Function-based web visualisation

Liu, Qi

2007

Liu, Q. (2007). Function-based web visualisation. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/2492>

<https://doi.org/10.32657/10356/2492>

Nanyang Technological University

Downloaded on 26 Apr 2023 22:06:29 SGT

Nanyang Technological University

Function-based Web Visualisation

Liu Qi

A thesis submitted to the Nanyang Technological University
in fulfillment of the requirement for the degree of
Doctor of Philosophy

**School of Computer Engineering
Nanyang Technological University
Singapore 639798**

March 2007

Statement of Originality

I hereby certify that the content of this thesis is the result of work done by myself and has not been submitted for a higher degree to any other University or Institution.

.....

Date

.....

Signature

Abstract

One of the 3D web visualisation methods is based on model transmission where 3D objects are rendered at the client computers while the models are transmitted over the Internet together with other communication data. However, when large models are transmitted, it may be prohibitive for efficient web visualisation due to the bandwidth limitation of Internet. This deficiency can be eliminated by employing function-based shape modelling where complex geometric objects are defined by relatively small formulae.

A hybrid function-based shape modelling framework unifying different classes of mathematical functions, is proposed in this thesis. Parametric, implicit and explicit functions can be used concurrently for defining geometry, appearance and transformations of shapes. When modelling a geometric shape with the proposed approach, the geometry and appearance of the shape are defined in their respective domains and then merged into the final shape.

The general function-based shape rendering pipeline is designed. It defines the overall structure and particular modules that have to be available in each software implementation. In the proposed framework, geometric shapes, colours and geometric textures are defined by functions. The proposed framework allows for its extension with any proprietary function-based models.

A web-based interactive application framework is proposed for developing different collaborative applications. Because of the small size of function-based models, the models on client computers can be synchronised efficiently. With the framework, it is possible to use

the function-based models together with large data sets coming from data acquisition devices such as CT, MRI, or range scanners.

The proposed approach was implemented as an extension of Virtual Reality Modelling Language (VRML) and Extensible 3D (X3D). The extension, FVRML/FX3D, allows for defining time-dependent geometric shapes, their appearances, and operations by parametric, implicit and explicit functions which can be used concurrently. FVRML/FX3D opens VRML and X3D to practically any type of geometry, 3D colours and geometric textures. It also introduces to VRML and X3D set-theoretical operations (union, intersection, difference) as well as allows for including any other operations which can be defined analytically or procedurally. FVRML/FX3D allows for a great reduction of the model size and provides an unlimited level of detail. The function-defined shapes can be used together with the standard VRML and X3D shapes and appearances.

The implementation is done by extending the existing browsers using the Script node. Compared to other implementation methods, it provides both speed and portability. FVRML/FX3D can be easily amended by replacing any of the components of the visualisation pipeline. To improve the efficiency which is essential for time-dependent models, mesh caching and JIT technology are exploited.

The FVRML/FX3D extension is illustrated by its application to teaching computer graphics and as a part of the projects Virtual Campus of NTU and Collaborative Virtual Shape Modelling Laboratory. To illustrate potentials of FVRML/FX3D for practical applications, a web-based collaborative function-based shape modelling tool is created based on the extension. The interactive tool is used in several real-life application projects. Other examples of applying FVRML/FX3D include development of functions interpolating volume data, scattered points, and NURBS surfaces.

Acknowledgements

First of all, I would like to thank my supervisor Associate Professor Alexei Sourin for his invaluable instructions. I am grateful to his help throughout my graduate study in Nanyang Technological University. He helps me to understand basic ideas, and to correct theoretical findings. Without his efforts, ideas and comments, the project could not be completed.

I would also like to thank Nanyang Technological University and School of Computer Engineering for providing me the opportunity and financial support to further my study and research. Special thanks to Centre for Advanced Media Technology for providing me the first class equipments and technical supports.

Last but not least, I would like to thank my friends, Miss Jing Wang, Mr. Konstantin Levinski, Mr. Chen Chen, Mr. Yu Yu, and Mr. Yimin Wang, for their precious support. Furthermore, I wish to thank my parents. Their selfless concern and encouragement are the best support during my postgraduate study.

Contents

Abstract	i
Acknowledgements	iii
Contents	iv
List of Figures	ix
List of Tables	xii
1 Introduction	1
1.1 Motivation	1
1.2 Objective	3
1.3 Proposed approach	4
1.4 Organisation	6
2 Web visualisation and function-based modelling	7
2.1 Web visualisation approaches	7
2.1.1 Image transmission	9

CONTENTS

2.1.2	Event transmission	10
2.1.3	Model transmission	12
2.2	3D model transmission standards and applications	16
2.2.1	Universal 3D	17
2.2.2	Collaborative Design Activity	18
2.2.3	Moving Picture Experts Group	19
2.2.4	Java 3D	19
2.2.5	Virtual Reality Modelling Language	20
2.2.6	Extensible 3D	24
2.2.7	Applications	26
2.2.8	Problems of 3D model transmission	27
2.3	Function-defined shape modelling approaches	28
2.3.1	Representation	28
2.3.2	Rendering	38
2.4	Function-based web visualisation	43
2.5	Summary	44
3	Hybrid function-based web shape modelling framework	46
3.1	Function-based shape rendering pipeline	46
3.2	Function-defined shapes and properties	48
3.3	Modelling framework	51
3.4	Example	55

CONTENTS

3.5	Summary	57
4	Function-based extension of VRML/X3D	59
4.1	Selection of the implementation platform	59
4.2	Extension scope	62
4.3	Function definition	66
4.3.1	Analytical formulae	67
4.3.2	Function scripts	70
4.4	Prototypes	71
4.4.1	FVRML prototype specifics	74
4.4.2	FX3D prototype specifics	77
4.5	Summary	79
5	Implementation of FVRML/FX3D	86
5.1	General considerations	86
5.2	Modules and visualisation pipeline	88
5.3	Ways to extend VRML/X3D	92
5.3.1	External API	92
5.3.2	Browser-specific API	93
5.3.3	Script node	94
5.4	Development platform	95
5.4.1	Xj3D and Java SAI	96

CONTENTS

5.4.2	BS Contact and native script	97
5.5	Performance optimisation and analysis	98
5.5.1	Function evaluation optimisation	99
5.5.2	Mesh caching	101
5.6	Summary	104
6	Extensibility of FVRML/FX3D	106
6.1	Overview	106
6.2	Module replacement	107
6.3	Function script add-on	110
6.3.1	Custom function in function scripts	110
6.3.2	Native function in function scripts	112
6.4	Summary	113
7	Web-based collaborative modelling tool	115
7.1	Collaborative modelling	115
7.1.1	Concurrency	116
7.1.2	Synchronisation	118
7.1.3	Connection	119
7.2	Interactive FVRML/FX3D application framework	120
7.3	Development platform	121
7.4	Implementation of the interactive collaborative modelling tool	123

CONTENTS

7.5	Platform-specific issues	131
7.6	Summary	132
8	Examples of using FVRML/FX3D	133
8.1	Static models	133
8.2	Time-dependent models	140
8.3	Other function-defined models	146
8.4	Creating models interactively	151
8.5	Working with hybrid function-based models	155
8.6	Summary	163
9	Conclusion and future work	165
9.1	Conclusion	165
9.2	Future work	168
9.2.1	Advanced modelling tools	168
9.2.2	Fast polygonisation	169
9.2.3	Haptic device integration	171
	Acronyms	172
	Author's Publications	173
	Bibliography	175

List of Figures

3.1	Function-based shape rendering pipeline and mapping functions	47
3.2	FRep and parametric objects	52
3.3	Mapping colours and objects	52
3.4	Framework of hybrid function-based shape modelling	53
4.1	Scene diagrams of the function-based extension	72
4.2	Prototypes of the FVRML	75
4.3	Prototypes of the FVRML (continue)	76
4.4	Prototypes of the FX3D	81
4.5	Prototypes of the FX3D (continue)	82
5.1	The visualisation pipeline of the implemented extension	89
7.1	GUI of the modelling tool	126
7.2	Definition of the custom tool by its implicit formula and removing the material with this tool	128
7.3	Colouring with different tools and blending values	129
7.4	Exporting the design to FVRML source code	129

LIST OF FIGURES

8.1	FRep shape and FRep texture	134
8.2	FRep shape and parametric texture	136
8.3	Parametric shape and FRep texture	137
8.4	Parametric shape and parametric texture	138
8.5	FRep object with FRep colour	139
8.6	FRep object with parametric colour	141
8.7	Parametric object with FRep colour	142
8.8	Parametric object with parametric colour	143
8.9	Union of two cylinders with blending	144
8.10	Geometry animation	145
8.11	Geometric texture animation	147
8.12	Colour animation	148
8.13	Combined geometry, colour and geometric texture animation	149
8.14	Shapes created with an interactive function-based shape modelling programme and imported into VRML/X3D for viewing and further modification with the function-based extension	150
8.15	Blobbies using custom function in function script	152
8.16	Bunny using trilinear interpolation of volume data with modifications and colouring	153
8.17	Example of web-based interactive shape modelling	154
8.18	Interactive make-up of the NURBS head	155
8.19	Proposed workflow for hybrid function-based modelling technique	156

LIST OF FIGURES

8.20 The Stanford bunny model	158
8.21 Modelling a function-based bunny	159
8.22 Model modification in web-based modelling tool	160
8.23 Bunny created according to the workflow together with standard VRML objects	162
8.24 Human brain coloured according to density	163

List of Tables

4.1	FVRML/FX3D field definitions	83
5.1	VRML source file size (in bytes)	100
5.2	Rendering time for gzipped files (in seconds)	101
5.3	Performance of mesh caching	104

Chapter 1

Introduction

In this chapter, a brief introduction to this project is given. In Section 1.1, the motivation of the project is presented, including brief discussion of deficiencies in VRML/X3D and function-based shape modelling. Section 1.2 demonstrates the problems that this thesis solves. Brief explanation of the approaches proposed in this thesis is illustrated in Section 1.3. Finally in Section 1.4, the organisation of this thesis is given.

1.1 Motivation

Visualisation is the process of converting data (e.g. engineering, business, scientific data, etc.) into visible elements (e.g. texts, images, charts, tables, etc.). The visible elements are abstraction of numerous data, which can give the reader complex information and/or conclusion. Because of the emergence of the personal computer and the Internet, visualisation can now employ the Internet and computers as media. The term web visualisation describes the visualisation process involving both the Internet and computers. Web visualisation is normally performed in several steps. First, the data is acquired by some methods, such as experiments, surveys, etc., either by hand or automatically by a computer. Then, the acquired data is stored on a server, which provides the visualisation service. Following this, the data is delivered through the Internet either in a processed or unprocessed form to the

Chapter 1. Introduction

client computer where the viewer wants to obtain the visualised result. Finally, the visible elements are displayed on the screen of the client computer. At the beginning, texts were the only visible elements for web visualisation due to the limited ability of hardware of the client computer and the limited bandwidth of the Internet. However, rapid development of personal computer hardware has greatly changed this situation. With modern display adaptors and fast CPUs, images, animations, interactive components and even three-dimensional virtual worlds have been introduced in web visualisation. Despite the development of computer hardware, the limitation of the Internet bandwidth becomes significant, since the transmitted enhanced components of web visualisation are usually large in size. Especially for three-dimensional virtual worlds, it usually takes up to several megabytes to model a complex object if common model representation approaches are employed. Therefore, the large file size and limited bandwidth of the Internet delay further development and usage of web visualisation.

Due to the existing conflict between large transmitted data and limited bandwidth of the Internet, two solutions can be proposed: increasing the bandwidth of the Internet, or reducing transmitted data. From technical point of view, increasing the bandwidth of the Internet is readily available, since network bandwidth can now reach over one gigabytes per second based on existing technologies. Nevertheless, it is unaffordable to upgrade the whole the Internet in order to significantly increase the bandwidth between any two peers. Therefore, the only practical choice is to reduce the transmitted data. Having been motivated by this, the project focuses on proposing a practical method for web visualisation, especially for visualisation of three-dimensional objects over the Internet, which reduces transmitted data without losing quality of final visualised result.

1.2 Objective

The function-based shape modelling techniques are becoming increasingly popular in the computer graphics area. The rationale of function-based shape modelling is that any shape, no matter how complex it is, together with its visual properties (e.g. colour, geometric texture, opacity, and shininess), and other properties (e.g. temperature, friction index, and hardness) can be defined by mathematical functions. All types of functions can be employed to define objects. Compared to polygonal meshes, which usually contain hundreds or even thousands of polygons, and other models such as point and voxel representations, function-based models, defining exactly the same object, are much smaller in terms of file size. Furthermore, since function-based models preserve all geometric details, the model can be magnified to any level of detail without loss of quality. Because of the advantages that function-based shape modelling has, it may give a boost to web visualisation. Thus, the basic idea of the project is to employ functions to define models in web visualisation.

Function-based shape modelling has many areas for research. First, the rendering hardware available nowadays does not support rendering function-based models natively. Many visualisation methods have been proposed to render function-based models. There are three major categories of visualisation methods designed for function-based models, including ray tracing, polygonisation followed by polygon rendering techniques, and point sampling followed by point rendering techniques. However, the incompatibility of three different types of mathematical functions (i.e. implicit functions, explicit functions, and parametric functions) greatly limits the usage of function-based shape modelling. Usually, for any given shape representation method, only one certain property representation method can be employed. Thus, the first objective of the project is to propose a framework for web visualisation using functions as model representation, which solves the incompatibility analysed above. Because

web visualisation can be done in many ways, the proposed framework must not be limited to one specific platform or implementation.

Besides the proposal of the function-based web visualisation framework, it is very important to verify its practicability. The proposed framework for function-based web visualisation has to be implemented on at least one platform. Thus, the second objective is designing and making several implementations based on different platforms and environments. The implementation may only support a few functional representations at first, but should be able to extend to virtually any functional representation if required.

Based on the implementations, certain web-based application must be created in order to illustrate the usefulness of the function-based models in web visualisation. Such application should be able to show the fact that the proposed function-based modelling framework can model both fascinating unreal objects, as well as objects coming from the reality.

1.3 Proposed approach

To achieve the objectives, the following aspects should be considered: a robust function-based web shape modelling framework, a complete function-based shape visualisation pipeline designed for the framework, several practical and useful implementations, and one web-based application using the framework implementation.

The key point of the project is to propose a function-based web shape modelling framework, which allows for describing shapes, visual properties and other properties by functions. In a robust function-based shape modelling framework, the functions should be able to be defined by any function representations, such as evaluation procedures or analytical formulae. At present, existing frameworks with the ability of defining shapes and visual properties can only employ one certain type of function, e.g. both the shapes and the visual properties can only be defined by explicit functions. This deficiency greatly limits the usage of function-

Chapter 1. Introduction

based shape modelling, since sophisticated colours or textures designed by one modelling method can not be adopted for other modelling framework easily. Thus, in our function-based shape modelling framework, such problem should be efficiently solved. By proposing the *hybrid function-based shape modelling framework*, one can employ different types of functions concurrently to define shapes as well as other properties.

Because visualising any function-based shape is not straightforward as visualising polygonal meshes, a special visualisation pipeline must be designed for the framework. Since our goal is to employ different types of function representation methods to model objects, the architecture of the visualisation pipeline should be open. Each component in the visualisation pipeline should be able to be replaced independently without the influence of other components.

In order to verify the practicability of the framework as well as the visualisation pipeline, certain implementations for web visualisation have to be done. The implementations fully implement the framework and the visualisation pipeline. As the project is for web visualisation, the implementation must allow users to freely exchange and construct 3D models using function-based shape modelling techniques through the Internet. The implementation should also be built on a widely accepted and used standard, such as the Virtual Reality Modelling Language [1] and its successor Extensible 3D [2]. By using different types of functions, shape modelling can be done in a more convenient way. Due to the nature of function-based shape modelling, the generated objects are compact and have small file size. Thus, one can concentrate on the creation of objects without worrying about optimisation. Finally, the implementation must be easy to learn and use. It should not introduce a new programming language or a new syntax.

Besides the framework implementations, an interactive web-based application is also necessary to show the benefits introduced by the function-based shape modelling framework. Since it is sometimes difficult to create function-based models, an interactive web-based

modelling tool would be necessary. It may even hide the detail of the mathematical background of the function-based shape modelling framework, while still using the framework as the back-end. Thus, it can attract more authors to create function-based models without learning the details of the function-based shape modelling concept.

1.4 Organisation

This thesis is organised as follows. A survey of present web visualisation techniques and function-based shape modelling techniques is presented in Chapter 2. A hybrid function-based web shape modelling framework, which is able to use different representation of functions concurrently to describe several properties of a model, is described in Chapter 3. In Chapter 4, a function-based shape modelling extensions of VRML and X3D are illustrated. The implementation details and examples are also discussed and demonstrated in Chapter 5. The ability of introducing additional function representations to the extensions of VRML and X3D is shown in Chapter 6. In Chapter 7, the web-based collaborative modelling tool is presented. Hybrid function-based models rendered by the extensions, as well as objects made by the web-based collaborative modelling tools can be seen in Chapter 8. Finally, in Chapter 9, conclusions are summarised, and the future work is discussed.

Chapter 2

Web visualisation and function-based modelling

In this chapter, a comprehensive survey of web visualisation and function-based shape modelling is provided. In Section 2.1, web visualisation techniques are classified into three different classes according to the data transmission over the Internet. Commonly used standards for model transmission over Internet, are analysed in Section 2.2. The problems of existing model transmission schemes are also analysed. Current function-based representation methods as well as existing visualisation techniques of function-defined shapes are reviewed in Section 2.3. Finally, the existing function-based web visualisation approaches are discussed in Section 2.4.

2.1 Web visualisation approaches

The Internet has been rapidly growing for the last two decades. It is now the largest distributed information repository in the world. However, since humans can not acquire information from computer data directly, web visualisation becomes more and more important. Various web visualisation methods have been developed to enhance users' experiences.

At first, web visualisation only involves visualising texts. The texts are small in terms of

Chapter 2. Web visualisation and function-based modelling

size, as at most two bytes are needed for one single character of any language existing on earth. Because of this, the transmitted data over the Internet can just be the final result of the visualisation process. HTML was the first international standard for delivering visualised texts, as well as hyperlinks, which indicate the relationship between different texts. Due to the rapid evolution of the Internet, as well as the hardware development of personal computers, transmitting and visualising images and other interactive components such as Macromedia Flash become possible. This improvement happens because images and interactive components can contain more information than pure texts, and enhance the web visualisation experiences with simple participations.

Visualising and manipulating three-dimensional scenes on the web are always tempting, as the real world is actually three-dimensional. Compared to two-dimensional elements such as texts and images, three-dimensional virtual worlds are more familiar to viewers. Thus, from navigating through a three-dimensional virtual worlds and manipulating three-dimensional objects, viewers will gain knowledge more precisely and understand faster. Many methods have been developed towards the destination of providing realistic 3D visualisation over the Internet.

Though there are many approaches to 3D visualisation over the network, the procedures of these approaches, which can be called visualisation pipelines, are rather the same. A typical 3D visualisation pipeline over network is proposed by Upson et al [3]. This visualisation pipeline contains three processes, which are filtering, mapping and rendering. Filtering process filters the data being visualised by eliminating the unnecessary part and the noises, in order to make the data more meaningful and less voluminous. Mapping process is the core of the visualisation pipeline. According to the filtered data, it generates objects composed by primitives, such as points, polygons, parametric functions, implicit functions, voxels, etc., which can be rendered. Finally, in the rendering process, the objects, which are mapped from the filtered data, are converted into pictures, which are the final results and visible to viewers.

Chapter 2. Web visualisation and function-based modelling

Classification of 3D visualisation over network approaches can be done according to the transmission occurrence in the pipeline. However, as the filtering process eliminates unwanted data and reduces the size of data, and transmitting redundant data over network is unreasonable and unnecessary under any circumstance, it is obvious and logical that the filtering process should always be done on the server side. Therefore, 3D visualisation over network schemes can be categorised into three different classes: *image transmission*, *event transmission* and *model transmission*. Similar classification can be found in [4]. Nevertheless, the classification done by Bender et al. is not complete as image transmission is missing.

2.1.1 Image transmission

Servers usually have higher performance than clients. Thus, all processes in the visualisation pipeline can be done solely at server side. In this case, the transmission of data occurs after rendering process, and the transmitted data are rendered images, so this approach can be called image transmission. By employing the powerful hardware of servers, it is possible to use algorithms such as ray tracing and radiosity. These algorithms can produce high quality photo-realistic images. Furthermore, since images can be integrated as a part of HyperText Markup Language [5] at present, this approach can be implemented even on handheld computers and mobile phones with standard HTML browsers, since there are almost no special software requirements [6]. In addition to static images, certain interactivity can be done by feedback mechanisms built in HTML browsers, such as scripts, forms, buttons and hyperlinks. However, image transmission is not suitable when immediate responses are required, such as smooth navigation through a virtual world and free manipulation of virtual objects. This is caused by heavy demands of bandwidth when transmitting the 2D images. Thus, from this point of view, image transmission is not an actual 3D visualisation approach, but a method, which generates several snapshots of 3D virtual worlds by the feedbacks of viewers.

When the network bandwidth is large enough and the network delay is low, such as on

Chapter 2. Web visualisation and function-based modelling

local area network, image transmission could be used as a feasible approach for interactive rendering on lightweight devices, such as PDAs or smartphones. As in [6], a network rendering framework for handheld devices is proposed. In such framework, a cluster of computers on LAN grouped as a Grid is used as the rendering server. When a handheld device is connected to the rendering server via wired or wireless network, the model is rendered on the Grid using a default viewpoint. The rendered image is transferred back to the handheld device. If the viewpoint is changed on the handheld device, the rendering server is notified by such event, and re-render the whole scene.

Because image transmission approaches can not achieve immediate responses on the Internet due to its limited bandwidth and high delay, these schemes are usually used as complementary methods to other visualisation approaches. One such hybrid systems was developed by Engel et al. [7] using Java/Java3D. The system proposed has three working modes, which are local rendering, remote rendering, and hybrid rendering. The system assumes that local computers have limited 3D hardware-accelerated support, and the remote server has full 3D hardware-accelerated support including advanced 3D techniques such as 3D texturing ability. When local rendering is used, low quality images are rendered with interactive manipulation. When remote rendering is used, high quality images are generated and transferred to the local computer with no immediate manipulation. The hybrid rendering is the combination of the local and remote rendering. It manipulate the data interactively using local rendering, while generating high quality images using remote rendering when required.

