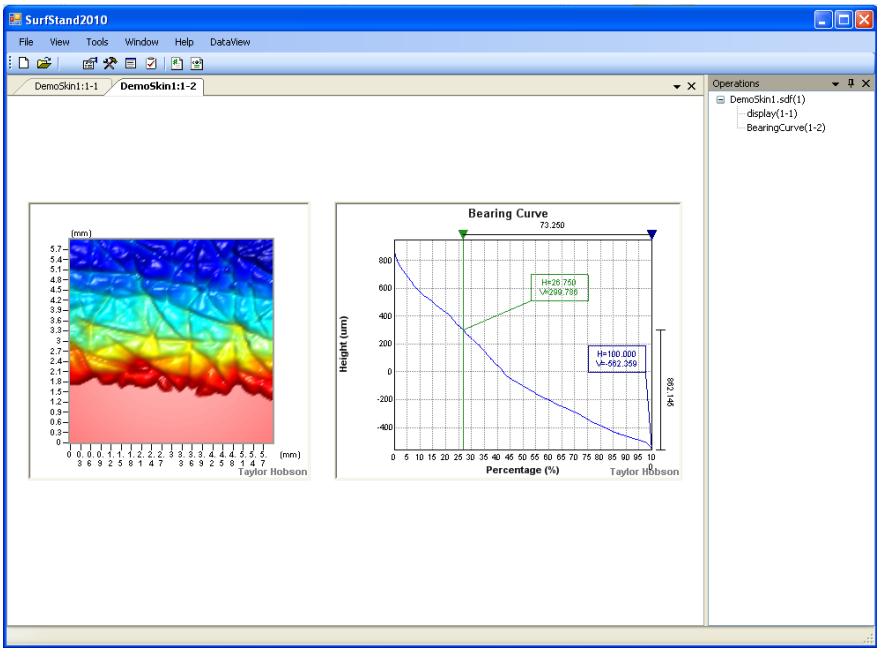


Figure 7.14:The surface data that is processed with Bearing curve analysis component.

In this case, the new functional component bearing curve analysis is successfully added and removed dynamically. The more important thing is that this configuration would not affect the execution of the system.

7.4.2 Test Case 2: Compound Command Test

Surface characterisation ought to be realised with a surface verification operator which is composed of a series of operations. As discussed in Chapter 5, a compound command consisting of a number of commands is beneficial to the realisation of the surface verification operator. This case aims to test whether the compound command can be created and works as expected. For instance, a compound command which is composed of levelling and Gaussian Regression filtering will be demonstrated.

1. Open the ‘command setting’ dialog and add a new compound command ‘Test command’. Select ‘New View’ to display the analysis result for this new command as shown in Figure 7.15.

Figure 7.15: Compound command creating dialog.

2. Add the levelling, Gaussian Regression filtering to subcommand list box as shown in Figure 7.16. After clicking the 'OK' button, the new command is created as shown in Figure 7.17. Meanwhile, a new menu item is also created for 'Test command'.

Figure 7.16: Subcommands modification of Test command.

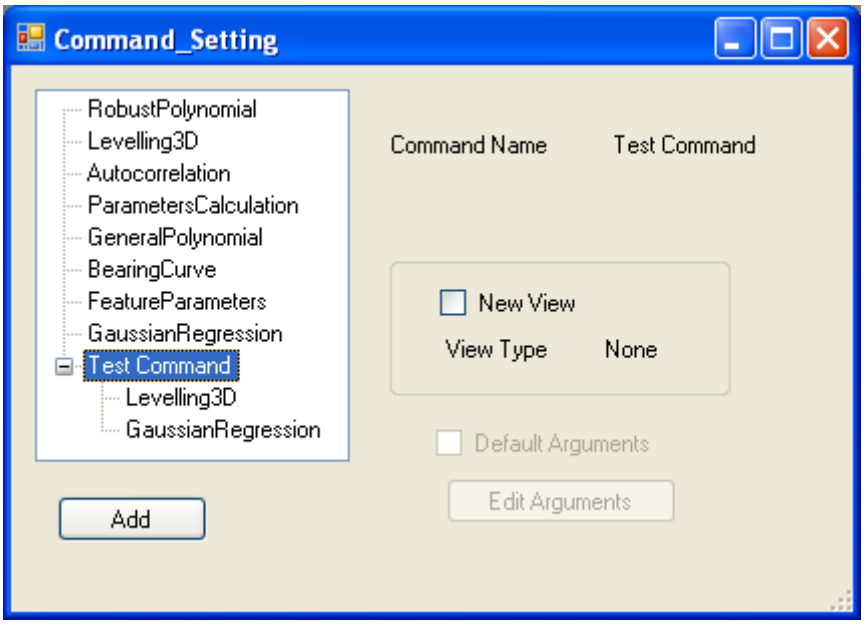


Figure 7.17: Test command occurs in the Command setting dialog.

3. Open a surface file as shown in Figure 7.1, and then check whether all arguments of each sub command are appropriate for the surface. Arguments can be modified by clicking the "Edit Arguments" button when the command is selected. Figure 7.18 shows the arguments dialog of Gaussian Regression filter.

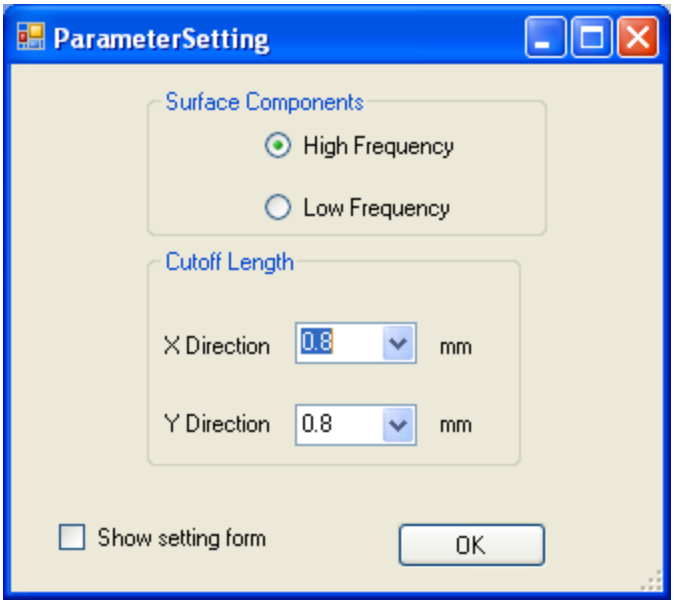


Figure 7.18: Parameters setting dialog of Gaussian regression filter.

4. Execute the compound command. The result is displayed in Figure 7.19 and it can be used for parameters calculation directly. All the actions applied to the surface are listed in the right window. It is an action list which is convenient for users to change the parameter setting of each step.