2.1.2 Event transmission

Traditionally, the processing power of personal computers can be ignored compared to powerful servers. However, because of the rapid development of computer hardware, the computational power of clients increased dramatically. Nowadays, off-the-shelf personal computers are commonly equipped with 3D acceleration hardware. Therefore, it is now reasonable to

Chapter 2. Web visualisation and function-based modelling

utilise the processing power of clients. In the extreme, the whole 3D visualisation pipeline can be placed on the client side. In this case, the transmission of data occurs before the mapping process, and only necessary filtered data are transmitted, so this approach can be called event transmission. In our definition, events can be an event type together with some data as its parameter. Obviously, since all the rendering is done locally, viewers can get an immediate response even for very complex manipulations. Nonetheless, since clients are still not as powerful as servers, only limited rendering algorithms can be used, such as fast polygon rendering, which is the current standard for consumer display adaptors, and simple ray casting without reflections and global illumination, which is very useful in medical visualisation. Though many new techniques are developed to increase the reality, the image quality is still not as good as rendering on the server side. Since only events are transmitted over the network, the models must be pre-installed on the client computer, together with the rendering tool. For different applications, models are rarely the same, which causes viewers to install one program for each different application. Furthermore, when pre-installed models have to be changed, such as adding new models, a new patch must be downloaded and installed on every client. Therefore, this method is not suitable for common rapid Internet publishing.

Since event transmission can achieve interactive responses for complex manipulations and fairly good image quality, it is very popular in 3D network games, such as America's Army [8]. In America's Army, all the models, including persons, weapons, buildings and terrain, are pre-installed on the client. Though some weapons may not be available to the player at the beginning, they exist on the client side, instead of being delivered when playing. During the visualisation, only model movements and orientations, firing, and other small packages are transferred over network, while the game engine deals with all data processing and rendering. Here, server merely acts as a data exchanger and a filter, which eliminates the error data and possible cheaters.

Event transmission is also very useful in medical image reconstruction and visualisation.

Chapter 2. Web visualisation and function-based modelling

Though raw medical image data are usually as large as tens of megabytes, delivering raw volume data actually reduces the network traffic because visualising medical data often requires very complex operation on the whole volume, such as window level adjustment, window width adjustment, and surface extraction. If model transmission or image transmission is used, for each operation on the data volume, at least the modified part of the data volume has to be transmitted over the network. If many operations are done, the whole transmission time and delay will be much greater than to transmit data volume at the beginning. Qi et al. [9] developed a web-based medical visualisation system according to the idea of event transmission. Their work is based on Java Applets, which can be embedded into web pages. Due to the Java security problem, the visualising data have to be uploaded to the server first. When the Applet is loaded, data would be downloaded to the client and the visualisation pipeline is triggered.

Event transmission can also be adopted when the model is very simple or very small. One of the examples was represented by Malandrino et al. [10], in which a specialised visualisation system called 3WPS is developed. 3WPS is designed to remotely monitor and control the system processes by a 3D objects in a virtual world. The server is responsible to retrieve the process information and user information from the monitored machine, and then to send them to the client for visualisation. The client is a Java Applet, which converts the information into VRML scene according to a program, which is written in a special programming language JESS. The models used in the system are basic VRML elements (e.g. cubes, spheres, etc.).

2.1.3 Model transmission

In order to avoid installation of large client-side software, models, which take the most space in the client-side software, are removed. For each visit, models are downloaded from the server. Therefore, the visualisation pipeline is distributed between both the client and the server. In this case, the transmission of data occurs after the mapping process. Because the

Chapter 2. Web visualisation and function-based modelling

transmitted data are mainly models, this approach can be called model transmission. Similar to event transmission, the rendering stage is totally placed on the client side, so viewers can interactively navigate through the virtual world or manipulate objects with an immediate response. Since the models are removed, this approach only requires relatively small client rendering software, which can be easily downloaded, installed and upgraded if necessary. In addition, since only limited rendering techniques are available at present, for different applications the rendering parts are almost the same. Therefore, in many cases, one piece of rendering software can be used by many applications.

Despite the advantages of this approach, complex models can be large and it may require an unreasonably long time to download the model before its viewing and manipulating can be done. Research work has been done in order to shorten this waiting time. Briefly, all these approaches can be classified into progressive delivery and model compression.

Progressive delivery is based on a multiresolution technique. It first delivers an initial model, which is coarse, small in size, and able to be delivered in a very short period of time. When the initial model is successfully delivered, viewers can immediately commence navigation and manipulation, meanwhile the server continues delivering refined model data in a background. The refined model data usually only contains some instructive information, which is able to gradually construct a refined model from the initial model, not the whole refined model. By this coarse-refine procedure, viewers can start working also immediately without losing the details since the refined model will be available when the refining data is complete. Though network overheads do exist in progressive delivery, as only the difference of the coarse model and the refined model is transmitted in addition to the initial model, progressive delivery is practical for visualisation over the Internet.

The main problems in progressive delivery are how to construct a good initial model and how to deliver and construct the refined model from the initial model. Many methods were proposed for different model representation methods in this direction. A progressive deliv-

Chapter 2. Web visualisation and function-based modelling

ery method for polygonal meshes with controls similar to the Quality of Service (QoS) was developed by Chen and Nishita [11]. In their approach, the initial model is formed by mesh simplification of removable vertex selection and half edge collapse. To provide a recognisable model in removable vertex selection, preservation of the sharp edges and corner vertices is done by a priority queue, in which the sharp edges and corner vertices have higher priority. For each vertex removal and half edge collapse process, vertex coordinate and collapsed edges are stored as patches, which are used as information to reconstruct the original model. When transmitting the data over the Internet, a feedback mechanism similar to the QoS protocol is employed to automatically adjust the quality of refined model based on the bandwidth. A web visualisation framework called WebCAME using a similar approach was presented by Grabner [12]. Besides polygonal meshes, an approach of progressive delivery for point-based representation – Streaming QSplat – is also available in [13]. In their system, sparsely sampled points are transmitted first as the initial model, which is rendered using large splats. Gradually, higher resolution points are transmitted and rendered at client side.

Model compression means that certain compression is to be done during the transmission in order to shorten the delay. On the server side, models are compressed, which makes the file size small, and then transmitted to the client. On the client side, models are decompressed and visualised. Before the delivery of compressed models is complete, viewers can neither do navigation nor manipulation, which is like the traditional model transmission schemes. For model compression, usually lossless compression schemes are required to preserve the original look of the models. In model compression, computational overheads exist on both servers and clients. However, many computationally inexpensive compression schemes have been developed. Compared to the visualisation cost, the overheads are acceptable. There are many compression schemes, which are able to be employed in model compression. Together, they can be classified into general compression schemes and polygonal mesh compression schemes.

Chapter 2. Web visualisation and function-based modelling

General compression schemes are compression approaches, which can be used for any kind of binary data, such as audio, executable file, video, and of course, any 3D model representations. Gzip algorithm, which is now adopted as a part of the Hypertext Transfer Protocol [14] and VRML/X3D, and ZIP algorithm are typical general compression schemes. These general compression schemes can reach quite high compression ratio and quite fast speed. However, because they are universal and no specific properties of models are exploited, they are not the best compression schemes for model compression.

Since polygonal meshes are the most popular model representation at present, and polygonal meshes are usually large in size for complex models, special compression schemes have been developed for them. When these special compression schemes are adopted in model compression, it can be called mesh compression. The concept of mesh compression is pioneered by Deering [15]. In his approach, the compression scheme consists of geometry compression, position compression, colour compression, and normal compression. Triangles are considered as the only possible primitive. In geometry compression, the triangle meshes are converted into an efficient linear strip form called generalised triangle mesh. The compression is largely done by linearising triangle meshes into triangle strips since for general triangle mesh, three vertices are needed to form a triangle. For a triangle strip, only one vertex is needed to form a triangle except for the first triangle which needs three. By this method, two vertices are compressed without any loss. However, as streamed data can not provide random access, a stack structure, which stores the re-used vertices in the triangle stripes, is adopted in a generalised triangle mesh. For position, colour, and normal compression, Huffman compressed delta encoded data are exploited. Based on this framework, many improvements for triangle meshes were presented [16, 17, 18, 19, 20, 21, 22]. Though compression schemes for triangle meshes can be used for polygonal meshes together with triangulation, approaches [23, 24, 25] for polygonal meshes without triangulation have been

Chapter 2. Web visualisation and function-based modelling

proposed in recent years, since triangulation sometimes becomes a tough job for polygonal meshes when some of the polygons are not convex.

2.2 3D model transmission standards and applications

Usually, web contents are frequently changed, as the Internet is considered to be the biggest media in the world. It is unlikely that the 3D models published on the web would remain unchanged for a very long time. Therefore, the event transmission approaches are not suitable for general 3D web publishing purpose. In addition, when the content is published on the Internet, usually it is expected to be accessed tens to hundreds of times simultaneously. If image transmission is used, even if enough bandwidth is provided by the Internet service provider, it would need a large rendering cluster in order to support such concurrent network traffic. Based on these considerations, model transmission would be the most feasible 3D web visualisation approach, and it is widely used and accepted nowadays.

As explained in Section 2.1, the model transmission scheme transfers the whole model from the server to the client, including the scene structures, shapes, textures, and the behavioural information if required. The model is then rendered on the client side. The model transmission scheme itself does not impose restrictions on the modelling method and the rendering method. Therefore, the model being transmitted may either be a static model (i.e. a model remains the same across different users and access time), or a dynamic model (i.e. a model changes across different users and/or access time). The model itself can use any arbitrary modelling method, no matter whether it is a polygon-based model, point-based model, voxel-based model or function-based model. The rendering method on the client can also be different, such as using OpenGL, Direct3D or even pure software rendering.

Many international standards are built based on the model transmission scheme, since it is possible to design one model transmission scheme suitable for various rendering en-

Chapter 2. Web visualisation and function-based modelling

vironments. In this section, some commonly seen 3D model transmission standards and applications are discussed.

2.2.1 Universal 3D

Universal 3D [26], which is also known as Acrobat3D, is a file format designed to exchange 3D Computer-Aided Design objects and data. The objects defined in U3D files can be modified or visualised by 3D modelling software. Besides the common functionality, such as defining static shapes and scene structure, U3D also contains key-frame and bone-based animation support, as well as compression and progressive delivery. Due to the high precision requirement from the CAD area, U3D file format is designed as a binary format with continuous level of detail geometry.

The primitives of a U3D file are blocks. A block is binary-encoded data, which contains information about a node, a geometry, a modifier, or a resource. The node block creates the scene graph of the model. The modifier block defines animation of the geometries. The resource block provides information of the visual properties of the geometries.

The advantage of U3D file format is its compactness. U3D natively supports a binary compression scheme. In a U3D file, any block can be reused by its reference. However, as a recently proposed international standard, there are only a few tools which can create and modify U3D files. The only available way to visualise U3D files in a web browser is embedding them into PDF files via Adobe Acrobat. Furthermore, U3D file format only provides support for polygon-based, point-based or line-based 3D models. Though the specification itself allows for extensions, as there is no programming ability inside U3D file format building a customised U3D file viewer would be the only way to add support for new models. It is impractical for each user to install a customised browser in order to support different extensions. As the main purpose of U3D is to exchange CAD models, there is no interactivity

Chapter 2. Web visualisation and function-based modelling

or advanced rendering techniques (e.g. vertex shaders and fragment shaders) defined in the U3D file format.

2.2.2 Collaborative Design Activity

The purpose of proposing COLLADA [27], which is the acronym of the Collaborative Design Activity, is to enable 3D authoring applications to exchange created objects and scenes. Such requirements are raised by the game and movie industry. In that area, the companies usually use many different 3D authoring applications in order to build the final product. For example, the software developers might use a totally different application compared to the model designers. Since there is no common standard designed for such usage, sometimes the developers have to write conversion tools. Such conversion tools might be invalid if any of the software is upgraded. Moreover, the model designers often want to re-use the previously created models in order to quicken the designing phase. Therefore, managing the created models, which is actually the digital assets of the company, would also be desired.

COLLADA is designed to fulfil the requirements by the game and movie industry. Since the main concern for COLLADA would be the ease of management and the exchangeability, it uses the XML standard, which is essentially a human-readable format. It does not contain any compression or binary data in the file format. The model representation support by COLLADA would be de facto industry standard polygon-based models and spline models. However, since the game and movie industry heavily use new rendering technologies, COLLADA features the support of advanced appearance (i.e. programmable shaders) and the physics of the objects. Besides storing static objects, it can contain animations as well. However, COLLADA was not designed to be used by the final users or viewers, but rather intended for internal use by the companies. Before the models become a part of a product, they have to be converted to a company-specific binary format in order to protect the intellectual property of the company.

Chapter 2. Web visualisation and function-based modelling

Based on the above analysis, COLLADA is not very suitable for 3D web visualisation. First, it is naturally large in size, since there is no compression or binary-encoding method available. Second, it cannot define interactivity between the users and the models. Third, since there is no programming ability in the COLLADA format, it is not possible to easily extend the COLLADA file format. Finally, currently there is no plugin for the web browsers to visualise COLLADA files.

2.2.3 Moving Picture Experts Group

Moving Picture Experts Group, which is commonly referred as MPEG, is a working group affiliated to ISO/IEC in charging of video and audio encoding standards. The standards proposed by this working group include MPEG-1, MPEG-2, MPEG-3, MPEG-4 and MPEG-7. In these standards, MPEG-1, MPEG-2, MPEG-3 and MPEG-4 are audio and video encoding standards. These standards can be used for multimedia streaming over the network. As defined in MPEG-1, MPEG-2 and MPEG-3, only two dimensional video can be streamed over the network, however, it is possible to stream 3D content with MPEG-4 standard. MPEG-7 standard provides the feature of describing the multimedia content. The description is organised by binary encoded XML, so it can be processed and indexed by computers automatically.

The problem of MPEG standards is the difficulty to extend them, since they do not provide any means of embedded programming. Any modifications to these standards require the approval of the working group. Therefore, it is practically impossible to create any plugins to these standards.

2.2.4 Java 3D

Java 3D is a set of API which can be used to build, control and render 3D virtual objects and worlds. Since Java 3D is based on Java programming language, it is possible to create an

Chapter 2. Web visualisation and function-based modelling

applet using Java 3D, which can be delivered over the network within a browser. There are two ways to create a 3D virtual scene based on Java 3D, either by embedding the scene and objects in the software, or by using a standard file format to store the scene and objects.

When embedding the scene and objects in the software, it is very difficult for common users who know little about programming to build the 3D environment. Everything must be built from scratch using Java language. When the scene needs to be changed, the source code has to be changed and re-compiled accordingly. Viewers have to download the new applet before seeing the changes. This is improper for web visualisation.

When using a standard file format to store the virtual environment, the Java applet can be re-used even if the scene is changed. However, in this case, the Java applet is merely a renderer of the standard file format (e.g. a U3D viewer based on Java 3D). In this case, it cannot be called a Java 3D based model transmission scheme, since the actual model transmitted over the network is a model in the standard file format (e.g. U3D files).

2.2.5 Virtual Reality Modelling Language

Virtual Reality Modelling Language was proposed and adopted as International Standard in 1997. Generally, a VRML file is a collection of objects, which are polygon-based shapes, sounds, lights and viewpoints. Fundamental elements of VRML are called nodes. Nodes can contain other nodes as their children, and fields as the data for the nodes. VRML is capable of representing static and animated dynamic 3D and multimedia objects with hyperlinks to other media such as text, sounds, movies, and images. It employs a hierarchical model structure, which includes such nodes as transforms, geometries, textures, materials, behaviours, sensors and even scripts. It also includes event processing mechanism, which makes interactivity possible.

Though VRML is capable of representing static, animated, and interactive 3D virtual en-

Chapter 2. Web visualisation and function-based modelling

vironments by scripts and events, there are several deficiencies, which may limit application of VRML for wider and further usage. First, if complex shapes are used, the VRML file may become large, and it will take a long time to download it before the scene can be viewed. Even with the gzip compression, a VRML scene can be tens of megabytes including image textures. Second, the standard VRML does not support such advanced visualisation techniques as *Bump Mapping*, *Environmental Mapping*, etc. Third, there is no native constructive solid geometry support in VRML, and it only can be done by using commercial software through import and export functions at the level of polygon meshes. Finally, the animation support existing in VRML is still limited. Many solutions were proposed to overcome these deficiencies.

To provide VRML the ability of automatic generation of smooth animation between several key objects, Alexa et al. proposed morph extension [28]. The key node in this extension is *VectorInterpolator*, which linearly interpolates any two vectors with the same number of elements. Based on this, vertex pairs of two key objects are pre-mapped manually while actual coordinates are linearly interpolated during the morphing. Colours and normals assigned to each vertex are also linearly interpolated using the same mechanism. An alternative approach of smooth animation, which also reduces the file size of time-dependent objects, is to replace a set of complex models with several simple parameters and coordinates. In Humanoid Animation extension [29], human avatars are simplified into segments and joints. With compression and decompression method, humanoid animations can be even transmitted through telephone lines [30]. In addition to this, common human motions can be predefined with several parameters. For example, in [31], avatars in virtual worlds can have several JavaScript controlled predefined motions (e.g. walking, running, gesticulating). However, this kind of approaches requires pre-installation of all models before visualization, which is not practical if frequent changes of the models are expected. Part of the MPEG-4 standard can also be employed to create smooth human and head animation. In [32], a lightweight

Chapter 2. Web visualisation and function-based modelling

talking head visualisation framework is created based on the MPEG-4 Facial and Body Animation. In MPEG-4 FBA, the facial and body animation are encoded as the movements of the key points. Such information can be seamlessly transferred to the client together with other audio-visual content.

GeoVRML [33] is presented as an extension for representing and visualising geographic data. In this extension, instead of defining a coordinate using Cartesian coordinate system, a latitude/longitude/height coordinate system is employed, which is commonly used by geoscientists. The corresponding nodes to *ElevationGrid*, *Coordinate*, *CoordinateInterpolator* are constructed. In addition to standard coordinate nodes, level of detail node is also provided. The ability of locating arbitrary VRML model on a sphere is also provided by *GeoLocation* and *GeoOrigin* nodes.

The ability of visualising volume data is provided by *Volume* node developed by Behr and Alexa [34]. The volume data is transmitted as several pictures or a movie. If 3D texture is available on the client side, the whole volume is sliced into parallel planes perpendicular to the viewers. Otherwise the whole volume is sliced into parallel planes in orthogonal direction. By this method, the volume data is visualised.

Progressive delivery and mesh compression are also provided as extensions of VRML to shorten the waiting time. Fogel et al. [35] proposed *ProdIndexedTriSet* and *ProdLOD* nodes for progressive delivery, in which *ProdIndexedTriSet* extends *IndexedFaceSet* by introducing an array of *ProdLOD* nodes. The *ProdIndexedTriSet* contains the initial model while *ProdLOD* nodes contain the information to refine the model. A mesh compression approach suitable for VRML was proposed by Isenburg and Snoeyink [36]. In their approach, polygonal meshes are traversed in a deterministic way according to connectivity graph. The traversal operation of each step is stored as alphabets in VRML source code. The decompression is done by performing the reverse operation of each step.

Chapter 2. Web visualisation and function-based modelling

Besides the node extensions to the VRML, progressive delivery effects can also be achieved by using MPEG-4 standard. In MPEG-4 standard, 2D/3D objects and the scene structure can be defined in a binary form known as Binary for Scene (BIFS). The MPEG-4 BIFS specification is built based on the VRML standard. Since MPEG-4 natively supports network streaming, the MPEG-4 BIFS information can also be progressively streamed to the client. Such work was done by Hosseini and Georganas [37].

There are several VRML browsers, which also provide enhanced features for VRML. *Cortona* developed by ParallelGraphics [38] and Blaxxun's *blaxxun Contact* [39] provide many extensions including advanced rendering techniques such as *Bump Mapping* and *Environmental Mapping*. These extensions partially solve the limitation mentioned above. Besides the visual improvement, multi-user communication platforms for VRML communities are provided by *blaxxun Contact*, Geometrek's *DeepMatrix* [40] and VSPLUS developed by Araki [41].

The most recent Bitmanagement's *BS Contact VRML/X3D* viewer [42] provides many advanced visualisation features. Besides NURBS, *Bump Mapping* and *Environmental Mapping*, other new techniques including red-blue stereo viewing, particle systems, multitexturing, movie textures, and texture composition are also implemented natively as extensions. The most important improvement starting from BS Contact VRML/X3D 6.2 is that programmable shaders can be implemented for 3D web visualisation. This extension complies with the X3D Programmable Shaders specification proposal. The programmable shaders include vertex shaders and fragment shaders, and can be run faster by the graphics hardware acceleration. With the programmable shaders, procedurally generated textures, volume textures, real-time shadows, displacement mappings, and other special visual effects, which previously were only available in commercial games, can be delivered over the Internet. By introducing the programmable shaders and other extensions, some limitations of VRML are removed. Though shader programs can be run at an interactive rate, there are still many defi-

Chapter 2. Web visualisation and function-based modelling

ciencies. First, most shader programs can only be run with DirectX9 and high-end graphics hardware. Otherwise, the shader program will be either replaced by some simple effects, or run by a software emulator, which is slow. Second, designing a shader program requires either learning a new language (i.e. High Level Shader Language), or buying commercial software.

2.2.6 Extensible 3D

The Extensible 3D is a new open standard for real-time communication across various applications and networks. It includes features used in engineering and scientific visualisation, CAD, multimedia and entertainment, etc. X3D has been already widely accepted as the successor of Virtual Reality Modelling Language. It introduces many new features to enhance the VRML meanwhile still provides backward compatibility with VRML files. It is suitable for various purposes, such as immersive virtual environment using special devices [43, 44] and visualisation on clusters [45]. A free authoring tool X3D-edit [46] is also available to build and visualise the structure of X3D scenes.

For modelling and visualisation purposes, X3D offers many types of geometries, appearances and rigid transformations (i.e. rotation, translation and scaling). The geometries include 2D primitives (such as points, arcs, circles, ellipses, lines and polygons) and 3D primitives (such as boxes, cones, cylinders and spheres). If complex 3D geometries are desired, polygon meshes can be employed. Besides primitives for general modelling purposes, specialised components, such as Humanoid Animation [29] and NURBS for CAD applications, exist in X3D as well. The appearances in X3D include traditional 2D image textures, per-vertex colours, as well as many new techniques, such as multitexturing, programmable shaders [47], and 3D colour textures.

In addition to modelling static shapes, animation is also possible in X3D. There are various types of sensors in X3D which can trigger certain events to start, stop, or maintain the

Chapter 2. Web visualisation and function-based modelling

state of animation. Authors can either choose to use filters, triggers, sequencers and interpolators to achieve simple dynamic effects, or to write a script when complex animation is required. With the power of X3D scripts, authors may even attach additional information to the events in order to provide some extra features in their dynamic worlds.

To make shape modelling in X3D extensible, a unified programming interface is proposed. The programming interface is known as Scene Access Interface (SAI). It eliminates the incompatibility between different programming language bindings in X3D (e.g. ECMAScript and Java), as well as the differences between external applications and internal scripts. Through a stricter definition and more verbose description, the SAI should work exactly the same way with the browsers from different vendors. Authors will no longer suffer from writing numerous programs for different browser using different programming languages in order to achieve the consistent behaviour for the same scene.

Besides programming in X3D, there is another way to make X3D shape modelling extensible. It is called XML transformation. In X3D, there are three different kinds of encodings, namely XML encoding, classic VRML encoding, and binary encoding. Among these, XML encoding is the most important one, and can be utilised to extending modelling methods in X3D. XML encoding exploits the file format known as Extensible Markup Language (XML) [48], which is also an International Standard widely used in industry for exchanging various data. Unlike other X3D encodings, XML files can be generated or converted from other files automatically. There are many tools and standards which can achieve this. Therefore, one possible way of introducing new XML-based modelling methods may be using a transformation from such modelling approaches to X3D XML-encoded files via Extensible Stylesheet Language Transformation (XSLT) [49]. Since XSLT and XML are well-developed standards, developers only need to craft an XSL file for each XML-based modelling approach. The XSL file works on any platform automatically. Many visualisation applications based on X3D are

Chapter 2. Web visualisation and function-based modelling

built in this way, such as X3D-UML [50], visualising historical objects [51], and visualising chemical molecule [52].

In terms of shape modelling features, X3D has many benefits, and has already been widely used in many industrial areas. However, it still has several deficiencies. First, though it provides advanced polygon-based representation methods such as triangle fans and strips, as well as compressed binary encoding, the file sizes would still be large for complex shapes. Second, without commercial modelling software, constructive solid geometry (CSG) is not supported by X3D. Even with commercial software, CSG can only be done at the level of polygon meshes. Third, it is not easy for authors to apply geometric textures (i.e. the texture which modifies the geometry) to a 3D geometry. Finally, the animation support in X3D is still limited, despite some improvements which have been done.

2.2.7 Applications

Active Worlds [53] and Second Life [54] are two of the most popular virtual community applications. Both applications allow users to create their own models in the virtual environment. Since they use 3D model transmission scheme, such user-defined models can be visualised on other clients simultaneously without downloading and installing additional software patches. Both of these software use polygon-based models as the main representation method, while still providing the possibility to use Non-Uniform Rational B-Spline.

In Second Life software, there is an embedded modelling tool, which can be used to create new models while the user is in the virtual community. The network protocol and file format used to transfer the user-defined models is specific to the Second Life application. Open standards such as U3D, COLLADA, VRML or X3D are not used in this application. In order to create animation and interactivity for the user-defined model, a scripting language called Linden Script Language is provided in the software. Recently, the Second Life client application went open source. Anyone who is interested in fixing bugs or implementing new

Chapter 2. Web visualisation and function-based modelling

features can download the source code of the client software. It is possible to introduce other modelling techniques including function-based modelling approaches to the Second Life client software.

On the contrary, there is no embedded modelling tool in Active Worlds. Instead, it uses third party modelling software to create user-defined tools. In order to support third party software, Active Worlds employs the RenderWare 2.0 file format and Truespace file format. Instead of providing an internal programming language, Active Worlds provides a set of APIs to allow plugins to the software.

The common problems of these two virtual community applications would be the following. First, they require additional servers in addition to a conventional web server. Second, though the clients can be freely downloaded, the servers are either available commercially or not available to public at all. Finally, these applications are stand-alone programmes, which cannot be integrated into the web browsers. In conclusion, the protocols used by these applications are not suitable for general 3D web visualisation purposes.

2.2.8 Problems of 3D model transmission

When 3D model transmission is used, the whole model including geometries, appearances, textures and scene structures is transmitted over the Internet. In the current open model transmission standards, polygon-based models and point-based models are usually used. In medical visualisation field, sometimes voxel-based models are used as well.

Since the processing power of the clients have increased dramatically, the demands for highly detailed models have been raised. However, when the highly detailed models are used in the model transmission scheme, the file size becomes too large to be managed and transferred within a reasonable amount of time due to the limits of the peer-to-peer Internet bandwidth. This common problem exists in all the three common modelling methods used

Chapter 2. Web visualisation and function-based modelling

in model transmission. Moreover, when the geometry become more and more complex, the appearance, which essentially are image textures, becomes larger and larger. It is common nowadays that a complex model can reach several megabytes or even tens of megabytes. This amount of data is improper to be distributed over the Internet in a short period of time. Therefore, a new modelling approach must be employed to solve this problem.

2.3 Function-defined shape modelling approaches

Function-based shape representations, in which mathematical functions are used as representation primitives, have the ability to model most objects. Compared to polygonal meshes, function-base shape representations can greatly reduce the model size in many cases, because small formulae can define very complex objects. Furthermore, unlike polygonal meshes, which require several different models to achieve level of detail, one single function-defined shape preserves all the details and can be magnified without losing quality. Despite these advantages, which function-based shape representations provide, visualising function-defined shapes is not straight-forward and requires additional computation. Fortunately, because of the fast development of computer hardware, common personal computers now have a significant computation power. Therefore, function-based shape representations can be very useful in 3D web visualisation.

Generally, there are two aspects that function-based shape modelling focuses on, which are representation methods and rendering methods.

2.3.1 Representation

Function-based shape representation employs functions to define objects. Not only geometric shapes can be defined by functions. Other properties such as colours [55], geometric textures [56] can also be represented by functions. Perlin noise function [57] is another example,

Chapter 2. Web visualisation and function-based modelling

which is widely used to define both colours and geometric textures. It is known that mathematically there are three types of functions – implicit functions, parametric functions and explicit functions. Besides these representation methods, there are also mixed representation methods, which employ more than one representation method concurrently for defining objects. Based on this analysis, function-based shape representation can be classified into three categories, parametric approaches, implicit and explicit approaches, and mixed approaches.

2.3.1.1 Parametric approaches

Parametric approaches employ parametric functions as primitives. The basic idea of parametric approaches is that the parametric function can enumerate each point of the defined shapes, either surfaces (if two parameters are used) or solids (if three parameters are used). Therefore, shapes are actually defined by ranges of parametric functions. Based on this definition, parametric approaches do not necessarily represent 2-manifolds or 3-manifolds, so the existence of interior and exterior is not guaranteed, and point classification (i.e. whether the point is inside/outside the object or on the surface of the object) becomes difficult and even impossible for parametric approaches.

B-Splines, which are able to define parametric surfaces with arbitrary topology [58], are the most popular parametric representation methods, and they are exploited widely in the CAD area. As an extension to B-Splines, Non-Uniform Rational B-Splines have become de facto modelling standard in CAD. NURBS employs control points to construct parametric functions. Each control point is defined by a vertex of a non-uniform grid in one or two dimensional parametric space, and associated with a weight and a coordinate in two or three dimensional Cartesian space, which contains the generated curve or surface. Each control point has only a limited affect area, and the generated parametric curve or surface is guaranteed to be inside the convex hull constituted by several consecutive control points. Because B-Spline and NURBS have such good properties, the parametric objects can be modified by

Chapter 2. Web visualisation and function-based modelling

moving the control points intuitively, predictably and precisely, which is very important for industrial design. In spite of the CAD applications, NURBS have also been introduced in 3D web visualisation. Grahn et al. [59] proposed an extension allowing authors to employ NURBS in VRML.

One of the major problems in B-Splines and NURBS is that the parametric space should be completely tessellated. This problem becomes serious when only a very small part of the parametric object is needed to be modelled in great detail, as not only this small part is subdivided into tiny grids, but many other redundant parts as well. Research work has been done in this direction. Forsey and Bartels proposed a local refinement method employing hierarchical structure [60]. Because the control points of B-Splines and NURBS only influence locally, by carefully choosing additional control points, a small part of the original control grids in the parametric space can be refined by a subdivided control grids. Recently, a more robust and general solution, T-Spline (non-uniform B-Spline surface with T-junctions), is proposed by Sederberg et al. [61]. In their work, T-junctions are allowed in the control grids, which make the parametric space not necessarily entirely tessellated. By allowing T-junctions, for the same surface at the same precision, the number of control points is greatly reduced. Furthermore, cracking between different B-Spline surfaces is completely solved by T-Spline intuitively. A work of reducing control points of B-Spline surface and producing T-Spline surface can be found in [62].

2.3.1.2 Implicit and explicit approaches

Implicit and explicit approaches employ analytically or procedurally defined implicit and explicit functions as primitives. The basic idea of implicit approaches defined on $\mathbb{R}^n$ space can be represented by an equation:

$$f(x_1, x_2, \dots, x_n) = 0$$

Chapter 2. Web visualisation and function-based modelling

where all points $(x_1, x_2, \dots, x_n)$ satisfying this equation (i.e. the solution of this equation) compose the surface of the object. For explicit functions defined on $\mathbb{R}^n$ space, the general form is:

$$x_n = f(x_1, x_2, \dots, x_{n-1})$$

Two different ways can be found to define objects employing explicit functions. One is to use $(x_1, x_2, \dots, x_n)$ as coordinate defining an object. For example, 3D water waves can be defined by $z = \sin(\sqrt{x^2 + y^2})$. However, this method is rarely used in model representation because it is axis-dependent and can only define surfaces. The other one is to artificially increase the dimension of the space by one to define an object in the original space. In this case, for defining an object in $\mathbb{R}^n$ space, the explicit function becomes $x_{n+1} = f(x_1, x_2, \dots, x_n)$. If the value of the extra dimension is fixed, the function defines a surface called an isosurface. If an inequality is used, the original space is divided into two half spaces, in which one of the half spaces is a solid object defined by functions. For explicit approaches, algorithms and ideas can be borrowed from implicit functions since they can be easily converted.

Point classification can be easily done if shapes are represented by these approaches. If point P with coordinate $(p_1, p_2, \dots, p_n)$ is on the surface defined by function f , then $f(p_1, p_2, \dots, p_n) = 0$. If $f(p_1, p_2, \dots, p_n) < 0$, point P is on one side of the surface (e.g. outside the shape). If $f(p_1, p_2, \dots, p_n) > 0$, point P is on the other side of the surface (e.g. inside the shape). Many operations can be easily defined with using implicit and explicit approaches [63], such as set-theoretic operations (union, intersection and difference) with C^n continuity, blending operation, warping operation, twisting operation, etc. FRep, which was introduced in [64], and Boolean Compound Soft Object (BCSO) [65] are examples of such approaches. FRep employs a specific form $f(p_1, p_2, \dots, p_n) \geq 0$ to define the points inside and on the shape. For example, a 3D cylinder along Z-axis with the radius of one can be defined by

$$f = 1 - \sqrt{x^2 + y^2} \geq 0$$

Chapter 2. Web visualisation and function-based modelling

However, this cylinder has an infinite height and the two end caps are missing. If the height of the cylinder is 2, the two end caps can be defined by intersecting with two half-spaces bounded by planes $z = -1$ and $z = 1$. By using the intersection operation (represented by ‘&’), a finite cylinder can be defined by

$$cylinder = infinite\ cylinder \& half\text{-}space_1 \& half\text{-}space_2$$

Considering the continuity of the intersection operation, if C^1 discontinuity is allowed, simple min function can be employed to implement the intersection operation. In this case, the cylinder is defined by

$$f = \min(1 - \sqrt{x^2 - y^2}, 1 + z, 1 - z) \geq 0$$

Though this FRep function indeed mathematically defines a solid cylinder with radius of 1 and height of 2, for practical visualisation purpose, it is not sufficient, because sometimes the continuity of derivations are required in the visualisation pipeline, such as the calculation of normals. In order to provide higher C^n continuity, so-called R-functions are employed. For intersection operation, R-function $(f_1 + f_2 + \sqrt{f_1^2 + f_2^2})(f_1^2 + f_2^2)^{\frac{n}{2}}$ guarantees C^n continuity. Therefore, FRep function with C^n continuity can be used to define such a cylinder easily. Other operations can also be defined by FRep functions. For example, the operation of uniting two objects with blending can be defined by

$$(f_1 + f_2 + \sqrt{f_1^2 + f_2^2})(f_1^2 + f_2^2)^{\frac{n}{2}} + \frac{a_0}{1 + (\frac{f_1}{a_1})^2 + (\frac{f_2}{a_2})^2}$$

Compared to parametric approaches, there is no easy way for implicit and explicit approaches to enumerate every point on the surface, which causes some difficulties to the visualisation. Besides defining the implicit function explicitly by functions, many other convenient and easy-handling representation methods were proposed, and can be classified into several classes.

Chapter 2. Web visualisation and function-based modelling

Skeleton representation is one of the most important classes in implicit and explicit approaches. By employing one or several skeleton primitives, such as points, lines, curves, and even polygons, a filter kernel (usually a radial basis function which defines how the skeleton primitives influence the surrounding space), a blending function (any function that describes how the objects will behave when two or several skeleton primitives come close) and a threshold (a real number which defines where the surfaces are).

The first skeleton representation method was proposed by Blinn [66] in order to visualise molecular structures. In this representation method, points in 3D space are considered as the only skeleton primitive. The filter kernel is defined by a Gaussian function $D(x, y, z) = be^{-ar_i^2}$, in which r_i is the distance between (x, y, z) and the skeleton point P_i while a and b are real constants. The blending function is simply $D = D_i + D_j$, and the threshold is defined as $T = be^{-aR^2}$, in which R is a user-specified radius. The objects defined by this method are called *blobbies*. Similar to *blobbies*, *metaballs* proposed by Nishimura et al. [67] and *soft objects* proposed by Wyvill et al. [68] are other skeleton approaches defined by only points as skeletons.

Skeleton representation is very intuitive in respect of modelling. Shapes can be easily predicted according to the shapes and the locations of skeleton primitives. Moreover, the generated objects are very smooth. However, due to the simple blending function, the blending parts of skeleton representation methods are usually prone to bulge. Even if a skeleton is broken into several consecutive skeletons, the shape would be very different. A mathematical operation convolution can be used to solve this problem. By replacing the simple addition operator by carefully chosen convolutions, as presented by Bloomenthal and Shoemake [69], the bulge disappears. In addition to providing bulge-free blending, convolution surfaces also allow for employing any 2D shapes, including lines, curves, polygons, and even arbitrary surfaces, as the skeleton. Level of detail solutions of convolution surfaces using subdivision of

Chapter 2. Web visualisation and function-based modelling

the skeleton primitives can be seen in [70] and [71]. With different level of detail, interactive modelling objects by skeleton representation becomes possible.

Sometimes, unwanted blending can also be seen in skeleton approaches. For example, a shape defined by five consecutive lines with ‘S’ shape. Apparently, the generated shape should be in an ‘S’ shape as well. However, because the skeleton lines are too close to each other, if blending is not well controlled, the result would be a box-like shape. Research work has been presented in this direction. In [72], Angelidis et al. represented a novel blending control approach by introducing the concept of local convolution. Even for contacted models, the blending result is satisfactory. In this approach, when evaluating the function value for a certain coordinate (called the query point), only the blending function uses a certain portion of the skeleton, which is defined by a neighbourhood of constant size around the corresponding point on the skeleton, (called the reference point). In order to find the reference point, a coarse local mesh is to be built. Referring to the local mesh, finding the reference point becomes finding the planes passing through the query point followed by linear interpolation, which can be done quite easily.

Radial Basis Function (RBF) can also be used to create function-based models. In [73], RBFs are exploited to reconstruct medical volume data. During the reconstruction, the missing part of the real geometry represented by the volume data can be extrapolated by RBFs.

The idea of spline can also be adopted for creating implicit surfaces. Algebraic spline curves (A-spline curves) and algebraic patch surfaces (A-patch surfaces), which are implicitly defined curves and surfaces, were introduced in [74] and [75] respectively. Briefly, A-spline curves and A-patch surfaces are defined by implicit functions generated from triangles and tetrahedra. They guarantee that the produced curves and surfaces are within the triangles and tetrahedra, which is similar to B-spline. Recently, converting A-spline curves and A-patch surfaces to approximate NURBS curves and surfaces were proposed by Bajaj et al. [76].

Chapter 2. Web visualisation and function-based modelling

Recently, discrete implicit representations draw special attention of researchers. A novel discrete implicit representation method called Adaptive Distance Fields (ADFs) was presented by Frisken et al. [77]. In their approach, the implicit function is replaced by trilinear interpolation of the distances of the vertices of each cube. When generating ADFs, the cubes are adaptively subdivided according to the error of trilinear interpolation results and the original distances. For simplification, the error of each cube is estimated by errors at the centre of the cube and the centre of each edge and face.

There are two schemes to generate colours for shapes defined by functions for implicit and explicit approaches. The first scheme is by porting the existing 2D texture mapping scheme of polygonal meshes. The difficulty here is that since the 2D texture must be mapped onto a surface, the coordinate system on the surface must be known. However, the actual size, topology and location of surfaces defined by implicit and explicit approaches are unknown, which is rather different from polygonal meshes and parametric surfaces. Various algorithms were proposed for solving this problem.

In [78], an approach was proposed to build an indirect mapping relationship between the texture and the surface defined by implicit and explicit approaches. First, a parametric supporting surface is constructed, onto which the 2D texture is pre-mapped. After this, particle systems are employed to build a mapping between the supporting surface and the implicit surface. Initially, particles are placed on some critical points of the implicit surface (e.g. vertices of polygonised implicit surface). Particles can be affected by a force field combined by the gradient of the implicit surface, the gradient of the supporting surface, and other point generating attractive/repulsive force. An extension to this work can be seen in [79]. Exploiting parametrically-defined skeleton primitives, Tiggs and Wyvill proposed a solution to the texture mapping problem when several textured objects are blended together.

The second scheme is known as solid texturing. Unlike 2D texture mapping, solid texturing not only generates colours on the surface, but has the ability to assign colours inside

Chapter 2. Web visualisation and function-based modelling

solids as well. Solid texturing is very useful to naturally model objects carved from solid materials such as wood chair, marble sculpture, etc., while 2D texture mapping is prone to give artificial effects. Wyvill et al. [80] first presented a modelling method, which allows function-defined solid textures to be applied to implicitly defined soft objects. Recently, *Constructive Hypervolume Modelling* was proposed by Pasko et al. [81]. This modelling method allows for defining solid texturing for FRep functions [82].

Geometric texture, which is different from traditional 2D or 3D texture, actually modifies the shape in contrast to bump texture where only the surface normal is modified. With geometric textures, not only the surface appears bumpy, but the surface itself is truly displaced as well. Defining geometric textures by functions can be seen in [83], where hairs are modelled by a continuous function $noise(x, y, z)$ producing real values, and [84], where function-defined geometric displacement can be applied to function-based models interactively.

2.3.1.3 Mixed approaches

Mixed approaches employ more than one representation method to model an object. At present, these approaches focus on how to incorporate function-based representation, mainly implicit approaches, with other representation methods, especially with point representations and polygonal meshes.

When points become the representation primitives, the representation method is called a point representation. In general, point representation methods can be used in two ways. One is to control the shape during modelling, and the other one is to reconstruct surfaces from range scanned point clouds.

When shape is controlled by points, usually additional constraints such as the topological relationship between these points or the normals of these points are imposed. In this case,

Chapter 2. Web visualisation and function-based modelling

the represented model is controlled by the points, which are also known as control points, and it is very useful in shape modelling. When the model controlled by control points is finally represented by functions interpolated according to control points, the method can be called a point-function mixed approach. Many systems and representation methods were proposed based on this idea, including *Interpolating Implicits* [85] where surfaces are created by solving a linear system composed of control points and other constraints such as normals, implicit modelling system using particle systems [86] where shapes are controlled very large particles on the surfaces, and *Implicit Partial Differential Equations* (Implicit PDEs) [87] where surfaces are created smoothly by solving partial differential equations built by control points as well as other constraints.

When points are gained from range scanned objects (i.e. point clouds), they become very dense and contain noise. Though point clouds are dense, the geometric information is unlikely to be known for lack of topological knowledge. Furthermore, the scanned data may contain holes due to mechanical problems of the scanning device. By fitting Radial Basis Functions (RBF) to point clouds at selected RBF centres, smooth and hole-free models can be obtained [88]. In [89], normals are considered as additional constraints during the fitting procedure. In [90], the RBF is fit adaptively according to the error between the fit RBF and the original point clouds. Software tools for generating FRep functions from point clouds using compactly support RBFs and octree structure is presented in [91].

Employing traditional polygonal meshes and implicit functions concurrently for modelling objects has also been studied recently. In [92], the HybridTree, which extends the BlobTree to use polygonal meshes as well as implicit functions, was presented by Allègre et al. In their work, polygonal meshes converted into implicit functions as if they are skeletons in a skeleton representation. By choosing a proper filter kernel and threshold, the displacement caused by the conversion becomes less obvious.

Chapter 2. Web visualisation and function-based modelling

2.3.2 Rendering

The final target for visualisation is to generate pictures on the computer screen, so rendering is also very important. Without proper rendering techniques, any perfect representation method will be useless. However, rendering is the weakness of function-defined shapes. Unlike rendering polygonal meshes, there is no straightforward way to render function-defined shapes since current computer hardware only supports fast polygon rendering techniques. Thus, specialised rendering techniques have been carefully studied ever since the proposal of function-based representation.