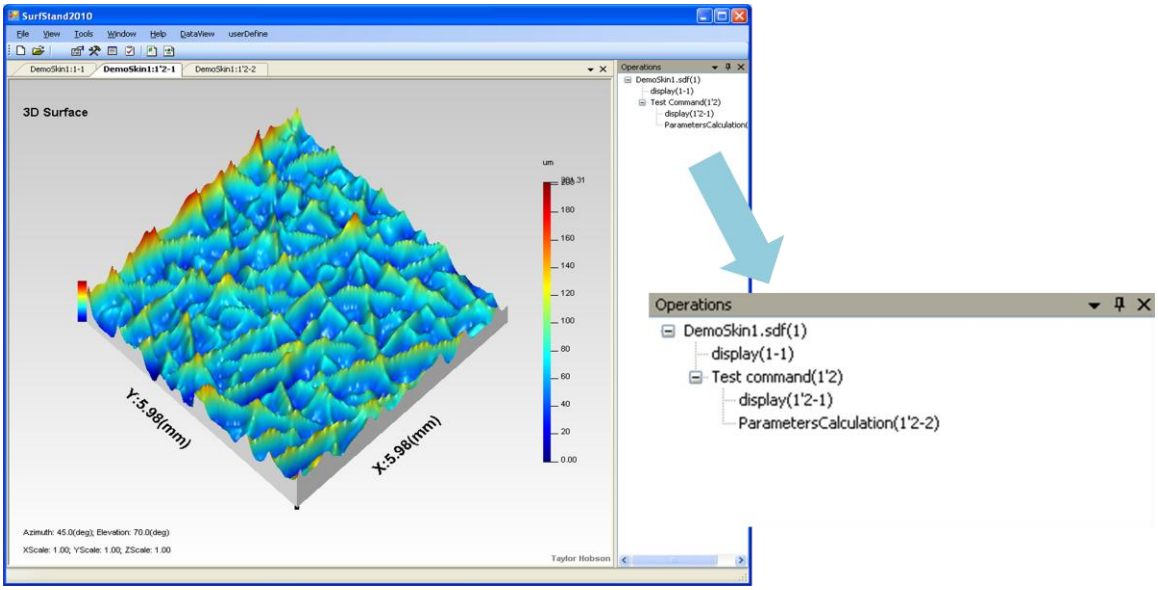


Figure 7.19: The surface data that is processed with Test command.

In a word, the surface operator is successful supported in the new surface characterisation system. Users could create any compound command as desired, and then apply them on different surfaces. Meanwhile, the action list is helpful to trace the actions that apply to the surface data.

7.5 Comparison

At present, there are many surface characterisation systems developed by different organisations. This section will give a brief comparison between the new surface characterisation system Surfstand and other systems. One system is "Talymap" which is a powerful surface characterisation system, and it is distributed on most of surface instruments produced by Taylor Hobson [8, 126]. The second is the online characterisation system developed by NIST [10, 127-129]. The last one is "SPIP^{TM}," which is provided by Image Metrology [77].

As shown in Table 7.1, Surfstand is superior to the other two systems in maintainability, extensibility and reusability. It also supports hot swap which means functional

components could be exchanged when the system runs. The advantage of the NIST online system is that it is compatible to any OS system as it is an internet application. Both Surfstand and Talymap could be extended by end users. However, Surfstand is much easier than Talymap. Instead of implementing some interfaces, the requirement would be to add a menu and write a script to add a new function for Talymap. Although SPIP^{TM} has good expansibility, ISO standards are not fully supported. In the meanwhile, the reusability of its functions is not so good.

Table 7.1: Comparison between Surfstand and other systems.

	SurfStand	Talymap	NIST online System	SPIP
Application Type	Desktop	Desktop	Web	Desktop
Standards Support	Yes	Yes	Yes	No
Hot swap	Yes	No	No	Yes
Plug and play	Yes	No	No	Yes
Verification Operator	Yes	Yes	No	No
Compatibility	Windows	Windows	Window, Linux, Mac OS	Windows
Function reusability	Good	Mediocre	Poor	Poor
System extendibility	Good	Mediocre	Poor	Good
System Maintain	Easy	Medium	Difficult	Medium
User Customise	Absolutely	Partially	No	Partially

7.6 Summary

This chapter provides detailed discussions on the test and evaluation carried out on the flexible surface characterisation system. To ensure the system works as expected, a series of test cases are given in order to highlight the system functions from different perspectives. After these testing, it is ensured that the proposed characterisation system can run normally as designed. All system functional components can be added, removed or replaced dynamically, and all bugs found during the test process have already been fixed. Users can customise their own functional components according the predefined interfaces. Moreover, they can also define their own verification operators by combing relevant operations.

The final comparison section shows that the system has the distinctive capability of dynamic reconfiguration. The proposed characterisation system supports the standard

surface verification operator and each operation is strictly accordance with GPS standards. In addition, as all functional components work in a “plug and play” way, the system is very easy to be extended and maintained. Meanwhile, functional components themselves are apt to be reused in other systems because they are developed as COM objects which are executable and language-neutral. With the flexible architecture, the functions of the system are no longer determined at the development stage, instead they can be reconfigured by users after the system is deployed.

Chapter 8 Conclusions and Future Work

In this thesis, a novel flexible software system SurfStand was designed and implemented for the purpose of characterising surface metrology. The fundamental philosophy behind developing this system in the manner described is construction rather than creation. All system functions related with surface analysis processing are separated from the system framework and developed individually as functional components. The system is established by assembling these independent components off-the-shelf. Consequently the philosophical approach facilitates expansion, customisation and user led development. This is a unique feature for such metrology systems. This chapter summarises the outcomes and highlights the contribution to knowledge in the relevant research domains. In addition, future work related to this system is also discussed.

8.1 Summary

Surface metrology as a discipline is undergoing rapid development. More and more analysis and evaluation methods are used to quantify surface features, and additional parameters are used to indicate these characteristics. Therefore, surface characterisation system should ideally be constantly updated to support new characterisation technologies. In reality however, most of the present surface characterisation systems are hard to maintain and be extended due to their poor extendibility, reusability and maintainability. This thesis is devoted to developing an architecture which could facilitate the system maintenance and evolution, and then establish a novel surface characterisation system. The work of this thesis comprises:

1) Investigation of surface characterisation techniques for surface metrology; It is necessary to have an excellent knowledge of how to characterise a surface with relevant techniques before implementing them with a certain programming language. Chapter 2 gives a brief introduction concerning conventional surface characterisation techniques.

2) Design of flexible system architecture; Normally system architecture is closely related with system functions, and it becomes more and more complex due to the increasing the scope of the system functions. As discussed in section 3.3, the

flexible system architecture is designed by separating system functions from the system framework. All functional components are invoked dynamically in a ‘plug and play’ manner.

3) Categorisation of functional components; There are three types of functional components classified according to their roles in the surface analysis process as shown in Figure 8.1.

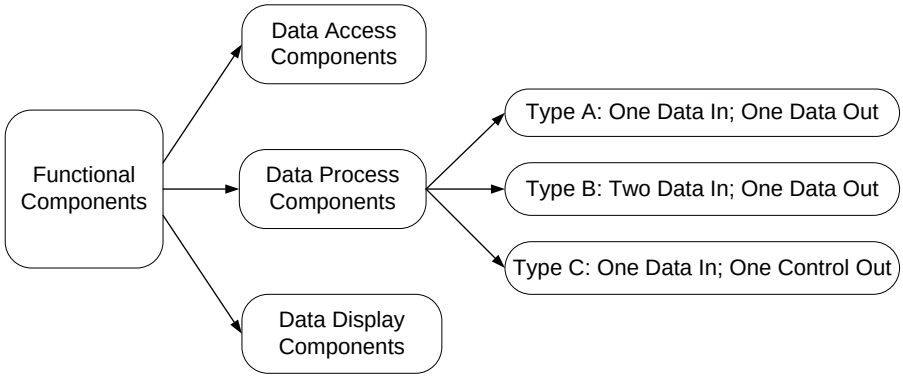


Figure 8.1: Categorisation of system functional components.

4) Design and implementation of three types of functional components; Data access components, data process components and data display components are elaborated in Chapter 4, Chapter 5 and Chapter 6 respectively. For the reason of brevity only one case of each type is introduced. Figure 8.2 illustrates all functional components that have been implemented.

5) Establishment of the user customisation mechanism; Users are able to develop their own functional components as long as associated interfaces are implemented. All functional components are developed in this way, and the developer of each functional component does not have to know the system architecture itself.

6) Accomplishment of system tests. As discussed in Chapter 7, SurfStand is verified to work as expected and all functional requirements are satisfied.

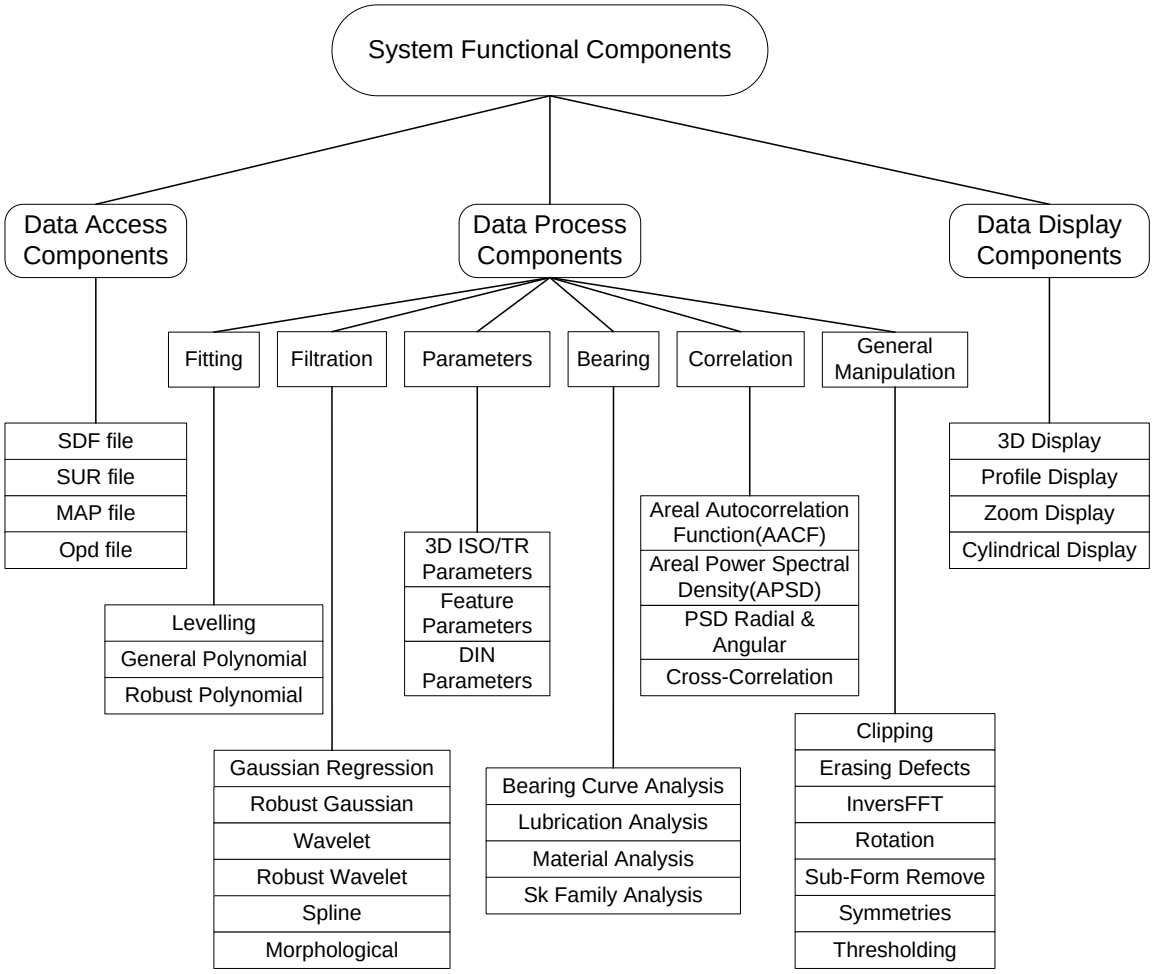


Figure 8.2: Completed system functional components.

8.2 Contribution to Knowledge

This thesis proposes a new approach to building a surface characterisation system, which addresses issues relating to the standardisation analysis, functions modularisation and system flexibility. The contributions of this thesis are listed as follows:

8.2.1 Standardisation Analysis

Surface characterisation systems are mostly deployed in measurement instruments, and they are developed by different organisations. It is convenient and effective for metrologists to carry out the characterisation and evaluation process after the measurement. However, there are large variations in analysis results among different characterisation systems, caused by diverse implementations, choices of analysis methods and ambiguous definitions. The analysis results are always incomparable and untraceable.

To reduce the variations of analysis results, the standardisation analysis process is supported in the new characterisation system SurfStand.

1) The implementations of surface verification operations are in strict accordance with GPS standards. ISO issued a series of characterisation standards which aim to standardise the characterisation methods. For example, ISO4287 specifies terms, definitions and parameters for the determination of surface texture by profiling methods, while ISO 25178-2 specifies them by areal methods. All of these are implemented in the present work.

2) Surface verification operators are supported. According to ISO 17450-1, the process of surface characterisation should be completed with a surface operator which consists of a series of operations. Although most of the present surface characterisation systems have realised plenty of surface operations, only a few of them support surface analysis with verification operators. In most cases, the actual surface operators are normally created by the metrologists along with the analysis process, and they are greatly influenced by subjective factors. Thus, it is very hard to ensure that the surface operators are absolutely the same even if they are created by the same metrologist. The uncertainty of surface operators may lead to huge variations of analysis results. Conversely, once surface operators are supported, they could be defined ahead and used as a whole. Different analysis results will not be realised as long as the same operator applied. Furthermore, Surfstand defines several surface operators according to the GPS standards, and they are used for standard parameters analysis.

8.2.2 Function Modularisation

Functions are certainly the most important properties of a software system, and they are realised to satisfy various system requirements. Generally the complexity of software systems is determined by the degree of coupling between the systems and their functions. Function modularisation is an effective way to achieve loose coupling which is beneficial to simplification of the system architecture. The implementation of function modularisation should be done according to the requirements and characteristics of the system. In SurfStand, there are two critical design issues for function modularisation.

1) Determine the size of functional modules.

This point is closely related to the design of system architecture. The more functional modules are defined, the more complex the system structure is. However, it does not mean that the size of functional modules should be defined as large as possible, because the complexity of functional modules will increase along with its size. Determining the appropriate granularity is significant to keep the balance of the complexity between functional modules and the system. In practice, the granularity of functional modules is determined according to surface operations whilst considering the standardisation analysis. In the present case every surface operation is defined as a single functional module and implemented as a functional component.

2) Categorisation of system functions.

This point is the foundation of system functions reconfiguration. Functional components are anticipated as being added or removed from the system framework dynamically. However, the system framework cannot treat functional components with different properties and purposes in the same way. Moreover, for the sake of system flexibility, writing relevant codes for each functional component individually is unfeasible. Hence, categorisation is necessary to enable the consistency of invocation. As discussed in section 3.5, all system functional components related to surface characterisation are categorised into three types: Data access components, Data process components and Data display components.

8.2.3 System Flexibility

Although most of present surface characterisation systems are adequate for various surface analyses, the rigidity and lack of flexibility of the system architecture make it difficult to maintain and evolve them to new analytical demands. Nowadays, as more and more technologies are employed to evaluate the quality of surfaces, surface characterisation systems should have the ability to be upgraded constantly. Therefore, this thesis proposes a flexible architecture for the surface characterisation system as discussed in section 3.3. Its advantages are concluded as below:

1) Reduction in development time and cost. The component, as an independent module, is designed and implemented separately. The loosely coupled

relationship between the components and framework allows parallel development. Thus, the system development cycle is shortened. In addition, many system functions can be implemented by reusing legacy codes, and they are able to be reused by other systems directly.

2) Reduction in maintenance costs. As functional components are encapsulated as a whole, they can be added or removed without affecting the system architecture. Hence, the complexity of system architecture is not changed no matter how many functional components have been configured. Meanwhile, the modification happens in one functional component will not lead to the recompilation and redeployment of other parts of the system.