In general, function-based shape rendering schemes can be classified into three different groups, which are ray tracing, polygonisation followed by fast polygon rendering, and point sampling followed by point rendering.

2.3.2.1 Ray tracing

The rationale of ray tracing [93] is very simple and intuitive algorithm based on physical laws. Rays are cast from the eye location to each pixel in a virtual screen placed in the scene. The virtual screen has the same resolution as the produced image. If rays intersect with objects in the scene, the intersection points are calculated and illuminated, the colours are accumulated. Depending on the physical properties of intersection points (e.g. transparency and refractive index), refraction rays and reflection rays are recursively traced and the illuminated colours are accumulated as well. Finally, the accumulated colour is assigned to the corresponding pixel in the image. Therefore, with ray tracing, photo-realistic images with global reflection illumination can be produced. If a radiosity technique is employed in addition to ray tracing, global diffuse illumination can also be implemented, which further improves the reality of the produced image.

In spite of the quality of rendered image, ray tracing is usually very slow because for

Chapter 2. Web visualisation and function-based modelling

rendering a scene there will be many intersection tests and intersection point calculations. Many methods were proposed in order to accelerate ray tracing by reducing computation caused by intersection tests. Since the number of rays is a constant for any rendered image with certain resolution, the intersection tests can only be reduced by minimising the number of objects tested, which can be done in two ways: spatial partitioning and employing object spatial coherence. The spatial partitioning approach was first proposed by Glassner [94], where Octrees were employed as a spatial partitioning structure for ray tracing. In [95], object spatial coherence is employed by intersecting packets of rays with the same subset of objects selected by Octrees in parallel. By carefully studying and using the modern processor's cache pipeline, even interactive rates can be achieved if the processor's built-in parallel floating point calculation instructions are used. Similar ideas were also been adopted for a ray tracing acceleration hardware [96]. Besides the acceleration for intersection tests, as ray tracing is naturally parallel due to the independent tracing procedure for each cast ray, distributed ray tracing algorithms can also be employed to shorten the rendering time [97]. [98] demonstrates an extension to [95] by distributing the calculation to a cluster of computers. This algorithm has minimal preprocessing and communication overhead, and shows good scalability. As well as ray tracing static scenes, acceleration for ray tracing animated scenes was also studied [99].

Because classic ray tracing only traces the ray cast along a line, aliasing effects usually exist in the ray traced images. Though aliasing can be reduced by introducing supersampling technique, since supersampling requires higher rendering resolution, it greatly increases the rendering time. During the last decade, beam tracing [100] and multiresolution point-sampling ray tracing [101] were proposed to reduce aliasing effects efficiently and precisely.

Chapter 2. Web visualisation and function-based modelling

2.3.2.2 Polygonisation

Polygonal meshes are still de facto standard 3D representation method for both industrial and entertainment applications, and many mature rendering techniques were developed for polygonal meshes. At present, almost every personal computer has acceleration hardware for rendering polygonal meshes. Because of this, rendering polygonal meshes is a relatively easy task without much worrying about the efficiency. Because rendering polygonal mesh is a readily available technique, rendering function-defined shape can be considered as two steps: the function-defined shape is polygonised (i.e. converted into polygonal mesh), and then the polygonal mesh is rendered by existing rendering technique. For different representation categories, different polygonisation approaches are employed.

Parametric shapes are usually easy to be polygonised as the enumeration of the points on the surface is a straightforward process for parametric shapes. The easiest way of polygonising parametric shapes can be done by sampling evenly tessellated grid in parametric domain. However, sometimes, this approach would generate bad polygonal meshes if a higher-order parametric function is processed. Such problem was discussed in [102], where *Normal Umbrella* was employed to generate near-uniform polygonal meshes for any parametric shape.

Polygonisation algorithms for implicit shapes are less straightforward and intuitive, as no surface point enumeration process is provided directly. Briefly, spatial partitioning methods and advancing front methods are the most important types of implicit shape polygonisation approaches, while other miscellaneous interesting approaches, such as employing particle systems to extract surfaces [103], extracting surfaces by connecting slice contours and gradual refinement [104], and Shrinkwrap/Inflate [105, 106], which gradually shrinks/inflates a polygonised sphere surrounding/inside the object homomorphic with a sphere, also exists.

The basic procedure of all spatial partitioning methods is quite the same. At first, the space containing the implicit shape is divided into several semi-adjacent polyhedra (e.g. cubes,

Chapter 2. Web visualisation and function-based modelling

tetrahedra). Then, the classification (i.e. inside or outside the surface) for each vertex of every polyhedron is done. Based on the assumption that each edge of every polyhedron will intersect the implicit surface at most once, triangles approximating the implicit surface can be extracted according to the classification for vertices and a prepared look-up table. The first spatial partitioning polygonisation algorithm, which decomposes the space into cubes, was proposed in [68]. However, the algorithm requires seed points which are located inside the object. This would not be a problem for soft objects, but might be difficult for general function-based models. A general spatial partitioning approach, Marching Cubes, which does not require seed points, was illustrated in [107] and then adopted to implicit surface in [108].

Though Marching Cubes is still very useful in many areas, drawbacks are still significant. First, Marching Cubes contains ambiguities when extracting triangles from implicit surface. By choosing the wrong case, holes will appear in the polygonal mesh. This problem is solved by decomposing cubes into tetrahedrons, which is also known as Marching Tetrahedra as appeared in [109, 110] or tetrahedralisation as appeared in [111]. Second, Marching Cubes is computationally complex, so it is impossible for polygonising shapes on-the-fly. In [112], a novel acceleration technique was proposed, where only visible part of the implicit surface is polygonised. In [113], a fast polygonisation algorithm is proposed which accelerates the function evaluation as well as reduces the function evaluation by applying *Lipschitz constraint* to bounding functions. Finally, Marching Cubes sometimes generates a large number of triangles, in which many are “bad” triangles (i.e. degenerated and thin triangles). Reducing unnecessary triangles can be done either on-the-fly or as a post-process. On-the-fly approaches are implemented either by adopting the idea of level of detail, where triangles located far from the view location can be replaced by coarse mesh [114, 115], or by generating only near-regularised triangles [110, 111]. Post-process approaches can be seen in [116, 117], where not only degenerated and thin triangles are eliminated, but sharp features in the implicit surface are also enhanced.

Chapter 2. Web visualisation and function-based modelling

Obviously, spatial partitioning approaches are easy to implement, however, it is unreasonable and redundant to sample the whole 3D space since only 2D polygonal meshes are extracted. Advancing front methods are designed to only sample points on the surface and directly generate near-optimal polygonal meshes. By maintaining a current mesh contour list initially consisting of only one seed on the surface, advancing front methods add new near-optimal triangles along the mesh contour iteratively until no more new triangles can be added. The first advancing front method, Marching Triangles, was proposed by Hilton et al. in [118], although this method can not be used on closed surfaces since holes will not be filled. Marching Triangles algorithm has been improved in many ways recently. In [119], an adaptive advancing front algorithm is described, including a hole filling process called *sewing*. This algorithm will produce denser polygonal mesh when the local curvature is high and sparse polygonal mesh when the local surface is quite flat. In [120], the algorithm also contains a *growing phase* and a *fill phase*, which is quite similar to [119]. However, by considering the radius of curvature, it can produce finer triangle mesh. In [121], Hartmann presented a modified advancing front method, in which hole filling process is no longer needed by merging close polygons and separating polygons when vertices are close. In order to require only one point on the surface as seed, hexagons are grown instead of triangles.

2.3.2.3 Point sampling and point rendering

As geometry of shapes becomes more and more complex, the common primitive of mesh – triangles are getting smaller and smaller. In many extreme cases, the primitives projected onto the image space are less than a single pixel. So, employing points as rendering primitives are gaining more and more attention. Rendering points is simple since it is unnecessary to maintain the topological relationship between the points, and even unnecessary to do any interpolation. Despite the advantages, a severe drawback narrows the usage of point rendering, since off-the-shelf display adaptors only treat points as vertices of degenerated polygons.

Chapter 2. Web visualisation and function-based modelling

This greatly limits the rendering speed. Thus, many improvements were proposed in recent years to accelerate the rendering speed. Multiresolution technique [122], network streaming technique [13], ray tracing technique [123], and hardware-accelerate rendering technique [124] have been developed for point as the rendering only primitive in the recent years meanwhile mixture rendering technique was also introduced in [125]. Point sampling methods have been developed as well, such as [126].

2.4 Function-based web visualisation

Function-based shape modelling will give a boost to 3D web visualisation. In many cases, a small formula can describe a complex object instead of using thousands of polygons. The size of the models may be thus greatly reduced.

In [127], the BlobTree was proposed to extend VRML with skeleton implicit surfaces and operations like blending, warping and Boolean operations. Thus, polygonal meshes, skeleton implicit surfaces and operations can be used concurrently under the framework proposed. However, the proposed extension was only partially implemented. In [128], Lai and Sourin proposed and implemented FShape node, which allowed for incorporating into VRML scenes shapes developed with implicit functions stored in different proprietary formats. The proposed design proved to be useful.

There were several limitations though. First, the implemented FShape node was based on only one type of the proprietary function-based model, which required an interactive modelling tool. Second, it did not allow for interactive changes of the models, which limited their use in the shared virtual worlds. Third, the shapes could only have an appearance (colour and texture), which VRML currently supports. Fourth, there were no transformation and operation for function-defined models besides those supported by VRML. Finally, the ability of defining time-dependent object in VRML is still very limited and there is a room for further

Chapter 2. Web visualisation and function-based modelling

research on this topic motivated by large sizes of animated VRML objects, limited set of animated transformations, and limited set of time-dependent models.

2.5 Summary

Nowadays, due to the rapid development of computer hardware, clients have got advanced computation ability and can render 3D models at an interactive rate. However, the limitation of the Internet bandwidth still exists. Thus, web visualisation usually employs model transmission schemes because they can achieve interactive manipulation as well as fulfil the requirement of common rapid Internet publishing.

Polygonal meshes are not the only method for defining 3D objects. Alternatives, such as sweeping representations, point representation and fractal representations, can also be employed to define 3D objects. Function-based shape representations, which employ functions as representation primitives, are one type of the general representation methods. Function-based shape representations are more compact than polygonal meshes. For complex objects, they result in small file size. Moreover, one single function-based representation can preserve all details of an object and provide infinite levels of detail. Many operations of shapes can also be defined easily with the help of function-based shape representations. However, visualising function-defined shapes is not as straightforward as visualising polygonal meshes. The visualisation requires lots of computation. Thus, it becomes time consuming. Also, since there are three types of functions, it is difficult to employ them concurrently to model the shape and other properties of one single object.

In conclusion, function-based shape representation methods may give web visualisation a boost since function-defined models are compact, have infinite levels of detail, and easily define operations on shapes. However, many unsolved problems in function-based shape

Chapter 2. Web visualisation and function-based modelling

representation area still exist, and it is an attractive area for a further research work in this direction.

Chapter 3

Hybrid function-based web shape modelling framework

This chapter introduces a hybrid function-based shape modelling framework, which enables users to employ different function-based representations to model the shape concurrently with its geometric texture, colour and other properties. First, the general function-based shape rendering pipeline is discussed in Section 3.1. In Section 3.2, we discuss why parametric and FRep functions are chosen from all the function-based representations to define shapes and other properties. Finally in Section 3.3, the whole diagram of the framework is proposed by considering two visual properties, colour and geometric texture. Finally, Section 3.4 presented a concrete example which helps understanding the concepts presented in this chapter.

3.1 Function-based shape rendering pipeline

Without reference to any particular type of function-based model, let's consider first the function-based shape rendering pipeline. Based on the definition of function as *a mathematical relation such that each element of one set is associated with at least one element of another set*, the rendering pipeline can be defined as it is illustrated in Figure 3.1(a).

The *function provider* defines functions f_i . Depending on the functions used, the set $\mathbb{R}^n$

Chapter 3. Hybrid function-based web shape modelling framework

(a) Function-based shape rendering pipeline

(b) Mapping functions

Figure 3.1: Function-based shape rendering pipeline and mapping functions

Chapter 3. Hybrid function-based web shape modelling framework

can be constituted by Cartesian coordinates of the object to be modelled, by different parametric coordinates, and by higher dimension parameters defining geometry, behaviour and properties of the object. The functions f_i can be defined analytically and procedurally. Each of these functions either represents the shape's geometry or its property. At the end of the rendering pipeline, we define the *renderer*, which eventually converts function values into senses which can be perceived by human, such as images, sounds, smells, forces, etc. Thus, the proposed rendering pipeline can be used in many areas other than visualisation, such as haptic rendering. When the *renderer* is a visual renderer, the function values are converted into either pixel colours constituting the resulting image, or into densities in case of rapid prototyping. For every possible set of values from $\mathbb{R}^n$, and based on certain *rules*, different functions f_i eventually constitute one *object function* f , which maps values from $\mathbb{R}^n$ to some set $\mathbb{R}^m$ that defines geometric and property information of the shape. The geometric information can be provided as an enumeration function (i.e., a function, which can enumerate every point in the solid or on the surface), or a classification function (i.e., a function, which can decide whether the given point is on the surface of the shape, inside it, or outside it). In commonly used web visualisation techniques, only two- or three-dimensional shapes can be rendered. However, higher dimensions can be always introduced if the need arises. Thus, time-dependent (animated) objects involve the time as an additional dimension. Property information may include colours and other parameters associated with the shapes.

Defining the *rules* of using together different types of mathematical functions becomes a key-point of the hybrid function-based shape rendering pipeline.

3.2 Function-defined shapes and properties

Generally, when modelling geometric objects with the proposed hybrid function-based approach, it may be convenient to think of the shape's properties (e.g., colours and geometric

Chapter 3. Hybrid function-based web shape modelling framework

textures) as of other geometric shapes defined functionally in their respective domains. Then, the original geometric shape and the property shapes will merge into the final geometric object defined in the domain of the original geometric shape. The original geometric shape and the property shape functions, as well as all the functions constituting the rules, will eventually create one object function f , as it is illustrated in Figure 3.1(a). This function superposition can be formed differently depending on the function models used for defining geometric and property shapes.

In our further considerations, we will use only two most general forms of function-based representations, which mathematically include all other possible forms. They are common *Parametric* and *FRep* [64] functions.

Parametric functions are defined by functions $f : \mathbb{R}^{n_i} \rightarrow \mathbb{R}^{m_i}$ ($1 \leq i \leq k$):

$$\begin{aligned} v_1 &= f_1(p_1, p_2, \dots, p_{n_i}) \\ v_2 &= f_2(p_1, p_2, \dots, p_{n_i}) \\ &\vdots \\ v_{m_i} &= f_{m_i}(p_1, p_2, \dots, p_{n_i}) \end{aligned}$$

FRep functions are defined by $f : \mathbb{R}^{n_i} \rightarrow \mathbb{R}$ ($1 \leq i \leq k$):

$$v = f_{m_i}(p_1, p_2, \dots, p_{n_i})$$

FRep functions are explicit functions of the shape coordinates, which may include Cartesian coordinates, time, or any other parameters. Depending on the sign of these functions, the points are considered belonging to the respective isosurface, or the solid shape bounded by this surface. As it was introduced in [64], positive values define points inside the solid, negative - outside it, and zero values are for the points on the surface. When only equality operators are considered, we are dealing with implicit functions.

Both the parametric and FRep functions can define the shape's geometry as well as its property. The functions defining the geometry of shapes are called *shape functions* **obj** =

Chapter 3. Hybrid function-based web shape modelling framework

$f(\mathbf{p})$ (Figure 3.1(b)). The arguments of these functions $\mathbf{p}$ are coordinates in the respective domain space. Similarly to the shape functions, properties (e.g., colours and textures) can also be represented either by parametric or by FRep functions. These functions are called *property functions*. Each property associated with the shape can be described as a vector $\mathbf{prop}$. If this vector contains only one real number, the property can be represented by an FRep function. If this vector contains two or more real numbers, the property can be represented by parametric or several FRep functions. The arguments of these functions $\mathbf{q}$ are coordinates in the property function's domain space. For any function-defined models, we consider only one geometric shape function $\mathbf{obj}_1 = f_1(\mathbf{p})$ and several property functions $\mathbf{prop}_i = f_i(\mathbf{q}_i)$ ($i = 2, \dots, k$).

Rules $f = rule(f_1, f_2, \dots, f_k)$, are functions defining how the shape and the properties are merged. The rules consist of two parts. First, the rules must establish a certain relationship between the domain spaces of the shape function and the respective property functions. These relationships are represented by domain *linking functions* $\mathbf{q}_i = link_i(\mathbf{p})$. Second, for any given $\mathbf{p}$ the respective shape and property functions will be evaluated, and their values $\mathbf{prop}_i = f_i(\mathbf{q}_i) = f_i(link_i(\mathbf{p}))$ will be used for creating the resulting object. This process is called *merging*, and the functions defining this process are named *merging functions* $\mathbf{obj}_i = merge_i(\mathbf{obj}_{i-1}, \mathbf{prop}_i)$. For each property, the respective linking and merging functions constitute one *mapping function* $\mathbf{obj}_i = map_i(\mathbf{p})$. Therefore, the number of mapping functions is equal to the number of properties defined in their respective domains. The mapping functions sequentially append the resulting object by each of the properties, thus superposing the final object function $\mathbf{obj} = rule(f_1, f_2, \dots, f_k)$.

Four classes of possible relationships between the domain space of the shape function and the domain spaces of the property functions can be identified. The first class is for constant property functions, for example, a uniform colour. In this case, no linking function is needed. The second class includes the property functions sharing the same domain with the shape function. In this case, the relationship between $\mathbf{q}$ and $\mathbf{p}$ is just $\mathbf{q} = \mathbf{p}$. The third class is

Chapter 3. Hybrid function-based web shape modelling framework

defined when the property function has the same type of domain space as the shape function, but some *coordinate transformations* (e.g., translation, rotation, scaling, etc.) are involved. In this case, $\mathbb{R}^{n_i} = \mathbb{R}^{n_1}$, so that the linking function becomes $\mathbf{q} = \text{transform}(\mathbf{p})$, where $\text{transform} : \mathbb{R}^{n_1} \rightarrow \mathbb{R}^{n_1}$. The last class is for the shape and property functions having different types of the domain spaces (i.e., the property function is a parametric function, while the shape function is an FRep function). In this case, *coordinate conversion*, which converts one type of domain space into another type, as well as coordinate transformations are involved. Thus, the linking function between $\mathbf{q}$ and $\mathbf{p}$ is $\mathbf{q} = \text{conversion}(\text{transform}(\mathbf{p}))$, where $\text{conversion} : \mathbb{R}^{n_1} \rightarrow \mathbb{R}^{n_i}$.

3.3 Modelling framework

Let's consider only two visual properties of shapes, which are *textures* and *colours*, and define a modelling framework for this case. Here, by textures we understand some fine geometric distortions and additions applied to the original geometric shapes (e.g. roughness, bristle, fur, etc.). By colours we understand parameters in the illumination model on the surface as well as inside the objects, including shininess, opacity, ambient reflectance, diffuse reflectance and specular reflectance. We consider the traditional image and solid textures as function-defined colours. Based on this classification, the function provider should somehow define three kinds of functions, which represent the *geometric shapes*, *textures* and *colours*. The final geometric objects are formed by applying textures and colours to the original geometric shapes. Thus, two separate steps form the rules in the rendering pipeline. First, all the textures are applied to the geometric shape. Then, colours are mapped to the textured object, so that the final object with colour contains all the necessary information for the renderer. These two steps are repeated if needed. In these steps, mapping functions are employed to define how textures are applied to the geometric shape, as well as how colours are assigned to the object. It is important to note that after applying textures, the resulting object will belong to the same

Chapter 3. Hybrid function-based web shape modelling framework

Figure 3.2: FRep and parametric objects

function type as the original geometric shape. Thus, a parametrically defined shape produces a parametrically defined object, no matter which type of texture is applied: parametric or FRep. Similarly, FRep shapes create FRep objects, which may however have parametric or FRep textures. This is illustrated in Figure 3.2. In these diagrams, the rectangles represent the function-defined shapes including surfaces and solids. The rounded rectangles represent the function-defined geometric textures. The lines linking the rounded rectangles and the rectangles represent the mapping functions, which define how the textures are applied to the shapes.

In the second step, a function-defined colour is to be mapped to the geometric object being modelled. Since both the colour and the object can be either parametric functions or FRep functions, four other mapping functions have to be used. Figure 3.3 illustrates the framework for the second step. In this diagram, the rectangles represent the objects constructed in the first step. The rounded rectangles represent the colours. The lines represent the mapping functions for colours and objects.

Figure 3.3: Mapping colours and objects

Chapter 3. Hybrid function-based web shape modelling framework

Figure 3.4: Framework of hybrid function-based shape modelling

By combining Figure 3.2 and Figure 3.3, the whole framework is constructed as it is illustrated in Figure 3.4. From this diagram, we can conclude that only eight mapping functions (represented by rectangles, M1 to M8) should be proposed in order to make this framework complete and feasible. According to the classification presented in Section 3.2, each linking function can be one of four types.

For the first case, no linking function is needed. It is the simplest case since it must follow the criteria that the property is a constant. In our framework, every mapping function can contain this kind of linking functions if the property is constant. For example, if the shape is defined by $1 - \frac{x^2}{4} - \frac{y^2}{9} - \frac{z^2}{16} \geq 0$ and the colour is red, the result will be a uniform red ellipsoid.

For the second case, a simple linking function $q = p$ is employed. Since q and p share the same domain space, only mapping functions with the same type of domain space (i.e. M1, M4, M5 and M8) can contain this kind of linking functions. For example, if the shape is defined by $4 - x^2 - y^2 - z^2 \geq 0$, the colour is defined by $c = 1 - \frac{x^2}{4} - \frac{y^2}{9} - \frac{z^2}{16}$, and the

Chapter 3. Hybrid function-based web shape modelling framework

merging function interprets $c = 0$ as green, $c = 1$ as blue, and linearly interpolates colours in between, the result will be a sphere with a colour varying from green to blue.