3) Enhanced extendibility and diversity. Instead of being combined together at the beginning, functional components are connected with the system framework dynamically. This means that the component could work in a 'plug and play' manner. Users can reconfigure system functions as they expect. Moreover, they are able to customise their own functional components to satisfy certain individualised functional requirements.

8.3 Future Work

The new flexible surface characterisation system has already been realised. With the implementation of a number of functional components, SurfStand is able to complete general surface characterisation. To improve the performance and enhance the practicability, some future work to SurfStand is suggested below:

1) Evaluation of User interface; User interface is the space where interaction between users and the system occurs. A good user interface usually makes it easy, efficient and enjoyable to operate the system. During the development process of SurfStand, user interface was not the subject of as much development as the system functions. There are still a lot of improvements which could be completed for user interface such as layout optimisation, input convenience and output multiplicity.

2) Expansion of surface data source; At present, surface data file is regarded as the unique data source of SurfStand. To expand the usable range, it is necessary to

increase the data source. For example, surface measurement instruments are the potential data sources as most of surface data stored in data files are captured from them. Therefore, new types of data access components are required to be defined to support fresh data sources.

3) Development of new functional components; Although many functional components have been developed, new functional components are always in demand as more and more characterisation techniques are developed to satisfy various analysis requirements. Specific functional requirement such as that concerned with ballistics metrology could be added and other function components removed to facilitate for example a dedicated ballistics metrology system thus illustrating the customisation capability of the overall approach adopted.

4) Investigation and development based on new surface operators. SurfStand has already defined a few standard surface verification operators which aim to calculate parameters defined in GPS standards. They are more suitable for stochastic surface characterisation, and new surface operators are required for other types of surface such as structure surface. Meanwhile, creating more surface operators would facilitate standard surface characterisation.

References

1. Blunt, L., & Jiang, X. (2003). *Advanced techniques for assessment surface topography*. Oxford: Butterworth-Heinemann.
2. Humienny, Z., Bialas, S., Osanna, P. H., Tamre, M., Weckenmann, A., Blunt, L., et al. (2001). *Geometrical Product Specification: Course for Technical Universities*. Warsaw: Warsaw Univ. of Technology Printing House.
3. Whitehouse, D. J. (1978). *Surfaces — A link between manufacture and function.* Paper presented at the Proceedings of the Institution of Mechanical Engineers.
4. Jiang, X., Scott, P. J., Whitehouse, D. J., & Blunt, L. (2007). Paradigm shifts in surface metrology. Part I. Historical philosophy. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Science, 463*(2085), 2049-2070.
5. Jiang, X., & Whitehouse, D. J. (2012). Technological shifts in surface metrology. *CIRP Annals-Manufacturing Technology, 61*(2), 815-836.
6. Conroy, M., & Armstrong, J. (2005). *A comparison of surface metrology techniques*. Paper presented at the Journal of Physics: Conference Series. http://www.iop.org/EJ/abstract/1742-6596/13/1/106
7. Whitehouse, D. J. (1997). Surface metrology. *Measurement Science and Technology, 8*(9), 955–972.
8. TaylorHobson. (2013). *Talymap Software*. Retrieved from http://taylor-hobson.virtualsite.co.uk/talymap_software.htm
9. Sacerdotti, F., Porrino, A., Butler, C., Brinkmann, S., & Vermeulen, M. (2002). SCOUT-surface characterization Open-Source universal toolbox. *Measurement Science and Technology, 13*(2). doi: 10.1088/0957-0233/13/2/401
10. Bui, S. H., Gopalan, V., & Raja, J. (2001). An internet based surface texture information system. *International Journal of Machine Tools and Manufacture, 41*(13), 2171-2177.
11. ISO 4287. (1998). *Geometrical Product Specifications (GPS) — Surface texture: Profile method-Terms, Definitions, and Surface texture parameters*. International Organization for Standardization.
12. ISO 25178-2. (2007). *Geometrical product specifications (GPS) — Surface texture: Areal — Part 2: Terms, definitions and surface texture parameters*. International Organization for Standardization.
13. Leach, R. K., & Harris, P. M. (2002). Ambiguities in the definition of spacing parameters for surface-texture characterization. *Measurement Science and Technology, 13*, 1924-1924.
14. Plakosh, D., & Lewis, G. A. (2003). *Modernizing legacy systems: Software technologies, engineering process and business practices*: Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA.
15. Mathia, T. G., Pawlus, P., & Wieczorowski, M. (2011). Recent trends in surface metrology. *Wear, 271*(3), 494-508.
16. Lonardo, P. M., Trumpold, H., & De Chiffre, L. (1996). Progress in 3D surface microtopography characterization. *CIRP Annals-Manufacturing Technology, 45*(2), 589-598.
17. Lientz, B. P., & Swanson, E. B. (1980). *Software maintenance management: a study of the maintenance of computer application software in 487 data processing organizations*. Reading MA: Addison-Wesley.
18. Lientz, B. P., & Swanson, E. B. (1981). Problems in application software maintenance. *Communications of the ACM, 24*(11), 763-769.

19. Sun, W., Zhang, X., Guo, C. J., Sun, P., & Su, H. C. (2008). *Software as a service: Configuration and customization perspectives.* Paper presented at the Congress on Services Part II, 2008. SERVICES-2. IEEE
20. Papajorgji, P., Beck, H. W., & Braga, J. L. (2004). An architecture for developing service-oriented and component-based environmental models. *Ecological Modelling, 179*(1), 61-76.
21. Grundy, J., & Hosking, J. (2002). Developing adaptable user interfaces for component-based systems. *Interacting with Computers, 14*(3), 175-194.
22. Schneider, J. G., & Han, J. (2004). *Components–the Past, the Present, and the Future.* Paper presented at the Workshop on Component-Oriented Programming.
23. McArthur, K., Saiedian, H., & Zand, M. (2002). An evaluation of the impact of component-based architectures on software reusability. *Information and Software Technology, 44*(6), 351-359.
24. Qureshi, M. R. J., & Hussain, S. A. (2008). A reusable software component-based development process model. *Advances in Engineering Software, 39*(2), 88-94.
25. Heineman, G. T., & Councill, W. T. (2001). *Component-based software engineering: putting the pieces together*. USA: Addison-Wesley
26. Crnkovic, I. (2004). Component-based software engineering-new challenges in software development. *Journal of Computing and Information Technology, 11*(3), 151-161.
27. Microsoft. (2009). *COM: Component Object Model Technologies*. Retrieved from http://www.microsoft.com/com/default.mspx
28. OMG. (2009). *Catalog of OMG CORBA/IIOP Specifications*. Retrieved from http://www.omg.org/technology/documents/corba_spec_catalog.htm
29. Sun. (2009). *Remote Method Invocation (RMI)*. Retrieved from http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp
30. Emmerich, W., & Tai, S. (2000). *Engineering distributed objects*. UK: Wiley Chichester.
31. OMG. (2009). *Unified Modeling Language (UML)*. Retrieved from http://www.omg.org/technology/documents/formal/uml.htm
32. Papajorgji, P., & Shatar, T. M. (2004). Using the Unified Modeling Language to develop soil water-balance and irrigation-scheduling models. *Environmental Modelling & Software, 19*(5), 451-459.
33. Deitel, H. (2008). *Visual c# 2008 how to program*. Harlow: Prentice Hall Press.
34. Szyperski, C., Gruntz, D., & Murer, S. (2002). *Component software: beyond object-oriented programming*. USA: Addison-Wesley.
35. Wolfgang, P. (1994). *Design patterns for object-oriented software development*. Reading, MA: Addison-Wesley.
36. Whitehouse, D. J. (1994). *Handbook of surface metrology*. Bristol: Institute of Physics.
37. Srinivasan, V. C. (2001). *An integrated view of geometrical product specification and verification.* Paper presented at the Proceedings th Cirp International Seminar on Computer Aided Tolerancing.
38. Durakbasa, M. N., Afjehi-Sadat, A., & Nomak, A. (2001). Dimensional and Geometrical Measurements and Interperation of Measuring results on the Basis of the Skin Model. *Measurement Science Review, 1*(1), 89-92.
39. ISO/TS 17450-1. (2000). *Geometrical Product Specifications (GPS) — General concepts — Part 1: Model for geometrical specification and verification*. International Organization for Standardization.
40. ISO 5436-2. (2001). *Geometrical Product Specifications (GPS) — Surface texture: Profile method; Measurement standards — Part 2: Software measurement standards*. International Organization for Standardization.

41. ISO 25178-7. (2009). *Geometrical Product Specifications (GPS) — Surface texture: Areal — Part 7: Software measurement standards*. International Organization for Standardization.
42. ISO 3274. (1998). *Geometric product specifications (GPS) — Surface texture: Profile method — Nominal characteristics of contact (stylus) instruments*. International Organization for Standardization.
43. ISO 11562. (1998). *Geometrical Product Specifications (GPS) — Surface texture: Profile method — Metrological characteristics of phase correct filters*. International Organization for Standardization.
44. ISO 13565-1. (1998). *Geometric product specifications (GPS) — Surface texture: Profile method — Surfaces having stratified functional properties — Part 1: Filtering and general measurement conditions*. International Organization for Standardization.
45. ISO 13565-2. (1998). *Geometric product specifications (GPS) — Surface texture: Profile method. Surfaces having stratified functional properties — Part 2: Height characterization using the linear material ration curve*. International Organization for Standardization.
46. ISO 13565-3. (1998). *Geometrical Product Specifications (GPS) — Surface texture: Profile method; Surfaces having stratified functional properties — Part 3: Height characterization using the material probability curve*. International Organization for Standardization.
47. Krystek, M. P. (2005). Spline Filters for Surface Texture Analysis. *Key Engineering Materials, 295*, 441-446.
48. Muralikrishnan, B., & Raja, J. (2008). *Computational Surface and Roundness Metrology*. New York: Springer.
49. Raja, J., Muralikrishnan, B., & Fu, S. (2002). Recent advances in separation of roughness, waviness and form. *Precision Engineering, 26*(2), 222-235.
50. Blateyron, F. (2006). *New 3D parameters and filtration techniques.* Paper presented at the Proceedings of the JSPE.
51. Whitehouse, D. J. (2002). *Surfaces and their measurement*. London: Hermes Penton.
52. Leach, R. (2001). *The measurement of surface texture using stylus instruments*. London: National Physical Laboratory.
53. Whitehouse, D. J. (1987). Surface metrology instrumentation. *Journal of Physics E: Scientific Instruments, 20*(10), 1145. doi: 10.1088/0022-3735/20/10/001
54. Schlesinger, G. (1942). *Surface finish: report of the Research Department*. London: Institution of Production Engineers.
55. Czichos, H., Saito, T., & Smith, L. E. (2011). *Springer handbook of metrology and testing*. New York: Springer.
56. Guenther, R. D. (1990). *Modern optics* (Vol. 1). New York: Wiley.
57. Leach, R. K. (2011). *Optical measurement of surface topography*. London: Springer.
58. Whitehouse, D. J. (1988). Comparison between stylus and optical methods for measuring surfaces. *CIRP Annals-Manufacturing Technology, 37*(2), 649-653.
59. Grandy, D., Koshy, P., & Klocke, F. (2009). Pneumatic non-contact roughness assessment of moving surfaces. *CIRP Annals-Manufacturing Technology, 58*(1), 515-518.
60. Robertson, T. J., Hutchins, D. A., Billson, D. R., Rakels, J. H., & Schindel, D. W. (2002). Surface metrology using reflected ultrasonic signals in air. *Ultrasonics, 39*(7), 479-486.
61. Mainsah, E., Greenwood, J. A., & Chetwynd, D. G. (2001). *Metrology and properties of engineering surfaces*. New York: Springer.
62. Griffiths, B. (2001). *Manufacturing surface technology: surface integrity and functional performance*. London: Elsevier.
63. Lancaster, P., & Salkauskas, K. (1986). *Curve and surface fitting. An introduction*. Waltham, Massachusetts: Academic Press.

64. Bjorck, A. (1996). *Numerical methods for least squares problems*. USA: Society for Industrial and Applied Mathematics.
65. WIKIPEDIA. (2012). *Least squares*. Retrieved from http://en.wikipedia.org/wiki/Least_squares
66. Guest, P. G. (2012). *Numerical methods of curve fitting*. Cambridge Cambridge University Press.
67. Jiang, X., Scott, P. J., Whitehouse, D. J., & Blunt, L. (2007). Paradigm shifts in surface metrology. Part II. The current shift. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Science, 463*(2085), 2071-2099.
68. Gruen, A., & Akca, D. (2005). Least squares 3D surface and curve matching. *ISPRS Journal of Photogrammetry and Remote Sensing, 59*(3), 151-174.
69. Stout, K. J., Sullivan, P. J., Dong, W., Mainsah, E., Luo, N., Mathia, T., et al. (2000). *Development of methods for the characterisation of roughness in three dimensions*. London: Penton press.
70. Whitehouse, D. J., & Reason, R. E. (1965). *The equation of the mean line of surface texture found by an electric wave filter*: Rank Organisation (Rank Taylor Hobson Division).
71. Raja, J., & Radhakrishnan, V. (1979). Digital filtering of surface profiles. *Wear, 57*(1), 147-155.
72. Yuan, Y. B., Vorburger, T. V., Song, J. F., & Renegar, T. B. C. (2000). *A simplified realization for the Gaussian filter in surface metrology.* Paper presented at the International Colloquium on Surfaces
73. Kumar, J., & Shunmugam, M. S. (2006). A new approach for filtering of surface profiles using morphological operations. *International Journal of Machine Tools and Manufacture, 46*(3), 260-270.
74. ISO/TS 16610-40. (2006). *Geometrical product specifications (GPS) — Filtration — Part 40: Morphological profile filters: Basic concepts*. International Organization for Standardization.
75. ISO/TS 16610-41. (2006). *Geometrical product specifications (GPS) — Filtration — Part 41: Morphological profile filters: Disk and horizontal line-segment filters*. International Organization for Standardization.
76. Schmahling, J. (2006). *Statistical characterization of technical surface microstructure.* (Doctoral thesis), Heidelberg University. Retrieved from http://www.ub.uni-heidelberg.de/archiv/6792
77. ImageMetrology. (2013). *SPIP-3D Image Processing*. Retrieved from http://www.imagemet.com/index.php?main=products&sub=modules&id=3
78. Dolenc, M. (2004). Developing extendible component-oriented finite element software. *Advances in Engineering Software, 35*(10-11), 703-714.
79. Won, M., Stiemerling, O., & Wulf, V. (2006). *Component-based approaches to tailorable systems*. New York: Springer.
80. Booch, G., Rumbaugh, J., & Jacobson, I. (1998). *The Unified Modeling Language User Guide*. USA: Addison-Wesley Professional.
81. Bernstein, P. A., Bergstraesser, T., Carlson, J., Pal, S., Sanders, P., & Shutt, D. (1999). Microsoft repository version 2 and the open information model. *Information Systems, 24*(2), 71-98.
82. Crnkovic, I., & Larsson, M. P. H. (2002). *Building reliable component-based software systems*. London: Artech House Publishers.
83. McInnis, K. (2000). *Component-based development: The concepts, technology and methodology*: Castek Software Factory.