For the third case, coordinate transformations are involved. Because $\mathbf{q}$ and $\mathbf{p}$ belong to the same type of domain space, only mapping functions M1, M4, M5 and M8 can contain this kind of linking functions. For example, the shape is defined by $4 - x^2 - y^2 - z^2 \geq 0$, while the colour is defined by $c = 1 - x^2 - y^2 - z^2$. The merging function interprets “ $c = -3$ ” as red and “ $c = 0$ ” as green. If no transformation is involved, the result will be a red sphere. But if the transformation is a scaling function with the factor of 0.5, the result will be a green sphere.

For the fourth case, coordinate conversions as well as coordinate transformations are involved. As $\mathbf{q}$ and $\mathbf{p}$ have different domain types, only mapping functions M2, M3, M6 and M7 can contain this kind of linking functions. For example, the shape is defined by $x = \cos(u)$, $y = \sin(u) \cos(v)$, $z = \sin(u) \sin(v)$, and the texture is defined by displacement $d = \sin(x) \sin(y) + \sin(x) \sin(z) + \sin(y) \sin(z)$. The coordinate transformation function will normalise u , v and w into the range $[-1, 1]$. The coordinate conversion function is $x = 4 \cos(\pi u)$, $y = 4 \sin(\pi u) \cos(\pi v)$, $z = 4 \sin(\pi u) \sin(\pi v)$. The merging function will use the value of texture function to define the amount of displacement along the surface normal. The result of these functions will be a bumpy unit sphere.

Merging functions for textures are functions, which accept the shape and the texture functions as their arguments and produce another shape function as their result. The resulting shape is the original shape distorted by both the texture function and the merging algorithm.

Merging functions for colours are functions, which accept the shape and the colour functions as their arguments, and produce exactly the same shape with colours on its surface and/or inside the shape. The colours assigned to the shape are calculated by both the colour function and the merging algorithm.

Chapter 3. Hybrid function-based web shape modelling framework

Operations on functions may also be considered as a part of the input functions in the rendering pipeline. In this case, there are multiple shapes defined by the original functions. At present, the proposed modelling framework only allows operations to be applied to the same kind of geometry functions due to the inefficiency converting from one kind of geometry functions to another. When applying the operations on the textured and coloured shapes, usually the properties and the geometries need to be treated separately. Otherwise, an unwanted result may appear. For example, suppose set-theoretic operation union is to be applied to one red sphere and one green box. The expected geometry is the union of the sphere and the box, while the expected colour is that the sphere part of the resulting shape is red, while the box part of it remains green. If the colour functions are treated the same way as geometry functions, the colour of the resulting geometry becomes a yellowish uniform colour. One of the ways to achieve the expected result would be by a procedural function, which evaluates the original geometry function, and decides which original colour function should be used.

3.4 Example

To gain a better understanding of the concepts presented in this chapter, a concrete example is analysed in this section. Suppose that there is a function-based rendering pipeline, which is able to visualise colours and applies geometric textures. If we want to render a skeleton function-based model distorted by a function based on Gardner series, and with colours defined by Perlin noises, three functions can be provided to the pipeline as inputs: the skeleton-based function defined by FRep function in $([-1, 1], [-1, 1], [-1, 1])$, the Gardner series function defined by FRep function in $([-2, 0], [-2, 0], [-2, 0])$, and the Perlin noise function defined by parametric function in $([-1, 1], [-1, 1], [-1, 1])$. In this case, the skeleton-based function is known as the *shape function*, while the Gardner series function and the Perlin noise function are known as the *property functions*. Since the functions may be provided in different representations (e.g. the shape function is represented by skeletons and field func-

Chapter 3. Hybrid function-based web shape modelling framework

tions, and the other functions may be represented by analytical formulae), the procedures, which can evaluate these functions, are known as the *function providers*. Then, the following steps are necessary in order to produce the rendering result, which forms the *rule* in the rendering pipeline.

First, the shape must be distorted by the corresponding geometric textures. In order to achieve this, the domain of the geometric texture function and the domain of the skeleton-based function must be matched, i.e. we must know how much distortion is needed for each point belonging to the shape. Since the domain of the Gardner series function is different from the skeleton-based function, a function can be employed to map $([-2, 0], [-2, 0], [-2, 0])$ to $([-1, 1], [-1, 1], [-1, 1])$. This function is called the *linking function*. As the type of the geometric texture function is the same as the type of the shape function, the only feature of this function is to transform the domain of the geometric texture function into the domain of the shape function. For example, a simple translation is enough to achieve this result. After this, the geometric texture defined by the Gardner series can be applied to the skeleton-based model to produce a new textured model which is defined by a new FRep function defined in $([-1, 1], [-1, 1], [-1, 1])$. This can be achieved by a function which displaces the original inputs to the unit shape function according to the corresponding value of the geometric texture function. This function with displacing feature is called the *merging function*.

Second, the textured skeleton-based function produced by the previous step should be coloured according to the colour function. The domain of these two functions must be matched as well. Though the domain of the Perlin noise function seems to be the same as the textured skeleton-based function, since the types of the functions are different, a domain parameter mapping must be done. In this case, a simple *linking function* $u = x, w = y, v = z$ can be used. The *merging function* is a function which adds three additional dimensions R , G , and B to the original textured shape function value.

Finally, the coloured textured skeleton-based function, which is a function taking three

Chapter 3. Hybrid function-based web shape modelling framework

numeric inputs and producing four numeric outputs, can be rendered by a function-based rendering system. This system should recognise the first numeric output as the FRep function value, while the additional three outputs as colours. For example, we may choose a polygonal rendering system with polygonisation algorithm to render the result, which is known as the *renderer*.

From this example, we can see that any system which can render function-based models should follow the presented function-based rendering pipeline.

3.5 Summary

All the existing function-based shape modelling frameworks have one common problem, that is different types of functions can not be employed concurrently to model both geometries and properties, such as colours and textures. The proposed hybrid function-based shape modelling framework is as an efficient solution to this problem, since it introduces only *mapping functions* in addition to the function-based rendering pipeline. In this modelling framework, the key is the proposed hybrid function-based shape rendering pipeline. Using this modelling framework, any shapes, colours and textures built by other function-based shape modelling approaches can be easily put together to create new objects.

The key parts of the hybrid function-based shape rendering pipeline are rules. They consist of mapping functions, which map different coordinate spaces and create new objects. If three-dimensional objects with colours and geometric textures defined by either parametric or FRep functions are considered, a full implementation should have eight mapping functions. The rules in the rendering pipeline should be formed by first applying the mapping function between geometric textures and the shape, and then by applying the mapping function between colours and the shape. Each mapping function must have one linking function and one merging function. Linking functions for any of the eight mapping functions should always be

Chapter 3. Hybrid function-based web shape modelling framework

defined by $link : \mathbb{R}^3 \rightarrow \mathbb{R}^3$. For merging functions between geometric textures and shapes, the resulting function should have the same form as the shape function, while for merging functions between colours and shapes, the resulting function should have more output values representing colours. In the next chapter, two practical implementations – the extension of Virtual Reality Modelling Language and the extension of Extensible 3D – are presented to illustrate the feasibility of the proposed framework.

Chapter 4

Function-based extension of VRML/X3D

In this chapter, the design of the function-based extension of VRML/X3D is presented. Function-based extension of VRML/X3D is an implementation of the hybrid function-based shape modelling framework described in Chapter 3. In order to illustrate that the proposed modelling framework is general and extensible, the design does not propose any limitations on implementation approaches. The reason to choose VRML/X3D as the design platform is given in Section 4.1. The general scope of the extensions is illustrated in Section 4.2. It includes the goal of the extension as well as the design concept of the prototypes. Section 4.3 presents the methods used to define functions in the extension, including analytical definitions of functions and a JavaScript-like scripting language. Finally, the extension prototypes for VRML and X3D are proposed and discussed in Section 4.4.

4.1 Selection of the implementation platform

Since model transmission schemes are the most commonly used approaches for 3D web visualisation, the implementation of the hybrid function-based shape modelling framework should also be based on the model transmission schemes. By introducing hybrid function-based models to model transmission schemes, the model size usually can be dramatically reduced compared to traditional modelling methods, such as polygon-based models, point-

Chapter 4. Function-based extension of VRML/X3D

based models or voxel-based models. Moreover, function-based models can achieve unlimited levels of detail. Thus, transferring a complex model defined by functions is comparably faster than transferring a complex model defined by traditional methods.

As mentioned in Section 2.2, there are many existing model transmission standards, such as U3D, COLLADA, RenderWare, MPEG-4, VRML, and X3D, and application-specific protocols, such as Second Life and Active Worlds. All of these standards can be extended to use hybrid function-based models if a new application is developed. For example, the original client of Second Life only supports objects created based on a limited set of primitives (e.g. boxes, cylinders, spheres, cones) using constructive solid geometry (CSG). However, since the client software is now open-source, any developer can modify the software for their own needs while still providing compatibility to the original software. With the source code, the extensions can access the internal models of Second Life at the polygon level. Thus, adding other polygon-based primitives as well as hybrid function-based models is possible.

It is very important to choose the proper platforms to be extended with the hybrid function-based models. The criteria to choose such standard would be the following. First, the standard must be easily extensible. The extension to the standard should be able to co-exist with the original objects. When a browser cannot recognise the extension, it should not crash or generate error messages, but simply ignore the extension. Second, the standard must provide some internal programming facilities. Although it is always possible to create our own stand-alone player or a web browser plugin for the chosen model transmission scheme. This would be very inconvenient to the users since they have to install new software and learn the user manual. Because of this, it is reasonable to try the best to keep the extension running on available software. Finally, since the extension is web-based, it should be possible to render such model in a web browser, such as Microsoft Internet Explorer or Mozilla Firefox. It is very important to be able to use off-the-shelf software to render the chosen model transmission scheme.

Chapter 4. Function-based extension of VRML/X3D

Taking all the criteria into consideration, as analysed in Section 2.2, VRML and X3D fulfil all the requirements. Although other standards and applications may be used to implement the hybrid function-based web shape modelling framework, they have various problems. For example, it is possible to implement a new Second Life client based on the open source official client, but the customised client might not be fully compatible with the official client. The implementation of the hybrid function-based web shape modelling framework is thus built as the Function-based Extension of VRML and the Function-based Extension of X3D.

VRML and X3D are current international standards approved by the International Standard Organization. The interest of VRML and X3D are growing among developers, authors and users. According to statistics, the number of queries for “VRML” and “X3D” continues to climb. The number of queries for “VRML” has increased 8 times in the last 2 years, reaching 10 million, while for “X3D”, the number has been tripled during the same period. The popularity of VRML and X3D standards is way ahead other competing Web3D technologies, such as COLLADA and U3D. In VRML and X3D standards, polygon-based models are the fundamental way to create 3D geometries. They also support conventional ways to define geometry appearances, such as per-vertex diffuse colours and 2D image textures. Transparency, shininess, ambient intensity, specular colour, and emissive colour can also be defined as the material on an object-basis. Affine transformations are also included as part of the standards. The transformations can rotate, translate and scale the defined objects to a proper place in the scene. Advanced visualisation methods, such as programmable shaders, are also supported by X3D and have already been proposed as VRML extensions. However, VRML and X3D are purely text-based file format. Even with the embedded gzip compression support, the file size would typically be as large as several megabytes.

In the VRML and X3D standards, there are also several time-dependent nodes available. These can be used to create animations. Based on the properties of the existing

Chapter 4. Function-based extension of VRML/X3D

time-dependent nodes, they can be classified into two groups, *Self-contained Time-dependent Nodes* and *Non-self-contained Time-dependent Nodes*.

Self-contained Time-dependent Nodes can be used independently of any additional nodes. These nodes always contain some internal time sensing mechanisms which allow them to track changes of time. Usually, the self-contained time-dependent nodes expose only a few parameters and events to other nodes. By changing these parameters, default behavior can be changed accordingly. In VRML, *AudioClip* and *MovieTexture* belong to this group. *AudioClip* is a node which can play an audio file. The cycle interval (i.e. the time required for playing the node) is defined by both the pitch property and the length information contained in the audio file. *MovieTexture* is used for playing a video clip. Its cycle interval is defined by both the speed property and the length information hidden in the video clip. Both these nodes can be used independently of external controls. Though they have exposed fields start time, stop time, and loop, in fact only infinite cycles can be played back since only the absolute time originated from January 1, 1970 can be used in VRML.

Non-self-contained Time-dependent Nodes cannot be used on their own. In order to be used together with other nodes, they have to expose as much properties and events as possible to provide for more flexibility. *TimeSensor* is such a node. For example, by combining the *TimeSensor* and various interpolators, time-dependent objects can be modelled in VRML and X3D.

4.2 Extension scope

The goal of the extension is to propose an extension to VRML and X3D which allows for visualising function-based models within the browsers. The design should fully consider every feature provided by the hybrid function-based web shape modelling framework, while still maintain the compatibility of the VRML and X3D standards. Generally speaking, the

Chapter 4. Function-based extension of VRML/X3D

extension should be a superset of the standard VRML and X3D nodes defining object-related information. All existing features, fields, and usages should be introduced in exactly the same way as they are in VRML and X3D standards.

Referring to the proposed modelling framework in Chapter 3, there are two visual properties defined by functions in addition to the geometry. The two properties are colours and geometric textures. In standard VRML and X3D, colours are provided as either image textures or per-vertex colours. However, the geometric textures are missing. In the design of the extension, the geometric texture should be included as part of the extension. Though in the modelling framework, there is no limitations on the dimension of the object defined by functions, only three dimensional geometries, colours and textures are considered here. This is so, because VRML and X3D standards are proposed to define virtual objects in three dimensional virtual world. All the coordinates defined in VRML and X3D are three dimensional. Therefore, there is no reason to consider lower or higher dimensional objects and properties, since they cannot be represented and visualised in VRML and X3D.

When standard VRML and X3D define static objects in the scene, the usual procedure is to define the geometry of the object in its own coordinate system first, and then to assign diffuse colours either by per-object colouring, image textures, or per-vertex colouring. After this, the object can be transformed via affine transformation in order to be scaled to a desired size and placed in a proper position in the virtual scene. Besides the diffuse colours, other visual properties can also be defined on per-object basis. These properties includes specular colour, emissive colour, ambient intensity, transparency, and shininess. As these fields are crucial to define static objects in the virtual world, the function-based extension should consider defining them by functions. All the fields mentioned above are defined by either a float number or a float vector, therefore, it is possible to define them by FRep functions or parametric functions. These are the only two types of functions which are considered in FVRML

Chapter 4. Function-based extension of VRML/X3D

and FX3D. This is because all other function representations can be converted to either one of these two.

When defining colours or geometric textures by functions, it is usually easy to consider them as a separate geometry in other 3D spaces, either Cartesian spaces or parametric spaces, depending on what kind of function is used. When mapping these spaces, according to the hybrid function-based web shape modelling framework, the mapping functions must be adopted in order to produce the final object. In this case, the merging functions are quite obvious, as it is already mentioned in Section 3.3, while the linking functions are not specified since virtually any functions can be used. In the design of the extension, it is unnecessary to confuse the users in exchange of the freedom of specifying user-defined linking functions. Instead, certain pre-defined linking functions can be proposed. The most natural way to map two bounded spaces would be by employing a 3D linear linking function which linearly maps the coordinates from one space to another. For example, if we have a point (x, y, z) in the range $(x_1, y_1, z_1) - (x_2, y_2, z_2)$, and want to map it to the range $(x'_1, y'_1, z'_1) - (x'_2, y'_2, z'_2)$, the linking function will be the following.

$$(x', y', z') = \frac{((x, y, z) - (x_1, y_1, z_1))((x'_2, y'_2, z'_2) - (x'_1, y'_1, z'_1))}{(x_2, y_2, z_2) - (x_1, y_1, z_1)} + (x'_1, y'_1, z'_1)$$

Besides static geometries, appearances, and transformations, VRML and X3D also allow the definition of animations of objects. It is also very useful to not only define static function-based models, but also time-dependent function-based models. Time-dependent function-based models may define animated geometries, colours, geometric textures, transformations, operations, or arbitrary combination of the listed items. Thus, it is very important to understand the representation of time in function-based models.

From a mathematical point of view, time is another dimension in addition to the existing space. Therefore, it is possible to represent any time-dependent property or geometry by a function defined in a higher dimensional space. However, time is usually considered to be

Chapter 4. Function-based extension of VRML/X3D

different from other dimensions. The most significant difference is that usually no transformations or operations involve time as one of the dimensions. From this point of view, time is rather similar to a parameter of a function. Thus, a time-dependent function can be defined as a function with multiple variables, where one variable represents time meanwhile the other variables represent coordinates in the original space. By this definition, for each specific time, a function defined in the original space can be obtained. This seems to be the most reasonable and straightforward function-based representation method for a time-dependent shape. For example, a 3D time-dependent FRep sphere can be defined by the following formula:

$$f(x, y, z, t) = at^2 - x^2 - y^2 - z^2$$

In this function, (x, y, z) represents coordinates in 3D Cartesian space, while t represents the time parameter. For each specific time T , by substituting t with the constant T , we can get a standard FRep function which defines a static sphere with radius $a|T|$:

$$f(x, y, z) = aT^2 - x^2 - y^2 - z^2$$

Based on the mathematical definition of the functions, two problems must be clarified. One is the range of the time. Usually, time is recognised as a positive real number. However, in order to provide more flexibility, we may want to visualise any range of the time parameter including negative real numbers. The other problem is how the time parameter changes during playback. Sometimes, we may want to control the playback speed of a time-dependent object. For example, we may require the radius of the above time-dependent sphere to change from 0 to 1 in only 1 second, which results in a very fast animation, or in two minutes, which results in a very slow animation.

In order to solve the problems, a linear time mapping function is proposed in our approach. First, we define the domain of the time parameter, which is a part of the whole real domain. For example, we may define the time domain of the time-dependent sphere as $(-1,$

Chapter 4. Function-based extension of VRML/X3D

2). Then, such time domain is linearly mapped to the time in the real world. For example, if the time range $(-1, 2)$ of the mentioned time-dependent object maps to 10 seconds, the whole animation will last exactly 10 seconds.

In addition to define function-based models alone, the extension should also include the ability to exploit function-based geometry, colours, and geometric textures together with polygon-based geometry and traditional image textures. This would be helpful when authors want to use function-based colours and geometric textures with their existing polygon-based models without conversion to the function-based geometries, or apply existing image textures to the function-based models without using function-based appearances. This free mixture of a function-based modelling approach and a traditional polygon-and-image-based modelling approach provides the authors maximum flexibility and convenience.

4.3 Function definition

Because VRML and X3D are text-based file formats, it is important for the extension to use clear texts to define functions in FVRML and FX3D as well. Since it is very inconvenient for authors to rely on external tools to edit the functions, it is very important to make function definitions human-readable and human-editable. Because of this, the function definition should be written in plain text, instead of any other binary encoded format, such as BASE64 encoded texts. Other than this consideration, we also want to provide more features to advance users, meanwhile keeping simplicity for novice users. Based on all these requirements, two different function definition methods are introduced in the extension design, namely analytical formulae for novice users, and function scripts for advance users.

In general, parametric functions and FRep functions can have an arbitrary number of variables, but for static and time-dependent web visualisation, usually three or four variables are enough. Therefore, we consider only four variables. As such, for defining geometry of

Chapter 4. Function-based extension of VRML/X3D

shapes with parametric functions, we use the following form:

$$x = \phi_1(u, v, w, t); y = \phi_2(u, v, w, t); z = \phi_3(u, v, w, t);$$

For FRep functions, we use the following form:

$$g = f(x, y, z, t) \geq 0$$

Both representations allow for defining surfaces and solid objects, either static objects or time-dependent objects. For modelling surfaces, either parametric functions with two parameters (u and v), or implicit functions, which are in fact a subset of FRep, are to be used. Solid objects can be defined either with parametric functions with three parameters (u , v , and w), or with FRep functions of three Cartesian coordinates. If animation is required, in addition to the three parameters, a time parameter t is also introduced in the function definitions. In the function formulae, in addition to the commonly used operators, two pre-defined useful operators are also introduced. These operators are intersection and union, with any desired continuity, as defined in [64].

The same classes of functions can also be employed to define colours and textures. Colours include uniform colours, as well as conventional and solid texturing. By textures, we understand distortions of the original geometry rather than colouring the shape. Conventional textures, including image textures, movie textures, environmental textures and bump textures, can be considered as colours generated by look-up tables. In our scope, these textures are classified as procedurally defined colours.

4.3.1 Analytical formulae

When talking about the representation of a function, most people with mathematical knowledge would choose an analytical formula, since it is simple and sufficient for definitions of most mathematical functions. Therefore, the most straightforward and natural way for the

Chapter 4. Function-based extension of VRML/X3D

users to enjoy the advantages of the function-based approach in web visualisation is to allow shape definition, as well as their appearances, transformations and operations, with analytical functions in the code.

In our design, a C-like syntax is used for defining analytical formulae. Arithmetic operators '+', '-', '*', '/', round brackets, and an additional operator '^', which denotes exponentiation, can be used. The following mathematical functions are pre-implemented as a library: *abs(x)*, *sqrt(x)*, *exp(x)*, *log(x)*, *sin(x)*, *cos(x)*, *tan(x)*, *acos(x)*, *asin(x)*, *atan(x)*, *ceil(x)*, *floor(x)*, *round(x)*, *max(x,y)*, *min(x,y)*, *atan2(y,x)*, *mod(x,y)*, *cosh(x)*, *sinh(x)*, *tanh(x)*, *log10(x)*. In addition, for FRep functions, set-theoretic intersection '&' and union '|' are supported with any desired continuity, as it is defined in [64].

The variable names for FRep formulae defining geometry of shapes are x , y , z and t , in which (x, y, z) defines the Cartesian coordinates in 3D space, and t defines the time parameter. The formulae are to look like $f(x, y, z, t)$. For example, for the origin centred half-sphere with a time-dependent radius, it can be

$$(t * t - x * x - y * y - z * z) \& y$$

For parametric functions defining geometry of shapes, u , v , and w represents the coordinates in the parametric space, while t defines the time parameter. For example, for the origin centred sphere with a time-dependent radius, it can be

$$\begin{aligned} x &= fabs(t) * \cos(v * \pi); \\ y &= fabs(t) * \cos(u * \pi) * \sin(v * \pi); \\ z &= fabs(t) * \sin(u * \pi) * \sin(v * \pi); \end{aligned}$$

For functions defining colour, the analytical definition use the same syntax as above, except that the variable names for parametric functions are different. Instead of using x , y and z , which define a point in the Cartesian coordinate space, r , g and b are used for defining a colour in the RGB colour space. If FRep functions are employed to define colours, the function values are mapped to an RGB colour linearly using a user-defined colour interpolator.