84. Chen, V. W., Thakur, D. S., & Leister, K. J. (1999). Systems Development Strategy: A Component Based Approach, The Architecture. *Journal of the Association for Laboratory Automation, 4*(5), 34-43.
85. Thakur, D. S., Chen, V. W., & Leister, K. J. (1999). Systems Development Strategy: A Component Based Approach, The Overview. *Journal of the Association for Laboratory Automation, 4*(5), 44-49.
86. Thakur, D. S., Chen, V. W., & Leister, K. J. (1999). Systems Development Strategy: A Component-based Approach, The Prototype. *Journal of the Association for Laboratory Automation, 4*(5), 28-33.
87. Papajorgji, P. (2005). A plug and play approach for developing environmental models. *Environmental Modelling & Software, 20*(10), 1353-1357.
88. Bobda, C. (2007). *Introduction to reconfigurable computing: Architectures, algorithms, and applications*: Springer Publishing Company, Incorporated.
89. Henning, m., & Vinoski, S. (1999). *Advanced CORBA programming with C++*. USA: Addison Wesley.
90. Reilly, D. (2006). *Java RMI & CORBA:A comparison of two competing technologies*. Retrieved from http://www.javacoffeebreak.com/articles/rmi_corba/index.html
91. Rector, B. E., & Sells, C. (1999). *ATL internals*: Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA.
92. Shaw, M., & Garlan, D. (1996). *Software architecture: perspectives on an emerging discipline*. Upper Saddle River, New Jersey: Prentice Hall Englewood Cliffs.
93. Al-Mudimigh, A., Zairi, M., & Al-Mashari, M. (2001). ERP software implementation: an integrative framework. *European Journal of Information Systems, 10*(4), 216-226.
94. Smith, B. L. (2002). Software development cost estimation for infrastructure systems. *Journal of Management in Engineering, 18*(3), 104-110.
95. McCormick, B. (1996). *Software Design and Implementation using the Real-Time Object-Oriented Modeling Language.* Paper presented at the Electrical Computer Engineering, Canadian
96. Polakovic, J., & Stefani, J. B. (2008). Architecting reconfigurable component-based operating systems. *Journal of Systems Architecture, 54*(6), 562-575.
97. Mennie, D., & Pagurek, B. C. (2000). *An architecture to support dynamic composition of service components.* Paper presented at the Proceedings of the th international workshop on component-oriented programming.
98. Ferron, J. R., Penaflor, B., Walker, M. L., Moller, J., & Butner, D. C. (1995). *A flexible software architecture for tokamak discharge control systems.* Paper presented at the Fusion Engineering Sofe "Seeking a New Energy Era".
99. Sun, W., Zhang, X., Guo, C. J., Sun, P., & Su, H. C. C. o. S. P. I. I. S.-I. (2008). *Software as a service: Configuration and customization perspectives*.
100. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design patterns: elements of reusable object-oriented software*. Upper Saddle River, New Jersey: Pearson Education.
101. ISO 25178-71. (2012). *Geometrical Product Specifications (GPS) — Surface texture: Areal Part 71: Software measurement standards*. International Organization for Standardization.
102. Hill, F. S., & Kelley, S. M. (2000). *Computer graphics: using OpenGL*. Upper Saddle River, NJ: Prentice Hall
103. Wright, R. S., & Lipchak, B. (2004). *OpenGL superbible*. Indianapolis: Sams Publishing.
104. Royce, W. W. (1970). *Managing the development of large software systems.* Paper presented at the Proceedings of IEEE Wescon.
105. Sommerville, I. (2004). *Software Engineering.* : Addison-Wesley.
106. Whitgift, D. (1991). *Methods and tools for software configuration management*: John Wiley & Sons, Inc. New York, NY, USA.

107. Boehm, B. (1986). A spiral model of software development and enhancement. *ACM SIGSOFT Software Engineering Notes, 11*(4), 14-24.
108. Stout, K., Blunt, L., Dong, W. P., Mainsah, E., Luo, N., Mathia, T., et al. (2000). *Development of methods for the characterisation of roughness in three dimensions*. London: Penton press.
109. NanoScience. (2011). *Supported File Formats*. Retrieved from http://www.nanoscience.com/products/spip/SPIP_formats.html
110. Scott, P. J. (2004). Pattern analysis and metrology: the extraction of stable features from observable measurements. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences, 460*(2050), 2845-2864.
111. Sarkar, B., Singh, L. K., & Sarkar, D. (2003). Approximation of digital curves with line segments and circular arcs using genetic algorithms. *Pattern Recognition Letters, 24*(15), 2585-2595.
112. Hu, W. C. (2005). Multiprimitive segmentation based on meaningful breakpoints for fitting digital planar curves with line segments and conic arcs. *Image and Vision Computing, 23*(9), 783-789.
113. Shirley, P., Ashikhmin, M., Gleicher, M., Marschner, S., Reinhard, E., Sung, K., et al. (2005). *Fundamentals of computer graphics*. Natick, Massachusetts: AK Peters Wellesley.
114. Cunningham, S. (2006). *Computer graphics: programming, problem solving, and visual communication*. Upper Saddle River, New Jersey: Prentice Hall.
115. WIKIPEDIA. (2012). *Comparison of OpenGL and Direct3D*. Retrieved from http://en.wikipedia.org/wiki/Comparison_of_OpenGL_and_Direct3D
116. Shreiner, D. (2009). *OpenGL programming guide: the official guide to learning OpenGL, versions 3.0 and 3.1*. Boston: Addison-Wesley Professional.
117. Martz, P. (2006). *OpenGL distilled*. Boston: Addison-Wesley Professional.
118. Astle, D., & Hawkins, K. (2004). *Beginning OpenGL game programming*. Cambridge, Massachusetts: Course Technology.
119. Hawkins, K., & Astle, D. (2001). *OpenGL game programming*. Cambridge, Massachusetts: Course Technology PTR.
120. Farin, G. E. (1996). *Curves and Surfaces for Computer-Aided Geometric Design: A Practical Code*. Waltham, Massachusetts: Academic Press, Inc.
121. MathWorks. (2012). *Set and Get current colormap*. Retrieved from http://www.mathworks.co.uk/help/matlab/ref/colormap.html
122. Schreiner, D., Woo, M., Neider, J., & Davis, T. (2006). *OpenGL programming guide*. Boston: Addison-Wesley.
123. WIKIPEDIA. (2012). *Software testing*. Retrieved from http://en.wikipedia.org/wiki/Software_testing
124. Patton, R. (2005). *Software testing*. Indianapolis: Sams Publishing.
125. Microsoft. (2013). *ActiveX Control Test Container (tstcon32.exe)*. Retrieved from http://msdn.microsoft.com/en-GB/library/ms241446(v=vs.80).aspx
126. DigitalSurf. (2013). *Mountains surface imaging & metrology software*. Retrieved from http://www.digitalsurf.fr/en/mntkey.html
127. Bui, S. H., & Vorburger, T. V. (2007). Surface metrology algorithm testing system. *Precision Engineering, 31*(3), 218-225.
128. Bui, S. H., Renegar, T. B., Vorburger, T. V., Raja, J., & Malburg, M. C. (2004). Internet-based surface metrology algorithm testing system. *Wear, 257*(12), 1213-1218.
129. Bui, S. H., Muralikrishnan, B., & Raja, J. (2005). A framework for Internet-based surface texture analysis and information system. *Precision Engineering, 29*(3), 298-306.