Chapter 4. Function-based extension of VRML/X3D

When geometric textures are defined by analytical functions, the syntax is exactly the same as for defining geometry. In FRep functions, the function value defines the distance of a displacement along the normal. In parametric functions, variables x , y and z define the absolute displacement of a given point.

Other visual properties, such as specular colour, emissive colour, ambient intensity, transparency and shininess can also be defined by analytical formulae. However, due to the restrictions of VRML and X3D (i.e. these properties are object-based), it is not possible to specify the properties for each coordinate in the shape. Therefore, they can only be defined as a time-dependent functions, and the value of coordinates will be fixed as 0. Furthermore, since transparency and shininess are defined as a single value in the VRML and X3D, only FRep functions are employed.

When defining operations, FRep functions are employed. Three predefined set-theoretic operations, as well as any analytically defined operations are supported. The FRep functions used to define operations have the syntax $g = h(f_1, f_2, t)$. In this formulae, f_1 and f_2 are functions of models, which can be either FRep or parametric functions. Parameter t defines the time when the operation is time-dependent, such as a linear morphing between two geometry. When the operation is static, this parameter should be ignored.

In the design of FVRML and FX3D, it is also possible to define function-based time-dependent affine transformations, as VRML and X3D provide the ability to define static affine transformations if no external nodes are used. When doing affine transformations, two different types of vectors are used, which are 4D vectors and 3D vectors. 4D vectors are used to define the rotation as well as the scale orientation, while 3D vectors are used to define translation and scaling. When defining these vectors, only parametric functions are used. The only parameter of the transformation functions would be t , which represents the time parameter. Therefore, for 4D vector used to define time-dependent rotation and scale

Chapter 4. Function-based extension of VRML/X3D

orientation, the following form is used

$$x = \phi_1(t); y = \phi_2(t); z = \phi_3(t); a = \phi_4(t)$$

Here, the value x , y , z , and a have exactly the same meaning as defined in standard VRML and X3D. For 3D vectors, only a value is missing while all other three values remain the same. The meaning of x , y , and z are also the same as if they are defined in standard VRML and X3D.

4.3.2 Function scripts

Usually defining complex geometry, colour, geometric texture or operation assumes the usage of multiple formulae, instead of one simple formula. Moreover, for certain functions, it is required to have temporary variables, flow controls, and even sub functions with recursions. All these are elements of a programming language which can define a function procedurally.

Therefore, certain programming language must be included in FVRML and FX3D. It is known that in VRML and X3D, a scripting language named ECMAScript (which is actually a variant of JavaScript) is included as part of the standards. To ease the learning curve and to provide consistency with the original standards, the programming language in FVRML and FX3D are designed as an emulation of the standard ECMAScript. The syntax of this JavaScript-like language is exactly the same as the ECMAScript, except all the source codes must be written in functions. There is also no global variables provided. To simplify programming, this language is case-insensitive, and all variables, arrays and constants have only one type *float*.

This scripting language can be considered as an expansion of the analytical formulae. Every feature provided in the analytical formulae is available in the function scripts. When an author wants to define additional custom functions, the same way as defining a function in

Chapter 4. Function-based extension of VRML/X3D

ECMAScript can be used. All flow control mechanisms provided by ECMAScript are available in the function scripts, namely *for-loops*, *while-loops*, *do-while-loops*, *break*, *continue*, and *if-then-else*. Besides these ability, recursions are also allowed in the function scripts.

When the function scripts are used to define FRep functions, a function named ‘frep’ must exist in the function scripts. It can take up to 4 arguments, representing x , y , z , and t respectively. The return value of this function would be the function value of the defined FRep function.

When the function scripts are used to define parametric functions, several functions whose names start with ‘parametric_’ must exist. For geometries, geometric textures, translations and scalings, the function names must be ‘parametric_x’, ‘parametric_y’, and ‘parametric_z’. When defining rotations and scale orientations, in addition to the three functions, a function called ‘parametric_a’ must also exist. For colours, instead of these three names, ‘parametric_r’, ‘parametric_g’, and ‘parametric_b’ should be employed to define red, green, and blue component of the colour. All these functions can take up to 4 arguments, representing u , v , w , and t respectively.

When both FRep function names and parametric function names exist in one source code, the FRep function must be used, while other functions will be considered as custom sub functions used by the FRep function.

4.4 Prototypes

Based on the above design, six additional nodes of FVRML and FX3D are proposed. The proposed node prototypes are *FShape*, *FAppearance*, *FGeometry*, *FMaterial*, *FTexture3D* and *FTransform*.

In Figure 4.1, the scene diagrams defining these nodes, as well as their integration with standard VRML nodes *Transform* and *Shape* are given.

Chapter 4. Function-based extension of VRML/X3D

Figure 4.1: Scene diagrams of the function-based extension

Chapter 4. Function-based extension of VRML/X3D

FShape node is a container similar to the VRML's and X3D's *Shape* node. It contains *FGeometry* or any standard geometry node (e.g. *IndexedFaceSet* node), and *FAppearance* or the standard *Appearance* node. These nodes define the geometry and the appearance of the shape, respectively. *FShape* may be called from *FTransform* node or from the standard *Transform* node.

FGeometry node contains information about a function-defined geometry including type, definition of function, parameters, rendering resolution and bounding box. Alternatively, the function definition can be also given in a text file pointed by its URL.

FAppearance node may contain *FMaterial* or the standard *Material*, the standard *Texture* node, and *FTexture3D* node. *FTexture3D* defines geometric textures while the standard *Texture* node is used for image texture mapping.

FMaterial node is similar to the *Material* node except that the numeric fields (i.e. *diffuseColor*, *emissiveColor*, *specularColor*, *ambientIntensity*, *transparency*, and *shininess*) can be defined by functions. These fields can also be a URL representing a text file. A colour interpolator is also provided by *patternKey* and *patternColor* fields. This colour interpolator can simplify the job of interpolating colours. For each float number in *patternKey* matching the colour function values, the respective colour is to be defined in *patternColor*. The actual colour is then linearly interpolated following the function values. As it is used in the standard *ColorInterpolator* node, the colour interpolation is done in the HSV space instead of the RGB space.

FTexture3D node defines 3D geometric textures using explicit and parametric functions. These functions define displacements of the original shape's geometry defined in *FGeometry* or in any standard geometry node (i.e. polygonal meshes). The functions are defined as a string, or via a URL to the function definition. If the standard geometry node is used in place of *FGeometry* node, the standard shapes of VRML and X3D will be assigned an appearance

Chapter 4. Function-based extension of VRML/X3D

defined in *FAppearance* node. This ability to use the function-based nodes with the standard geometry and appearance nodes of VRML and X3D adds many new features to VRML and X3D.

FTransform node contains function-defined operations, as well as fields of *Transform* node except *bboxSize* and *bboxCenter* because the bounding box will be calculated automatically. It may only contain several *FShape* and *FTransform* nodes as its children. The operation is either the function defining the operation applied on the children, or an URL pointing to a text file. In order to be consistent with the standard *Transform*, affine transformations are provided as well. The affine transformations can also be defined as functions in *FTransform* node.

4.4.1 FVRML prototype specifics

As our prototypes are designed an extension to the standard VRML language, it is very important to keep the prototypes consistent with their corresponding standard nodes. Also, since there are two time control mechanisms defined in VRML, both of them should be provided by the prototypes. Moreover, *cycleTime* field is missing in the standard self-contained time-dependent nodes. Since such field can be very useful in many cases, the function-based extension prototypes should include this field. In the prototype, *timeSpan* defines the range of the time used in visualisation. Event *set_fraction* is designed to communicate with an external *TimeSensor* whose *fraction_changed* event will be routed. Other fields and events are designed to be consistent with the standard *TimeSensor*. By providing these fields and events, our prototypes can be used in both self-contained way (with no external *TimeSensor*) and non-self-contained way (with external *TimeSensor*).

In Figure 4.2 and Figure 4.3, the VRML prototypes of the proposed extension are listed. The definitions of all fields in the prototypes can be seen in Table 4.1 at the end of this chapter.

Chapter 4. Function-based extension of VRML/X3D

```

PROTO FGeometry [
    exposedField SFString definition ""
    exposedField MFFloat parameters [0]
    exposedField MFInt32 resolution [50 50 50]
    exposedField SFVec3f bboxCenter 0 0 0
    exposedField SFVec3f bboxSize 10 10 10
    exposedField SFString type ""
    exposedField MFNode parents []
    exposedField SFVec2f timeSpan 0 1
]
PROTO FAppearance [
    exposedField SFNode material NULL
    exposedField SFNode texture NULL
    exposedField SFNode textureTransform NULL
    exposedField SFNode texture3D NULL
    exposedField MFNode parents []
    eventIn SFBool refresh
]
PROTO FMaterial [
    exposedField SFString diffuseColor ""
    exposedField SFString type ""
    exposedField MFCOLOR patternColor []
    exposedField MFFloat patternKey []
    exposedField SFString ambientIntensity ""
    exposedField SFString emissiveColor ""
    exposedField SFString shininess ""
    exposedField SFString specularColor ""
    exposedField SFString transparency ""
    exposedField MFNode parents []
    exposedField SFVec2f timeSpan 0 1
]
PROTO FTexture3D [
    exposedField SFString definition ""
    exposedField SFString type ""
    exposedField MFFloat parameters [0]
    exposedField MFNode parents []
    exposedField SFVec2f timeSpan 0 1
]

```

Figure 4.2: Prototypes of the FVRML

Chapter 4. Function-based extension of VRML/X3D

```

PROTO FShape [
    exposedField SFNode geometry NULL
    exposedField SFNode appearance NULL
    exposedField SFString polygonizer ""
    exposedField MFNode parents []
    eventIn SFBool refresh
    exposedField SFTIME startTime 0
    exposedField SFTIME stopTime 0
    exposedField SFTIME cycleInterval 1
    exposedField SFBool loop FALSE
    exposedField SFBool enabled TRUE
    eventOut SFTIME cycleTime
    eventIn SFFloat set_fraction
]
PROTO FTransform [
    exposedField SFString operation ""
    exposedField SFVec3f center ""
    exposedField SFString rotation ""
    exposedField SFString scale ""
    exposedField SFString scaleOrientation ""
    exposedField SFString translation ""
    exposedField SFString type ""
    exposedField SFString polygonizer ""
    exposedField MFFloat parameters [0]
    exposedField MFNode children []
    exposedField MFNode parents []
    eventIn SFBool refresh
    exposedField SFVec2f timeSpan 0 1
    exposedField SFTIME startTime 0
    exposedField SFTIME stopTime 0
    exposedField SFTIME cycleInterval 1
    exposedField SFBool loop FALSE
    exposedField SFBool enabled TRUE
    eventOut SFTIME cycleTime
    eventIn SFFloat set_fraction
]

```

Figure 4.3: Prototypes of the FVRML (continue)

4.4.2 FX3D prototype specifics

In FVRML, six nodes were proposed, which are *FShape*, *FGeometry*, *FMaterial*, *FTexture3D*, *FAppearance*, and *FTransform*. These nodes were designed according to the VRML97 standard. As a further development of FVRML, the function-based extension of X3D should still incorporate these six nodes in order to provide backward compatibility to FVRML in function-based extension of X3D. However, compared to VRML, X3D not only introduces new features, but changes some existing nodes as well. Therefore, simple porting of FVRML to X3D would not comply with X3D standard any more.

Before proposing the concrete node definition, we must first understand what abstract node types needed by function-based extension of X3D should be introduced. Though the function-based extension nodes are designed according to the standard X3D nodes, such as *Shape*, *Material*, *Appearance*, and *Transform*, there are slight differences due to the nature of function-based shape modelling technique. The most significant problem is that the function-based extension only supports operations on function-defined models, not a mixture of function-defined models and polygon meshes. This restriction can be seen in the scene diagram (Figure 4.1). Therefore, *X3DChildNode* is not suitable as the type of the extension, since it may cause incorrect usage of operations on polygon meshes. However, *FShape* and *FTransform* should be able to cooperate with the standard grouping node. Based on this consideration, the *X3DFunctionChildNode* extending *X3DChildNode* is proposed.

FShape node is designed to associate a function-defined or polygonal geometry and a function-defined or standard appearance, which is quite similar to *Shape* node. Therefore, it derives from the standard node type *X3DShapeNode*. In addition to the information of geometry and appearance, the function-defined object may be time-dependent, and should be able to be used independently. Thus, *FShape* must also implement *X3DTimeDependentNode*

Chapter 4. Function-based extension of VRML/X3D

type. Finally, *FShape* may be incorporated as a child of a function-defined operation, which results in the implementation of *X3DFunctionChildNode*.

FGeometry node is designed to define a geometry using either FRep functions or parametric functions. Since function-defined geometry can be regarded as a special kind of 3D geometry in X3D, it should be derived from *X3DGeometryNode* type.

FAppearance node represents the function-defined or standard material, the function-defined 3D geometric texture, and other normal appearance components (e.g. 2D/3D image texture, texture transformation, etc.). Hence, a property *texture3D* is included in addition to all fields in the *Appearance* node.

FMaterial node is designed in exactly the same way as the FVRML prototype, except that it is derived from *X3DMaterialNode*, which is the super type of *Material* node in X3D.

FTexture3D does not have corresponding standard node in X3D. It stores function definitions of a 3D geometric texture, such as vertex displacements. As 3D geometric textures are parts of the shape appearance, the super type of *FTexture3D* is *X3DAppearanceChildNode*, which is designed to be directly nested under *X3DAppearanceNode*.

FTransform defines both transformations (i.e. translation, rotation, and scaling) as well as operations of function-defined shapes. *X3DTimeDependentNode* should also be implemented in this node, because of the fact that the operation of function-defined shapes may be time-dependent. Moreover, because of the proposal of function-defined operations, in the function-based extension of X3D, only function-defined nodes can be used under *FTransform*.

All six proposed prototypes in function-based extension of X3D are listed in Figure 4.4 and Figure 4.5. The definitions of the fields are exactly the same as the FVRML prototypes (as listed in Table 4.1). Fields which do not exist in FVRML prototypes have the same meaning as the corresponding standard X3D fields.

4.5 Summary

In order to test the feasibility of the hybrid function-based web shape modelling framework, a function-based extension of VRML and X3D is proposed. In this extension, authors are allowed to freely mix the usage of FRep functions and parametric functions to define geometries, colours, geometric textures, operations, and affine transformations. Furthermore, standard VRML and X3D nodes are also allowed to be used in the FVRML and FX3D. This provides a possibility to re-use the existing models with advanced features provided by the hybrid function-based web shape modelling framework.

The extension is designed to be highly compatible with VRML and X3D standards. Six prototypes are proposed according to the corresponding nodes in the standards. All the possible fields in the standard VRML and X3D are designed to be able to be defined by functions. In this extension, two approaches are proposed to define functions, which are analytical formulae for novice user and simple objects, and function scripts for experienced user and complex objects. Finally, the scene diagrams and the prototypes in the extension are illustrated and discussed.

The extension shares some common aspects with the BlobTree, which was introduced in [127] as an extension of VRML. They have the similar hierarchical structure, enable the usage of function-defined objects, perform operations on the function-defined objects, and follow the design of standard VRML nodes. However, the extension presented in this thesis has several significant differences with the BlobTree. First, the extension supports the new XML-encoded X3D standard, whereas the BlobTree only extends VRML. Second, FVRML/FX3D allows using any type of functions (parametric, implicit, and explicit) to define shapes and appearances, whereas the BlobTree only allows skeleton-based implicit function-defined models to be used. Third, FVRML/FX3D allows defining any customised operations by writing the formula directly without further programming, whereas the BlobTree only has a limited

Chapter 4. Function-based extension of VRML/X3D

set of pre-defined operations. Finally, the time parameter is introduced in FVRML/FX3D, which can be used to define animations directly in the function.

Chapter 4. Function-based extension of VRML/X3D

```

X3DFunctionChildNode : X3DChildNode {
  SFNode    [in,out] metadata    NULL          [ X3DMetadataObject ]
}
FGeometry : X3DGeometryNode {
  SFNode    [in,out] metadata    NULL          [ X3DMetadataObject ]
  SFString  [in,out] definition  " "
  MFFloat   [in,out] parameters [0]             (−∞,∞)
  MFInt32   [in,out] resolution [50 50 50]      (0,∞)
  SFString  [in,out] type        " "
  SFVec2f   [in,out] timeSpan    0 1            (−∞,∞)
}
FAppearance : X3DAppearanceNode {
  SFNode    [in,out] metadata    NULL          [ X3DMetadataObject ]
  SFNode    [in,out] fillProperties NULL        [ FillProperties ]
  SFNode    [in,out] lineProperties NULL        [ LineProperties ]
  SFNode    [in,out] material     NULL          [ FMaterial ]
  SFNode    [in,out] texture      NULL          [ X3DTextureNode ]
  SFNode    [in,out] textureTransform NULL      [ X3DTextureTransformNode ]
  SFNode    [in,out] texture3D    NULL          [ FTexture3D ]
}
FMaterial : X3DMaterialNode {
  SFNode    [in,out] metadata    NULL          [ X3DMetadataObject ]
  SFString  [in,out] ambientIntensity " "
  SFString  [in,out] diffuseColor  " "
  SFString  [in,out] type          " "
  MFFloat   [in,out] patternKey    []            (−∞,∞)
  MFColor   [in,out] patternColor  []            [0,1]
  SFString  [in,out] emissiveColor " "
  SFString  [in,out] shininess     " "
  SFString  [in,out] specularColor " "
  SFString  [in,out] transparency  " "
  SFVec2f   [in,out] timeSpan      0 1            (−∞,∞)
}
FTexture3D : X3DAppearanceChildNode {
  SFNode    [in,out] metadata    NULL          [ X3DMetadataObject ]
  SFString  [in,out] definition  " "
  SFString  [in,out] type        " "
  MFFloat   [in,out] parameters [0]             (−∞,∞)
  SFVec2f   [in,out] timeSpan    0 1            (−∞,∞)
}

```

Figure 4.4: Prototypes of the FX3D

Chapter 4. Function-based extension of VRML/X3D

```

FShape : X3DShapeNode, X3DTimeDependentNode, X3DFunctionChildNode {
  SFNode    [in,out] appearance NULL          [FAppearance]
  SFNode    [in,out] geometry   NULL          [FGeometry]
  SFNode    [in,out] metadata   NULL          [X3DMetadataObject]
  SFVec3f   [in,out] bboxCenter 0 0 0          ( $-\infty, \infty$ )
  SFVec3f   [in,out] bboxSize  10 10 10       [0,  $\infty$ )
  SFString  [in,out] polygonizer "analytical"
  SFTIME    [in,out] cycleInterval 1          (0,  $\infty$ )
  SFBool    [in,out] enabled    TRUE
  SFBool    [in,out] loop       FALSE
  SFTIME    [in,out] pauseTime  0              ( $-\infty, \infty$ )
  SFTIME    [in,out] resumeTime 0              ( $-\infty, \infty$ )
  SFTIME    [in,out] startTime  0              ( $-\infty, \infty$ )
  SFTIME    [in,out] stopTime   0              ( $-\infty, \infty$ )
  SFTIME    [out]   cycleTime
  SFTIME    [out]   elapsedTime
  SFBool    [out]   isActive
  SFBool    [out]   isPaused
  SFFloat   [in]    set_fraction
  SFInt32   []      frames      0              [0,  $\infty$ )
}

FTransform : X3DTimeDependentNode, X3DFunctionChildNode {
  SFNode    [in,out] metadata   NULL          [X3DMetadataObject]
  SFString  [in,out] operation  "union"
  SFString  [in,out] type       "setoperator"
  MFFloat   [in,out] parameters [0]              ( $-\infty, \infty$ )
  SFString  [in,out] center     " "
  SFString  [in,out] rotation   " "
  SFString  [in,out] scale      " "
  SFString  [in,out] scaleOrientation " "
  SFString  [in,out] translation " "
  MFNode    [in,out] children   []              [X3DFunctionChildNode]
  SFString  [in,out] polygonizer "analytical"
  SFTIME    [in,out] cycleInterval 1          (0,  $\infty$ )
  SFBool    [in,out] enabled    TRUE
  SFBool    [in,out] loop       FALSE
  SFTIME    [in,out] pauseTime  0              ( $-\infty, \infty$ )
  SFTIME    [in,out] resumeTime 0              ( $-\infty, \infty$ )
  SFTIME    [in,out] startTime  0              ( $-\infty, \infty$ )
  SFTIME    [in,out] stopTime   0              ( $-\infty, \infty$ )
  SFTIME    [out]   cycleTime
  SFTIME    [out]   elapsedTime
  SFBool    [out]   isActive
  SFBool    [out]   isPaused
  SFFloat   [in]    set_fraction
  SFInt32   []      frames      0              [0,  $\infty$ )
}

```

Figure 4.5: Prototypes of the FX3D (continue)

Chapter 4. Function-based extension of VRML/X3D

Table 4.1: FVRML/FX3D field definitions

<i>definition "frep(x,y,z,t)" or</i> <i>definition "frep(x,y,z,t){</i> <i>script;return frep(x,y,z,t);}" or</i> <i>definition "x=x(u,v,w,t);</i> <i>y=y(u,v,w,t);z=z(u,v,w,t);" or</i> <i>definition "</i> <i>function parametric_x(u,v,w,t){</i> <i>script;return x(u,v,w,t);}</i> <i>function parametric_y(u,v,w,t){</i> <i>script;return y(u,v,w,t);}</i> <i>function parametric_z(u,v,w,t){</i> <i>script;return z(u,v,w,t);}"</i>	<p>In this field analytical definition of the shape's geometry or geometric texture has to be defined. It can be given either as individual implicit or parametric formulae or as a function script. The names of Cartesian coordinates are x, y, z, while the names of parametric coordinates are u, v, w. Parameter t is the time. The number of variables is optional. For defining surfaces and solid objects, x, y, z variables are to be used. For defining parametric surfaces, two of the parametric coordinates are to be used (e.g., u and v). For defining curves, only one of the parametric coordinates is to be used. New variable names can be introduced in function scripts as it is normally done in JavaScript. The formulae can be also called by a URL.</p>
<i>parameters [p1 p2 p3 p4 p5 p6]</i> <i>Default: parameters [-1 1 -1 1 -1 1]</i> <i>continuity 0</i>	<p>It defines parameters' range for u, v, w and continuity of the predefined set-theoretic operations. If the continuity equals to 0, the set-theoretic operations are implemented with <i>min/max</i> functions. If the continuity is greater than 0, they are implemented with the so-called <i>R</i>-functions.</p>
<i>bboxCenter x y z</i> <i>Default: bboxCenter 0 0 0</i>	<p>It defines the origin of the coordinate system of the implicitly/explicitly defined surface.</p>
<i>bboxSize x y z</i> <i>Default: bboxSize 10 10 10</i>	<p>It defines the size of the bounding box, and thus the range of variables x, y, z for the implicitly/explicitly defined shape.</p>
<i>resolution [x y z]</i> <i>Default: resolution [50 50 50]</i>	<p>It defines the rendering resolution. There must be 3 values for solid shapes, 2 values for parametric surfaces, and 1 value for parametric curves.</p>
<i>diffuseColor "frep(x,y,z,t)" or</i> <i>diffuseColor "frep(x,y,z,t){</i> <i>script;return frep(x,y,z,t);}" or</i> <i>diffuseColor "r=r(u,v,w,t);</i> <i>g=g(u,v,w,t);b=b(u,v,w,t);" or</i> <i>diffuseColor "</i> <i>function parametric_r(u,v,w,t){</i> <i>script;return r(u,v,w,t);}</i> <i>function parametric_g(u,v,w,t){</i> <i>script;return g(u,v,w,t);}</i> <i>function parametric_b(u,v,w,t){</i> <i>script;return b(u,v,w,t);}"</i>	<p>In this field the function definition of the shape's diffuse colour is to be given. It can be typed either as individual explicit or parametric formulae or as a function script. Cartesian coordinates are x, y, z, while the names of parametric coordinates are u, v, w. Ranges of the variables can be defined in parameters, <i>bboxCenter</i>, and <i>bboxSize</i> fields. Parameter t is the time. The formulae can be also called by a URL. In case if the colour is defined by an explicit function, colour interpolation will be done using the values defined in fields <i>patternKey</i> and <i>patternColor</i>.</p>

Chapter 4. Function-based extension of VRML/X3D

<code>emissiveColor "frep(t)"</code> or <code>emissiveColor "frep(t){</code> <code>script;return frep(t);}"</code> or <code>emissiveColor "r=r(t);</code> <code>g=g(t);b=b(t);"</code> or <code>emissiveColor "</code> <code>function parametric_r(t){</code> <code>script;return r(t);}</code> <code>function parametric_g(t){</code> <code>script;return g(t);}</code> <code>function parametric_b(t){</code> <code>script;return b(t);}"</code>	<p>It defines a time-dependent emissive colour by an explicit function or script. In case of the colour definition by one explicit function, colour interpolation will be done by the current pattern. Parameter t is the time. The formulae can be also called by a URL. The emissive colour is assigned to the whole shape and cannot vary across it. In case if the colour is defined by an explicit function, colour interpolation will be done using the values defined in fields <i>patternKey</i> and <i>patternColor</i>.</p>
<code>specularColor "frep(t)"</code> or <code>specularColor "frep(t){</code> <code>script;return frep(t);}"</code> or <code>specularColor "r=r(t);</code> <code>g=g(t);b=b(t);"</code> or <code>specularColor "</code> <code>function parametric_r(t){</code> <code>script;return r(t);}</code> <code>function parametric_g(t){</code> <code>script;return g(t);}</code> <code>function parametric_b(t){</code> <code>script;return b(t);}"</code>	<p>It defines a time-dependent specular colour by an explicit function or script. In case of the colour definition by one explicit function, colour interpolation will be done by the current pattern. Parameter t is the time. The formulae can be also called by a URL. The specular colour is assigned to the whole shape and cannot vary across it. In case if the colour is defined by an explicit function, colour interpolation will be done using the values defined in fields <i>patternKey</i> and <i>patternColor</i>.</p>
<code>ambientIntensity "frep(t)"</code>	<p>It defines a time-dependent ambient intensity by an explicit function or a script. The ambient intensity is assigned to the whole shape.</p>
<code>shininess "frep(t)"</code>	<p>It defines a time-dependent shininess by an explicit function or a script. The shininess is assigned to the whole shape.</p>
<code>transparency "frep(t)"</code>	<p>It defines a time-dependent transparency by an explicit function or a script. The transparency is assigned to the whole shape.</p>
<code>patternKey [a₁ a₂ a₃ ... a_n]</code>	<p>It defines the values of the explicit <i>diffuseColor</i>, <i>emissiveColor</i> or <i>specularColor</i> function or a script which will be used for mapping to <i>patternColor</i>.</p>
<code>patternColor</code> <code>[r₁ g₁ b₁ r₂ g₂ b₂ ... r_n g_n b_n]</code>	<p>It defines the colours which will be used for interpolating the colour defined by an explicit <i>diffuseColor</i>, <i>emissiveColor</i> or <i>specularColor</i> function or a script based on the current <i>patternKey</i>. Linear interpolation of colours will be done.</p>

Chapter 4. Function-based extension of VRML/X3D

<i>f</i> ...; <i>patternValue</i> (<i>f</i>);	The function is used in scripts to generate colours defined by <i>patternKey</i> and <i>patternColor</i> . The colour is stored in variable <i>r</i> , <i>g</i> , <i>b</i> .
<i>translation</i> "x=x(t);y=y(t);z=z(t);"	It defines translation parameters by explicit functions or scripts.
<i>scale</i> "x=x(t);y=y(t);z=z(t);"	It defines scaling parameters by explicit functions or scripts.
<i>rotation</i> "x=x(t);y=y(t);z=z(t);a=a(t);"	It defines rotation parameters by explicit functions or scripts.
<i>center</i> "x=x(t);y=y(t);z=z(t);"	It specifies a function-defined translation offset from the origin of the local coordinate system (0, 0, 0).
<i>scaleOrientation</i> "x=x(t);y=y(t);z=z(t);a=a(t);"	It specifies a function-defined rotation of the coordinate system before the scale.
<i>operation</i> "union intersection difference" or <i>operation</i> "frep(f,g,t){ script;return frep(f,g,t);}" <u>Default</u> : <i>operation</i> "union" parameters [0]	In this field either a name of the predefined set-theoretic operation or an individual formula or a script defining a set-theoretic operation on shapes defined by functions $f(x, y, z, t)$ and $g(x, y, z, t)$ have to be typed. The respective <i>FShapes</i> are to be defined as children of <i>FTransform</i> node. Set-theoretic operations do not apply to parametrically defined shapes.
<i>timeSpan</i> t_1 t_2 <u>Default</u> : <i>timeSpan</i> 0 1	It defines a time interval of variable <i>t</i> when defining a time-dependent process (shape's geometry, colour, texture or transformation). Variables t_1 and t_2 can take any real values including negative.
<i>startTime</i> <i>t</i> <u>Default</u> : <i>startTime</i> 0	It defines the values of time to start a time-dependent process.
<i>stopTime</i> <i>t</i> <u>Default</u> : <i>stopTime</i> 0	It defines the values of time to stop a time-dependent process.
<i>cycleInterval</i> <i>t</i> <u>Default</u> : <i>cycleInterval</i> 1	It defines the duration of the time-dependent process cycle in seconds.
<i>frames</i> <i>n</i> <u>Default</u> : <i>frames</i> 0	It defines the number of frames to be cached. If 0 is defined, rendering time-dependent processes will be performed on the fly.
<i>loop</i> TRUE FALSE <u>Default</u> : <i>loop</i> FALSE	It enables or disables looping of time-dependent processes.
<i>enabled</i> TRUE FALSE <u>Default</u> : <i>enabled</i> TRUE	It enables or disables time-dependent processes.

Chapter 5

Implementation of FVRML/FX3D

This chapter focuses on the more detailed information on how to implement FVRML and FX3D proposed in Chapter 4. In order to illustrate that FVRML and FX3D can be actually implemented on various platforms, two different implementations are presented in this chapter. In Section 5.1, some general guidelines and goals of the implementations are discussed. All the FVRML and FX3D implementations should follow the guidelines closely. Next, the general function-based web visualisation pipeline is proposed in Section 5.2. Both the two actual extensions presented in this chapter follow this pipeline. As there are several ways to actually extend standard VRML and X3D, the advantages and disadvantages of these approaches are discussed in Section 5.3. Finally, the detailed technical information and some optimisation methods to accelerate the process are shown in Section 5.4 and Section 5.5.

5.1 General considerations

When proposing the design of the function-based extension of VRML and X3D, the extension is assumed to be browser-independent and system-independent. There must be no limitations or restrictions on the implementation platform and system, since there exist multi-platform VRML/X3D browsers. In order to illustrate the possibility to build the extension on multiple

Chapter 5. Implementation of FVRML/FX3D

platforms, several implementations should be done using different development environments and browsers.

When developing the extension implementation, it is improper to create a new VRML/X3D browser. If a new browser is created, the user must learn how to use the new browser before they can take the advantage of FVRML/FX3D. Furthermore, the authors might have to use the features provided by a specific browser, and such feature does not exist in the standards. Therefore, the extension implementation should be a plugin to the existing browser.

The browser plugin should be easy for user to install. It can either be automatically downloaded and installed, or manually installed once. After the installation, it should not require to be downloaded again, and thus become “transparent” to the user (i.e. the plugin would act as if it is already part of the browser). Otherwise, it would be very inconvenient to download the same piece of software each and every time when visualising a function-defined model.

Another important aspect of the extension implementation is the backward compatibility. When the plugin is not installed on the client computer, the function-defined model should just be invisible without interfering with other standard objects. Under this circumstance, there should be no error messages or crashes as well.

When converting functions into polygons, many existing algorithms can be used, such as Marching Cubes [107] algorithm for rendering shapes defined by FRep functions, and even-tessellation polygonisation algorithm for parametric functions. Additional parameters of the functions needed by polygonisation algorithms are expected to be provided by the user, such as the domains of the functions, the resolution of the generated model, etc, since certain combination of these parameters may achieve special effect required by user. If these tuneable parameters are not provided, default values should be employed instead of raising an error, though the default values may not be suitable for certain special functions.

5.2 Modules and visualisation pipeline

To visualise the function-defined objects in VRML and X3D, we have to convert them into the standard VRML and X3D primitive objects (lines and polygons) which will be then rendered by the browser. In addition, this process should not block the user's interaction with the standard objects. The visualisation pipeline is illustrated in Figure 5.1. The proposed visualisation pipeline is a generalised function-based web visualisation pipeline, which can be used not only VRML/X3D, but also other platforms and browsers. The only requirement for the platforms is the ability to render primitive objects (e.g. polygons, points, voxels).

The diagram shows how function definitions can cooperate with the VRML and X3D browsers. Solid lines represent the execution flow, and dotted lines represent the communication between the threads. In this visualisation pipeline, two independent threads are used. One thread is the main thread spawned by the browser itself. The other thread is created by the extension itself when it is initialised by the browser. Usually, the thread created by the browser is the main execution thread of all scripts in the VRML/X3D file. Since the extension design also considers the possibility to define time-dependent objects, the execution of the scripts should never be blocked by the generation of the animation. If only one thread is used, when the process is time-consuming (e.g. polygonising the function-defined animation continuously), the whole scene freezes. On the contrary, if an additional thread is employed, user can still manipulate objects and navigate in the scene, while the function definition is still being processed. In the presented pipeline, the browser-created thread is responsible to parse the function definition in the VRML/X3D file, and compile the function definition into an appropriate format to be ready to evaluate. Usually, the parsing and compilation process can be finished within a second, therefore, it is safe to put them into the main thread while the user will not feel the difference. All other processes, such as function evaluation and polygonisation, are done in a separate thread without interfering user's interaction.

Figure 5.1: The visualisation pipeline of the implemented extension

Chapter 5. Implementation of FVRML/FX3D

There are eight different modules in this pipeline, which are: *Parser*, *Compiler*, *TimeGenerator*, *ShapeGenerator*, *TextureGenerator*, *ColorGenerator*, *TransformOperator*, and *PrimitiveGenerator*.

Parser is responsible to extract the function definition from the VRML and X3D file. The function definition does not only include the source code of the function, but also additional parameters, such as the range of the function, the parameters used by the function, etc. When the function definition is changed (i.e. the source code or the additional parameters are changed), *Parser* is also responsible to notify other modules to update their status or redo the whole process using the new information.

Compiler is responsible to compile the function definition extracted by *Parser*. After compilation, the function definition should be in a form that it is ready to be evaluated by other modules at any time. When the function definition is changed, the compilation is redone in order to evaluate the new defined functions.

TimeGenerator gives out the time information to other modules when any of the functions is a time-dependent function. When the function-defined object is static, this module always gives 0 as the time information should be ignored.

ShapeGenerator is responsible to identify the type of the function generated by the compiler. In the extension design, two function types are provided, parametric or FRep. When the function generated by the compiler is a parametric function, input coordinates would be interpreted as the coordinates in the 3D parametric space. The function values will be the 3D Cartesian coordinates. On the other hand, if the function generated by the compiler is an FRep function, input coordinates will be interpreted as the 3D Cartesian coordinates. The function value will be a single real number.

TextureGenerator receives and modifies the geometry interpreted by *ShapeGenerator*. Essentially, it modifies the function value calculated by the *ShapeGenerator*. In addition to

Chapter 5. Implementation of FVRML/FX3D

identifying the type of the texture function, it also has to recognise the type of the geometry function in order to modify the function value from *ShapeGenerator* correctly. Since the domain of the geometric texture and the domain of the geometry might be different, these domains are mapped via linear mapping. The result would still have the same domain as the original geometry. When there is no geometric texture defined for the object, the output generated by *ShapeGenerator* is simply passed through.

ColorGenerator appends the colour information to the textured geometry made by *TextureGenerator*. The domain of the colour function is mapped to the domain of the geometry linearly. In this module, there is no geometric modification to the textured geometry. It only associates the evaluated coordinates with a 3D vector representing the RGB colour of the given coordinates. When the colour function is an FRep function, the RGB colour is linearly interpolated using the *ColorInterpolator* defined by given pattern keys and colours.

TransformOperator applies either pre-defined or user-defined operations onto the hybrid function-based models. In the existing implementation, operations can only be applied to the same type of functions. The result of this module is still a single function-defined geometry together with its colour information. When calculating the resulting object, certain fields will be calculated automatically.

PrimitiveGenerator converts all function defined geometries and properties into the primitives that are supported by VRML/X3D rendering engine. In this version, two primitives are supported, i.e. lines and polygons. These output primitives are passed back to the VRML/X3D browser. Thus, the function-based models can be visualised and manipulated as if they are objects created by traditional modelling techniques.

In the visualisation pipeline, since the extension supports combination of standard nodes and function-based nodes, when some parts are defined by traditional methods instead of functions, the corresponding module is automatically by-passed. Furthermore, to provide

Chapter 5. Implementation of FVRML/FX3D

more flexibility, *Parser*, *Compiler*, *ShapeGenerator*, *TextureGenerator*, *ColorGenerator*, *TransformOperator* and *PrimitiveGenerator* (shown as grey boxes) are pluggable since the functionality of these modules may need to be changed. *TimeGenerator* module is still hard-coded in the implementation since it has a fixed task of generating time signals synchronously with the real time.

5.3 Ways to extend VRML/X3D

The implementation of FVRML and X3D is targeted at both shared worlds and general 3D web visualisation communities. Therefore, the main concern of the implementation is the compatibility. Such implementation should support major browsers which may be based on different platforms, and browsers which supports shared world communication. According to these criteria, different ways of extending VRML and X3D are examined first. The common way to extend VRML and X3D would be *external API*, *Script node*, and *Browser-specific API*. All the approaches would be analysed in this section.

5.3.1 External API

In both VRML and X3D standards, there is a way to embed the browser into a stand-alone application or a web-based applet. In this mechanism, the browser component can be used by the application as a native component via Component Object Model (COM) interface, or as a cross-platform component via the Java programming binding. Because of this, virtually any programming language can be used to develop the application and/or applet using the browser.

When the software needs to communicate with the embedded VRML/X3D browser, a special programming interface is used. Such API is defined as External Authoring Interface in VRML, and Scene Access Interface in X3D. Since the interface is standardised, the software

Chapter 5. Implementation of FVRML/FX3D

is not bound to a specific VRML/X3D browser. When running on the client computer, it is even possible to use a browser other than the browser used at the compilation time. Therefore, employing external API can ensure the compatibility throughout all the available browsers.

However, the external API can only be used when the browser is embedded into another software. If it is used to develop FVRML/FX3D implementation, it must be a separate application or applet. It is no longer a part of the original VRML/X3D browser. Moreover, when developing the extension in this way, authors not only have to change the VRML/X3D source files, but the web page layouts as well, since the web-based application or applet has to be added. Sometimes, it would be impossible to change the web page layouts, if it is generated by a programme on the server side. In conclusion, this approach is not suitable to implement the extension.

5.3.2 Browser-specific API

Browser-specific API implementation is done by developing an additional library using the software development kit (SDK) provided by specific browser vendors. The additional library can be highly integrated into the browser, and sometimes can even act as a part of the browser. These new features can be even standardised via components (i.e. a set of node types with similar functionalities), support levels (i.e. levels of features) and profiles (i.e. a self-contained combination of components and support levels) in X3D.

For certain open source browsers, such as FreeWRL [129] and Xj3D [130], the new component can be even merged into the source code of the software, instead of making an additional library. Thus, for the same programming language, browser-specific customisation has the best performance. Moreover, since the library is entirely integrated into the browser, nodes in the extensions can be used directly as the nodes in VRML and X3D standards, without declaring any prototypes. This benefits the users, since the syntax of using prototypes in the XML-encoded X3D file is very different from the existing nodes.

Though browser-specific customisation has many advantages, there are three major deficiencies, which prohibit us from using it in our implementation. First, the implementation using browser-specific customisation is not portable. Such implementation can only work for one kind of browser only, since the library must be highly-dependent on the browser implementation. Second, viewers are required to download and install either an external library or a whole customised browser before viewing the scene containing extended nodes. This may cause an inconvenience, since it is difficult for viewers to know what and where the library should be downloaded. Finally and the most important, when the viewer's browser does not have such an ability to support the extension, the scene will generate error messages, instead of ignoring the unsupported nodes.

5.3.3 Script node

Script is a node in the VRML/X3D standard which provides the authors with certain programming ability. Such programming ability can be used to develop advanced interactivity by dynamically creating, modifying, and/or deleting the nodes or fields in the scene. This feature can also be exploited to extend the standard together with the prototyping ability in VRML and X3D. The interface used to access the nodes and fields in the scene is called scripting interface in VRML. In X3D, the SAI would be used in the *Script* node as if it is accessed from an external application.

According to the standard specifications, compliant browsers should support both ECMAScript (i.e. a standardised JavaScript) and Java class files. Because of this, it is feasible to build prototypes encapsulating the *Script* nodes in order to implement FVRML and FX3D. Due to the fact that both JavaScript and Java class files are platform-independent, the implementations using SAI are automatically ready-to-use by any browser on any platform and architecture. Another advantage of the implementations using this approach is their installation-free feature. No software and other modules are needed to be downloaded and

Chapter 5. Implementation of FVRML/FX3D

installed, except the Java Runtime Environment (JRE), which is commonly pre-installed on personal computers. By using the *Script* node and the prototype mechanism, the prototypes, as well as the scripts, are downloaded automatically from the web by the browser. The whole process is absolutely transparent to the viewers without any interactions. Even for browsers, which do not fully support this approach, there will be no error messages and the extension nodes will be simply ignored.

For some browsers, it is even possible to develop scripts using C++ or other programming languages which generates native codes. This is done by writing a specific string in the *url* field of the *Script* node. The browser can automatically load a library file and execute the native codes in the library. This mechanism is called *native script*, and it is provided by many mainstream browsers in order to gain the performance advantage. The library files for different browsers have only slight differences, since they all use the same access interface as defined by the standards. Because of this, the source code can be ported with minor modifications. This becomes the best solution when a specific browser does not support Java class files in the *Script* node.

Though *Script* node approach has indeed many advantages over the other two methods, it still has one deficiency. Since it must employ prototypes to create the extension nodes, for XML-encoded X3D file a different syntax is employed compared to the standard nodes. This should not be a severe problem, since a detailed documentation can provide enough guidelines for the authors to use when X3D XML-encoding is needed.

5.4 Development platform

As all the possible ways to develop a VRML/X3D extension are carefully investigated and compared, the *Script* node approach appears to be the most suitable one to implement FVRML and FX3D. The only inconvenience would be the declaration of prototypes in each and every

Chapter 5. Implementation of FVRML/FX3D

VRML/X3D source file when FVRML or FX3D is used. Since the declaration of prototypes can be simply copied, and the size of the declaration will not be large, it remains the best solution. In our implementation, we choose this method.

To show the portability of the extension design, two implementations on different platforms using different programming languages are provided. They are Xj3D using Java, and BS Contact VRML/X3D using C++.

5.4.1 Xj3D and Java SAI

To make the hybrid function-defined models used in a larger area, all certified browsers should be able to support the function-based extension. As stated in the specification, all certified VRML/X3D browsers should support both ECMAScript and Java in its Script node. These two programming languages are platform-independent. Especially for Java, the binary class file is platform-independent too. This makes the source code invisible to the end-users, and it is good for protecting intellectual property. Nevertheless, due to the complexity of the extension, ECMAScript is not suitable for this implementation, since the extension involves many modules instead of a few individual functions. Other programming bindings may be supported by certain browsers, such as C++, and may have a better performance. However, since they are not fully standardised and are not platform-independent in binary form (executables or libraries), they are not considered when platform-independent choice is available for those browsers. Due to these reasons, we choose Java programming language for development.