Appendixes

A Critical Code Fragments of SurfStand SDK

A.1 Static methods for addition and removement of system functional components

```
       public static bool AddAssembly(String fileType, String className, String
asmPath,SSAssemblyType asmType)
        {
            List<SSAssembly> asmList=null;
            switch (asmType)
            {
                case SSAssemblyType.FILEACCESS:
                    asmList = m_listCollection.m_fileAccessList;
                    break;
                case SSAssemblyType.DATADISPLAY:
                    asmList = m_listCollection.m_displayList;
                    break;
                case SSAssemblyType.DATAANALYSIS:
                    asmList = m_listCollection.m_analysisList;
                    break;
            }
            foreach (SSAssembly ssasm in asmList)
            {
                if (ssasm.m_Type.Equals(fileType))
                    return false;
            }
            SSAssembly newasm = new SSAssembly();
            newasm.m_Type = fileType;
            newasm.m_Name = className;
            newasm.m_Path = asmPath;
            asmList.Add(newasm);
            return true;
        }
        public static bool RemoveAssembly(String type,SSAssemblyType asmType)
        {
            List<SSAssembly> asmList=null;
            switch (asmType)
            {
                case SSAssemblyType.FILEACCESS:
                    asmList = m_listCollection.m_fileAccessList;
                    break;
                case SSAssemblyType.DATADISPLAY:
                    asmList = m_listCollection.m_displayList;
                    break;
                case SSAssemblyType.DATAANALYSIS:
                    asmList = m_listCollection.m_analysisList;
                    break;
            }
            foreach (SSAssembly ssasm in asmList)
            {
                if (ssasm.m_Type.Equals(type))
                {
                    asmList.Remove(ssasm);
                    return true;
                }
            }
            return false;
        }
```

A.2 Static methods of adding and removing system commands.

```
        public static bool AddCommand(String command,  String classType, bool newView,
String displayType)
        {
            if (IsCommandExist(command))
                return false;
            SSCommandBio newCommand = new SSCommandBio();
            newCommand.Name = command;
            newCommand.ClassType = classType;
            m_listCollection.m_commandList.Add(newCommand);
            return true;
        }
//
        public static bool AddCommand(String command, List<SSCommandBio> cmdList, bool
newView, String displayType)
        {
            if (IsCommandExist(command))
                return false;
            SSCCommand newCommand = new SSCCommand();
            newCommand.Name = command;

            for (int i = 0; i < cmdList.Count; i++)
            {
                newCommand.AddCommand(cmdList[i]);
            }
            m_listCollection.m_commandList.Add(newCommand);
            return true;
        }
        public static bool AddCommand(ISSCommand cmd)
        {
            if (IsCommandExist(cmd.Name))
                return false;
            m_listCollection.m_commandList.Add(cmd);
            return true;
        }
        public static bool RemoveCommand(String command)
        {
            foreach (ISSCommand cmd in m_listCollection.m_commandList)
            {
                if (cmd.Name.Equals(command))
                {
                    m_listCollection.m_commandList.Remove(cmd);
                    return true;
                }
            }
            return false;
        }
```

A.3 Static methods of adding and removing system menu items.

```
private static bool AddNewMenuItem(String menuContent, String command)
        {
            ToolStripItemCollection currentMenu =
m_mainForm.MainMenuStrip.Items;//.MenuItems;
            String menuItemText = null;
            ToolStripMenuItem addItem;
            bool IsExist = false;
            int index;
            while ((index = menuContent.IndexOf(';')) >= 0)
            {
                IsExist = false;
                menuItemText = menuContent.Remove(index);
                menuContent = menuContent.Substring(index + 1);
                foreach (ToolStripItem menuItem in currentMenu)
                {
                    if (menuItem.Text.Equals(menuItemText))
                    {
```

```
                        IsExist = true;
                        currentMenu = ((ToolStripMenuItem)menuItem).DropDownItems;
                        break;
                    }

                }
                if (!IsExist)
                {
                    addItem = new ToolStripMenuItem(menuItemText);
                    currentMenu.Add(addItem);
                    currentMenu = addItem.DropDownItems;
                }
            }
            IsExist = false;
            foreach (ToolStripItem menuItem in currentMenu)
            {
                if (menuItem.Text.Equals(menuContent))
                {
                    IsExist = true;
                    return false;
                }

            }
            addItem = new ToolStripMenuItem(menuContent);
            addItem.Name = command;
            addItem.Click += OnClickMenuItem;
            currentMenu.Add(addItem);
            return true;
        }
        public static bool RemoveMenuItem(String menuContent)
        {
            if (RemoveOldMenuItem(menuContent))
            {
                foreach (SSMenuItem smenu in SSListCollection.GetInstance().m_menuList)
                {
                    if (menuContent.Equals(smenu.Path))
                    {
                        SSListCollection.GetInstance().m_menuList.Remove(smenu);
                        break;
                    }
                }
                return true;
            }
            return false;
        }
```

B Code Fragments for the Connection between the Framework and Components

B.1 Method of loading components in the framework

```
//Loading function components
 private void loadAssemblies()
 {
     AssemblyConfiguration aCog = new AssemblyConfiguration();
     ArrayList pathList = aCog.getAssembliesPath();
     foreach (String path in pathList)
     {
          Assembly fileAssembly = Assembly.LoadFrom(path);
          if (fileAssembly == null)
                    continue;
          Type[] types = fileAssembly.GetTypes();
          foreach (Type ty in types)
          {
               Type[] interfaces = ty.GetInterfaces();
               foreach (Type inface in interfaces)
               {
                   if (inface.Equals(typeof(SSObject)))
                   {
                       SSObject a = (SSObject)Activator.CreateInstance(ty);
                       a.OnRegister();
                       break;
                    }
                }
            }
        }
  }
```

B.2 Methods of adding and removing components in the framework

```
//Add a new function component
private void buttonAdd_Click(object sender, EventArgs e)
{
    OpenFileDialog fileDlg = new OpenFileDialog();
    fileDlg.RestoreDirectory = true;
    if (fileDlg.ShowDialog() != DialogResult.OK)
        return;
    Assembly fileAssembly = Assembly.LoadFrom(fileDlg.FileName);
    if (fileAssembly == null)
        return;
    Type[] types = fileAssembly.GetTypes();
    foreach (Type ty in types)
    {
        Type[] interfaces=ty.GetInterfaces();
        foreach (Type inface in interfaces)
        {
            if (inface.Equals(typeof(SSObject)))
            {
                SSObject a = (SSObject)Activator.CreateInstance(ty);
                if (a.OnRegister())
                {
                    AssemblyConfiguration aCog = new AssemblyConfiguration();
                    aCog.AddFileAssembly(fileDlg.FileName);
                    updateAssemblies(sender,e);
                }
                break;
            }
        }
    }
}
```

```
//Remove an existing function component
private void buttonDelete_Click(object sender, EventArgs e)
{
    foreach (ListViewItem lvi in this.listViewAssembliesManage.SelectedItems)
    {
        Assembly asmbly = Assembly.LoadFrom(lvi.SubItems[1].Text);
        Type[] types = asmbly.GetTypes();
        foreach (Type ty in types)
        {
            Type[] interfaces=ty.GetInterfaces();
            foreach (Type inface in interfaces)
            {
                if (inface.Equals(typeof(SSObject)))
                {
                    SSObject a = (SSObject)Activator.CreateInstance(ty);
                    if (a.UnRegister())
                    {
                        AssemblyConfiguration aCog = new AssemblyConfiguration();
                        aCog.DeleteFileAssembly(lvi.SubItems[1].Text);
                        updateAssemblies(sender,e);
                    }
                    break;
                }
            }
        }
    }
}
```

C Code Fragments for Surface Data Definition

C.1 Definition of surface data structure CSurfObject

```
class CSurfObject
{
public:
        CSurfObject(void);
        CSurfObject(CSurfObject& m_SurfObj);
        CSurfObject(CMatrix& m_DataOther, THSurfObjInfo& m_InfoOther);
        virtual ~CSurfObject(void);

//Data members
public:
        CMatrix         m_DataSurf;
        THSurfObjInfo   m_InfoSurf;

//function members
public:
        bool CopyFrom(CSurfObject& m_SurfObj);
        bool CopyFrom(CMatrix& m_DataOther, THSurfObjInfo& m_InfoOther);
        bool CastFrom(CSurfObject& m_SurfObj);
        bool CastFrom(CMatrix& m_DataOther, THSurfObjInfo& m_InfoOther);

        bool IsValidSurfObj();
        void Clear();
};
```

C.2 Definition of surface data information THsurfObjInfo

```
typedef /* [helpstring][version][uuid] */  DECLSPEC_UUID("A43EF600-B02A-11D6-9BCF-
00D0B74C4D6A") struct THSurfObjInfo
    {
    TCHAR cFileName[ 256 ];
    TCHAR cFileExt[ 16 ];
    TCHAR cFileVersion[ 16 ];
    TCHAR cManufacture[ 16 ];
    TCHAR cCreateDate[ 16 ];
    TCHAR cModifyDate[ 16 ];
    long sNumPoints;
    long sNumProfiles;
    float fXscale;
    TCHAR cXunit[ 16 ];
    float fYscale;
    TCHAR cYunit[ 16 ];
    float fZscale;
    TCHAR cZunit[ 16 ];
    float fXOffset;
    float fYOffset;
    float fZOffset;
    boolean bMetricUnit;
    TCHAR cDescription[ 256 ];
      }        THSurfObjInfo;
```

C.3 Definition of surface data class CTH3DData

```
class ATL_NO_VTABLE CTH3DData :
        public CComObjectRootEx<CComSingleThreadModel>,
        public CComCoClass<CTH3DData, &CLSID_TH3DData>,
        public ISupportErrorInfo,
        public IDispatchImpl<ITH3DData, &IID_ITH3DData, &LIBID_TH3DDataProcessLib,
/*wMajor =*/ 1, /*wMinor =*/ 0>
{
public:
        CTH3DData()
```

```
        {
        }

DECLARE_REGISTRY_RESOURCEID(IDR_TH3DDATA)

BEGIN_COM_MAP(CTH3DData)
        COM_INTERFACE_ENTRY(ITH3DData)
        COM_INTERFACE_ENTRY(IDispatch)
        COM_INTERFACE_ENTRY(ISupportErrorInfo)
END_COM_MAP()

// ISupportsErrorInfo
        STDMETHOD(InterfaceSupportsErrorInfo)(REFIID riid);

        DECLARE_PROTECT_FINAL_CONSTRUCT()

        HRESULT FinalConstruct( )
        {
                Clean();
                return S_OK;
        }

        void FinalRelease()
        {
                Clean();
        }
    BOOL Clean();

public:
        STDMETHOD(IsDataValid)(/*[out, retval]*/BOOL* m_bValid);
        STDMETHOD(UnitSystem)(/*[in]*/BOOL m_bMetricUnit);
        STDMETHOD(MinMax)(/*[out]*/float* Min, /*[out]*/float* Max);
        STDMETHOD(Cast)(/*[in]*/ITH3DData* source);
        STDMETHOD(Copy)(/*[in]*/ITH3DData* source);
        STDMETHOD(CopyToSafeArray)(/*[out]*/THSurfObjInfo *pDataInfo, /*[out]*/VARIANT
*pDataMatrix);
        STDMETHOD(CopyFromSafeArray)(/*[in]*/THSurfObjInfo *pDataInfo, /*[in]*/VARIANT
*pDataMatrix);
        STDMETHOD(CopyFromVariant)(/*[in]*/THSurfObjInfo *pDataInfo, /*[in]*/VARIANT
*pDataMatrix);
        STDMETHOD(CastFromVariant)(/*[in]*/THSurfObjInfo *pDataInfo, /*[in]*/VARIANT
*pDataMatrix);
        STDMETHOD(CastToVariant)(/*[out]*/THSurfObjInfo *pDataInfo, /*[out]*/VARIANT
*pDataMatrix);
        STDMETHOD(get_DataInfo)(/*[out, retval]*/ THSurfObjInfo *pVal);
        STDMETHOD(put_DataInfo)(/*[in]*/ THSurfObjInfo newVal);
        STDMETHOD(HasMissingData)(BOOL* m_bMissingData);
        STDMETHOD(GetThumbnailData)(SHORT nWidth, SHORT nHeight, BYTE** pbData);
        STDMETHOD(ThumbnailDataToSafeArray)(LONG m_nWidth, LONG m_nHeight, VARIANT*
pDataMatrix);
        STDMETHOD(ShiftX)(LONG m_offsetX);

        CSurfObject    m_objSurf;
public:
        STDMETHOD(CheckSum)(LONG* CHKSum);
        STDMETHOD(ReleaseData)(void);
        STDMETHOD(Subtract)(ITH3DData* subtracter);
        STDMETHOD(Addition)(ITH3DData* addend);
};
```

D Definition of Command Interface: ISSCommand.

```
public abstract class ISSCommand
    {
        protected String m_Name;
        protected bool m_defaultArgs;
        protected List<SSArgument> m_argList;

        protected Object m_Tag = null;
        public Object Tag
        {
            get { return m_Tag; }
            set { m_Tag = value; }
        }
        public String Name
        {
            get { return m_Name; }
            set { m_Name = value; }
        }

        public bool DefaultArgs
        {
            set { m_defaultArgs = value; }
            get { return m_defaultArgs; }
        }
        public List<SSArgument> ArgList
        {
            get { return m_argList; }
        }
        public abstract bool Execute(List<SSData> operands);
        public abstract void ModifyArgument();
    }
```

E Code Fragments for Method ModifyArgument of Feature Parameters Component

```
public void ModifyArguments(ISSCommand command)
 {
     //THDataSegmentLib.THFeatureParas
     ParameterSetting ps = new ParameterSetting();
     ps.m_reset = command.DefaultArgs;
     ps.m_bHillFeatures = (int)command.ArgList[0].Value;
     ps.m_fWolfPrune = (float)command.ArgList[1].Value;
     ps.m_fVolumePrune = (float)command.ArgList[2].Value;
     ps.m_fAreaPrune = (float)command.ArgList[3].Value;
     ps.m_fCircumferencePrune = (float)command.ArgList[4].Value;
     ps.ShowDialog();
     command.DefaultArgs=ps.m_reset;
     command.ArgList[0].Value = ps.m_bHillFeatures;
     command.ArgList[1].Value = ps.m_fWolfPrune;
     command.ArgList[2].Value = ps.m_fVolumePrune;
     command.ArgList[3].Value = ps.m_fAreaPrune;
     command.ArgList[4].Value = ps.m_fCircumferencePrune;
 }
```

F Interface Definitions for the Three Subtypes of Data Process Components

```
    public interface SSAnalysisA : SSAnalysis  //bioA
    {
        bool Execute(SSData pSrc, ref SSData pOut,ref bool defaultArgs, List<SSArgument>
argList);
    }
    public interface SSAnalysisB : SSAnalysis
    {
        bool Execute(SSData pSrc1, SSData pSrc2, ref SSData pOut, ref bool defaultArgs,
List<SSArgument> argList);
    }
    public interface SSAnalysisC : SSAnalysis   //single
    {
        bool Execute(SSData pSrc,ref bool defaultArgs, List<SSArgument> argList);

        object GetDisplayCtrl();
    }
```

G Related Publications

Lan, X., Jiang, X., Blunt, L., & Xiao, S. (2009). *An adaptive system framework for surface characterisation*. Paper presented at the The Proceedings of Computing and Engineering Annual Researchers' Conference 2009, Huddersfield.

Lan, X., Jiang, X., Blunt, L., Xiao, S., & Zeng, W. (2011). *A feasible way to analysis microstructures on a surface based on the extraction and construction of geometrical features.* Paper presented at the The 10th International Symposium on Measurement Technology and Intelligent Instruments, Daejeon, S. Korea.