As the scripting interface for VRML and SAI for X3D are fully standardised, any browser can be chosen as the development browser. In this case, Xj3D browser is chosen as the development environment. Xj3D browser is an example implementation of X3D browsers approved by the Web3D Consortium. It is fully written in Java, and provides almost all features defined by VRML and X3D standard. More importantly, it is recommended by the

Chapter 5. Implementation of FVRML/FX3D

Web3D Consortium for developers to create applications to support VRML/X3D technology. Therefore, it is the best platform to try out the function-based extension of VRML and X3D. The extension developed in this environment can be used by other certified browsers without any modification, even in the binary form (i.e. class files) without source codes.

The *Parser* module of the Java-based implementation is generated by *Yet Another Compiler Compiler* (YACC) for Java. YACC is de facto standard to create programming language parsers and compilers. The input file of YACC is a text file which contains the grammar definition of the programming language. The grammar definition is written in a form similar to Backus Naur Form (BNF). By running the YACC executable, the input file is converted into a Java source file, which can parse the grammar of the user-defined programming language. However, the generated source code still cannot interpret the grammar rules into executables. By filling this part by the developer, a compiler or parser can thus be built. Since both the analytical formulae and function scripts can be considered as a kind of programming language, YACC is employed here.

5.4.2 BS Contact and native script

As VRML and X3D are also widely used in shared virtual communities, it is also very important to enable the community users and authors to use function-based models. However, VRML/X3D standard and Xj3D browser do not natively support shared virtual communities. Thus, another VRML/X3D browser with shared virtual community support should be chosen as the development platform as well. BS Contact VRML/X3D is the tool which has extensions to support shared virtual communities. Together with blaxxun Communication Server, BS Contact VRML/X3D is able to see, chat, and interact with other users in the same environment. It is also possible to see interaction between other users and the objects in the scene. In Nanyang Technological University, it is already used in the Nanyang Technological

Chapter 5. Implementation of FVRML/FX3D

University Virtual Campus project [131]. Therefore, it is chosen as the development platform for shared virtual communities.

BS Contact VRML/X3D does not provide support for Java class files in its *Script* node. The Java-based implementation cannot be used by the browser. Though it is possible to use ECMAScript to implement the extension, as mentioned in previous section, it is not a reasonable solution due to the poor performance and the complexity of the extension. Thus, the second best choice is chosen, which is the native script.

In BS Contact VRML/X3D, by writing a specially crafted string in the *url* field of *Script* node, this browser can automatically load a Dynamic Link Library file, and execute the functions in the library. The library can be written in C++, Delphi, Visual Basic and other programming languages, which can generate native codes running on the machine. Therefore, it has the best performance.

Based on these requirements, the function-based extension of VRML and X3D based on BS Contact VRML/X3D browser is built using C++ programming language. The *Parser* module is implemented with help of *YACC* for C, which is similar to *YACC* for Java. Instead of generating Java source codes, it can generate C/C++ source codes for further development. For the sake of consistency, the same BNF grammar definition is used in the native implementation as it is used in the Java-based implementation.

5.5 Performance optimisation and analysis

When the hybrid function-defined model is time-dependent, the visualisation becomes a time-critical task. It is known that polygonisation of function-based models can be sometimes process demanding. Therefore, even the slightest delay in the polygonisation may affect the animation. In order to achieve smooth animation, optimisation of the extension must be

done. The most significant optimisation is an improvement of the function evaluation speed and caching the polygonised results.

5.5.1 Function evaluation optimisation

Improving the evaluation performance would be helpful to provide more smooth animation. It would also improve the user experience for static objects. However, for complex function definitions, interpreting the intermediate code on a virtual machine significantly slows down the polygonisation. One way to speed up the process is to optimise the intermediate code. However, this optimisation would be difficult to implement and it can only gain a little performance improvement. The other way is to employ Just-In-Time (JIT) compiling techniques [132], which has been already widely used by cross-platform languages such as Java and C#. JIT techniques dynamically convert the intermediate code into high-performance binary code, and then the assembly code is executed directly by computer. The optimised assembly code demonstrates the best performance.

For native script implementation, a customised JIT execution engine is built in the *Compiler* module. After compiling the function definition into intermediate codes, the JIT execution engine translates the intermediate codes into native assembly language automatically before evaluating the function. Since the function definition only involves floating-point number operations, in order to accelerate the evaluation process, the JIT execution engine employs the Streaming SIMD Extensions instructions in Intel x86 assembly language. The SSE is based on SIMD technology. A single SSE instruction can involve multiple floating-point numbers as the inputs, and it is commonly available on modern CPUs, such as Pentium III, AMD Athlon XP or higher. Using the SSE instructions can further boost the performance of the function evaluation.

For Java SAI implementation, the JIT execution engine is not provided directly. It is a known fact that Java cannot access arbitrary memory address and execute arbitrary code due

Chapter 5. Implementation of FVRML/FX3D

Table 5.1: VRML source file size (in bytes)

	Polygon-based		Function-based	
	Original	gzipped	Original	gzipped
Water Mellon	21,020,361	3,142,389	3,072	816
Heart	8,969,908	1,381,224	3,299	900
Fancy Flower	3,085,140	475,521	3,196	837

to the lack of pointers. So, it is impossible to convert the intermediate code into the native assembly language directly. However, for most Java Runtime Environment, the JIT engine is already available and embedded in order to gain the performance for Java programs. In order to employ the JIT engine in JRE, we managed to convert the intermediate code into a Java class file according the Java Virtual Machine Specification. The Java class file can then be loaded into the Java runtime environment and executed by the JIT engine in JRE.

To analyse the performance, the VRML platform and traditional polygon models are chosen for comparison purpose. By using the BS Contact VRML/X3D browser and the implemented FVRML, the statistics were obtained on a personal computer with Pentium 4 3.0G (with hyper-threading technology), 1 gigabytes memory and 80 gigabytes 7200RPM hard disk.

Since the VRML models are expected to be transmitted over network, the first important property should be the file size. The size of various models using polygon-based modelling technique and function-based modelling technique is shown in Table 5.1. For normal function-based models, the file size is around 3 kilobytes (including the external prototype declaration), however, the polygon-based models vary from several megabytes to tens of megabytes. Even with compression, usually the polygon-based models are over one megabyte. It is obvious that comparing to the traditional polygon-based models, function-based models can be very compact, which results the shorting transmission time.

Besides the transmission time, another important aspect is the rendering efficiency. This

Table 5.2: Rendering time for gzipped files (in seconds)

	Polygon-based	Function-based
Water Mellon	15	1.5
Heart	4	1.3
Fancy Flower	2	1

is measured locally, since we want to minimise the effect of different file size. For polygon-based models, the rendering process involves reading VRML source from hard disk, decompressing the gzipped file if compressed, parsing the content, and finally rendering the model. For function-based models, the rendering process involves reading source from hard disk, decompressing the gzipped file if compressed, parsing the content, converting the functions into polygons, and finally rendering the model. The rendering times for different models using different modelling techniques are shown in Table 5.2. From the results, we can conclude that function-based models can be efficiently rendered on modern personal computers equipped with a fast CPU.

5.5.2 Mesh caching

Generation of numerous primitive objects constituting animated shape metamorphoses requires very fast algorithms for making these primitive objects on the fly. If the polygonisation speed is faster than the rendering speed, it is not necessary to cache the frames, since the processing power is enough for real-time animation, and it would be merely a waste of memory. However, when computationally expensive mathematical functions and function scripts are used, caching of the meshes appears to be feasible. We implemented caching as follows. Whenever a mesh is required for a certain time parameter, it is first searched in the cache memory. If the mesh can not be found, it is generated and cached. If the mesh is available in the cache, we generate and cache the next unavailable mesh. Using this algorithm, a

Chapter 5. Implementation of FVRML/FX3D

progressive visualisation is achieved. Initially, the animation transition may be not smooth, but it improves gradually while all the required meshes are being generated and cached.

Due to the limited memory size, caching the FVRML/FX3D nodes using the internal format of the browser is not feasible. According to our experiments, a test mesh, which consists of 20,000 triangles, 10,000 vertices, 10,000 normals, and 10,000 colors, takes around 4,000,000 bytes memory space in the BS Contact VRML/X3D browser. It would take even more space if Xj3D browser were used because of the Java overhead. From this statistics, we estimated that a computer equipped with 1GB RAM can only cache a few hundreds of such meshes, which is very little for making any decent animation. Optimally, such polygon mesh should take only about 600,000 bytes if 32-bit integers and 32-bit floating point numbers were used to represent the indices, coordinates, normals, and colors. Compared to this, the overhead introduced by the browsers is very large. Since we are not able to alter the internal format of the browser to do any compression/decompression, we use our own data format to cache the generated meshes in order to eliminate the additional information stored by the browsers. In our data format we remove all unnecessary data introduced by the VRML/X3D internal mesh format from the cache. Therefore, the data format only stores coordinates, normals, and colors of the vertices, as well as indices of the polygon meshes. Such information is saved in their raw data form, i.e. 32-bit integers and 32-bit floating point numbers, instead of their string representations due to the memory efficiency consideration. By using this approach, the memory consumption of the same model is lowered to around 600,000 bytes, which is about 12% of the original memory consumption. In order to cache more generated meshes for a longer animation, the compression algorithms were taken into consideration as well. We found that the polygonisation algorithms for function-based models are prone to produce more vertices instead of more indices. Therefore, we do not use the mesh compression algorithms which mainly focus on compressing indices. Instead, we employ the gzip compression algorithm, which can be used on-the-fly. This can further re-

Chapter 5. Implementation of FVRML/FX3D

duce the memory consumption to around 400,000 bytes per mesh. Hence, the process of the caching mechanism can be described as follows. Once a polygon mesh is generated for an animated shape, the coordinates, normals, and colors of the vertices and the indices of the triangles are stored in the respective order using their raw data form in a data block. Such data block is compressed via gzip compression algorithm and then saved in the memory as a byte array. Whenever the mesh is requested by *TimeGenerator*, the byte array is decompressed and then converted into the browser's internal format using the provided APIs. Finally, the VRML/X3D node with the internal format can be rendered by the browser. By using this technique, 1GB memory is able to cache thousands of meshes similar to the tested one, which is sufficient for a reasonable animation.

When generating the cached frames, the rendering order of the frames should not be used. If the rendering order is used, since the polygonisation speed is slower than the rendering speed, the next rendering frame is always not available in the first round of the animation. Thus, we propose a binary subdivision algorithm (Algorithm 1) to generate cached frames. This algorithm is running in a thread parallel to the normal rendering thread.

Algorithm 1 Mesh caching algorithm

1. Set the current frame being generated (f) as the first frame: $f = 0$;
 2. Define the frame step (s) as the total number of frames;
 3. If the current frame is not generated yet, generate the frame;
 4. Set the current frame number as $f+s$;
 5. If the current frame is beyond the last frame, do: $s=s/2$, $f=0$;
 6. If the frame step is 0, terminate the process;
 7. Go to step 3.
-

The purpose of such algorithm is in giving the user a progressive delivery of the animation. Initially, the user can only see the first and the last frames. Gradually, the animation

Chapter 5. Implementation of FVRML/FX3D

Table 5.3: Performance of mesh caching

Time	Cached frames	Frame per second	Memory consumption	CPU usage
Any	No caching	4.875	29,028KB	98%
10.0000s	180	0.300	70,636KB	98%
20.0000s	180	4.400	113,260KB	98%
47.6084s	180	10.000	187,508KB	27%
102.1548s	360	20.000	351,064KB	49%

becomes smoother and smoother by inserting the missing frames in the middle. This algorithm also guarantees that every cached frame will be generated, since the frame step will become 1 before the process stops. When rendering the cached frames, it is possible that the frame for the exact moment does not exist. It might be either because the cached frame is not generated yet, or because such a frame will never be generated due to the limited number of cached frames. Instead of keeping the previous rendered frame, we try to render the nearest cached frame. The nearest cached frame can either a past or a next frame. When such frame cannot be found, no rendering will be done at all.

The performance of the caching can be seen in Table 5.3. The data is acquired using a Pentium 4 HT 3.0G with 1G DDR2 RAM and nVidia GeForceFX 5200 display adapter under Windows XP Service Pack 2. The animation is eight cones defined by explicit function scripts rotating around the origin. It contains 93024 triangles. One cycle of the animation lasts 18 seconds.

5.6 Summary

The function-based extension of VRML and X3D are implemented on two different platforms using different programming languages and technologies. These implementations show the portability of the extension design, and prove that the extension can be virtually implemented and used on any browsers and systems available.

Chapter 5. Implementation of FVRML/FX3D

To implement the extension, a function-based web visualisation pipeline based on existing VRML and X3D browsers is proposed. There are eight modules in the pipeline. To provide extensibility, most of the modules can be replaced in order to support other function-based model representation and visualisation techniques.

The implementations of FVRML/FX3D are also highly optimised, so that it can achieve less waiting time for static objects and smoother animation for time-dependent objects. Two optimisations are done in the implementations, JIT execution and mesh caching. JIT execution improves the function evaluation performance while mesh caching eliminates the possibility of re-polygonisation in looped animations.

The performance of the extension for both static objects and time-dependent objects is analysed. The experiments show that the hybrid function-based models indeed can improve the user experiences compared to the polygon-based models. It can also provide more features to the VRML/X3D authors to create complex models, either using pure functions or based on existing polygon-based models.

Chapter 6

Extensibility of FVRML/FX3D

In this chapter, the ability to extend the existing FVRML/FX3D implementations is presented. The extensibility applies to both Java-based and C++-based implementations as shown in Chapter 5. In Section 6.1, the ways to extend the existing implementation are described. The ability to replace modules in the FVRML/FX3D visualisation pipeline is discussed in Section 6.2. The possibility to add new customised function in the function scripts is also proposed in Section 6.3, including defining customised functions in the function script source code, and defining customised functions using Java or C++.

6.1 Overview

In Chapter 4 and Chapter 5, the design and implementation of the function-based extension of VRML and X3D are described. In the design and implementation, two different function representations are used, namely analytical formulae and function scripts. Also, Marching Cubes algorithm for FRep functions and tessellation algorithm for parametric functions are employed as the polygonisation algorithms to convert hybrid function-based models into polygon-based models, which can be directly visualised by the VRML and X3D browsers.

However, in the original proposal of the hybrid function-based web shape modelling

framework, there is no limitation or assumption of any function representations. Therefore, defining functions by analytical formulae or function scripts would not be enough to illustrate the full power of the modelling framework. Extending FVRML and FX3D would indeed benefit the web visualisation, since there exist many other function-based models which exploit different function representation approaches. The polygonisation algorithm should also be able to be replaced when it is necessary. Based on these considerations, the FVRML and FX3D implementations should have ways to be extended.

In the implementations presented in Chapter 5, extending FVRML and FX3D can be done by *module replacement* or *function script add-on*.

6.2 Module replacement

In the visualisation pipeline of FVRML and FX3D, there are seven modules, which can be replaced if necessary. These modules are *Parser*, *Compiler*, *ShapeGenerator*, *TextureGenerator*, *ColorGenerator*, and *PrimitiveGenerator*. For current Java-based and C++-based implementations, these modules can be replaced on-the-fly even after the extension has been compiled into the binary form. It is possible for third parties to create their own modules after the release of the implementations without exposing the source code of the extensions. Since the modules are responsible to parse, compile and interpret the function definitions and parameters, by replacing the proper modules, the FVRML and FX3D implementations can be extended to support new polygonisation algorithms and appropriate function representations.

When implementing the customised modules, C++-based implementation and Java-based implementation have different mechanisms and interfaces. This is so because Java is a platform-independent programming language with no low level access, while C++ is a platform-dependent programming language with low level access. However, the main idea of building a customised module would be pretty much consistent. The customised module is essentially

Chapter 6. Extensibility of FVRML/FX3D

an implementation of some specific methods. As C++ and Java both support object-oriented programming, the module is a concrete class implementing a pre-defined interface. For Java modules, the class name must be specially crafted in order to be recognised and used by the extension via reflection technology. For C++ modules, there is no requirement on the class name, however, the DLL file must have a special function exposed to the extension in order to create the module instance.

When authors want to use the customised modules, the VRML/X3D file has to be changed to specify the customised module name. In the FVRML/FX3D prototype design, the field *type* and *polygonizer* are designed for this purpose. When the field is missing, the default modules in the visualisation pipeline are used. If the field value is specified, the specific DLL or class file is loaded and executed in place of the default modules.

A new polygonisation algorithm can be introduced to the function-based extension by replacing the *PrimitiveGenerator* module. Optionally, if the mapping functions (as proposed in Chapter 3) of the shape, geometric texture and colour are desired to be changed, *ShapeGenerator*, *TextureGenerator*, and *ColorGenerator* can also be placed in the same Java class file or DLL as the *PrimitiveGenerator* module.

If a new function representation other than analytical formulae and function scripts is needed to be supported in FVRML and FX3D, two customised modules, *Parser* and *Compiler*, should be provided in order to replace the original module recognising analytical formulae and function scripts. Usually, these two modules are replaced concurrently, since it has little meaning to have multiple compilers for the same language syntax parsed by one parser.

When a new function operator (such as boolean operations with blending) needs to be added to the extension, the function can be built as analytical formulae or as a native library. If native libraries are chosen, they will replace the *TransformOperator* module in the visualisation pipeline, if the *type* field in the *FTransform* node is specified as the library name.

Chapter 6. Extensibility of FVRML/FX3D

As an example of customised modules, two new function representations have been developed using this approach. They are, Non-Uniform Rational B-Spline surfaces, and models created by the Interactive Function-based Shape Modelling project [133].

The NURBS extension is developed using the existing NURBS library called NURBS++. It contains both *Parser* and *Compiler* modules. The definition of the NURBS surface function employs a similar syntax defined by the *NurbsSurface* node in the X3D standard. Instead of placing the control points, knots, dimensions, and orders directly as fields in the *FGeometry* node, all the information is placed inside the *definition* field in the *FGeometry* node. The syntax can be recognised as a new function representation, which can only be used to define parametric NURBS functions. When the *type* field is set as “nurbs”, *Parser* and *Compiler* of the NURBS extension are loaded and executed in order to render NURBS surface together with function-defined geometric textures and colours. By introducing NURBS surface into FVRML and FX3D, it opens the possibility to use models produced by popular modelling tools, such as Maya or 3D Studio Max.

The extension used to visualise models created by the Interactive Function-based Shape Modelling project replaces all the replaceable modules in the visualisation pipeline. The modelling tool created by the project is a stand-alone application, which has the ability to model function-based objects by gradually modifying a base object. As a part of the project, a extension to FVRML/FX3D is required, since the software must export the modelled object into FVRML/FX3D files for web visualisation purposes. The objects are eventually represented by FRep functions. Since the project is focusing on fast function evaluation, in order to improve the efficiency, the function definition is actually stored as a binary array. However, in the prototype of function-based extension, no field can accept a binary array. To solve this problem, BASE64 encoding algorithm is used to convert the array into a string. Such string is then set as the value of the *definition* field. In order to visualise the model, both *type*

and *polygonizer* must be set to “tetracc”, since it changes function representation, polygonise algorithm, and all mapping functions.

6.3 Function script add-on

Module replacement is the ultimate way to extend the implementations, since it can change every aspect defined by the visualisation pipeline, except the time generation. However, *module replacement* is comparably complex and difficult to make, since developers have to understand the whole visualisation pipeline of the implementations, and implement all the functionalities required by the module. Sometimes, the developers may only want to add new customised functions to the function scripts. These customised functions might implement a commonly used algorithm, or provide a specific set of function-defined basic elements used to build function-based models. Though adding customised functions can be done by building customised *Parser* and *Compiler*, it would be time consuming and very inconvenient if the source code of the extension is not available. Therefore, an easier way to add new customised functions to the function scripts language must be provided to the users.

In the presented implementation, two easier approaches – custom function and native function – are provided in order to add new customised functions to the function scripts.

6.3.1 Custom function in function scripts

As proposed in Chapter 4, function scripts emulates the syntax of JavaScript. Therefore, as allowed in standard JavaScript, the developers are enabled to have their own functions written in the function scripts besides the required FRep or parametric functions. By using this feature, the authors who want to add new customised functions, may write them directly into the *definition* field in FVRML/FX3D using function scripts syntax. Because the function

Chapter 6. Extensibility of FVRML/FX3D

scripts have common flow control mechanisms as provided by other programming languages, it is possible to implement complex algorithms or procedural functions using the syntax.

When defining customised functions using function scripts, the developers only need to append the function definition using the correct syntax to the original function script source code. For example, if a customised function is required to evaluate the blobby objects [66] defined by $b e^{-a((x-x_0)^2+(y-y_0)^2+(z-z_0)^2)} - t$, the function script can be written as

```
function blobby(x,y,z,x0,y0,z0,a,b,t)
{
  return b*exp(-a*((x-x0)^2+(y-y0)^2+(z-z0)^2))-t;
}
```

By adding this customised function into the function definition, it is now possible to directly use the function inside the main functions. For example, the following code generates 6 bobbies with different parameters and positions using the introduced customised function.

```
function frep(x1,y1,z1)
{
  var x0 = new Array(6);
  var y0 = new Array(6);
  var z0 = new Array(6);
  var p = new Array(6);
  var b = new Array(6);
  x0[0]=-7;x0[1]=-6;x0[2]=-4;x0[3]=-2;x0[4]=0;x0[5]=7;
  y0[0]=-7;y0[1]=-4;y0[2]=-7;y0[3]=-4;y0[4]=-2;y0[5]=1;
  z0[0]=-7;z0[1]=-6;z0[2]=-4;z0[3]=-2;z0[4]=0;z0[5]=4;
  p[0]=0.7;p[1]=1;p[2]=1;p[3]=0.3;p[4]=1;p[5]=1;
  b[0]=2;b[1]=1;b[2]=1;b[3]=1;b[4]=1;b[5]=0.5;
  x=x1;y=y1;z=z1;fun=0;
  for (i=0;i<6;++i)
    fun=fun+blobby(x,y,z,x0[i],y0[i],z0[i],p[i],b[i],0.05);
  return fun;
}
```

Writing customised function using function scripts is easy to learn, since the author does not require programming language knowledge other than the function scripts syntax. It almost takes no effect for anyone who knows the function scripts syntax to write a series of

commonly used function-defined basic elements. In many cases, it solves the problem very fast, since it can be quickly written and tested.

Despite the advantages, this approach has some deficiencies. First, the custom functions in function scripts are not portable. The developers cannot have a library installed, so that each and every customised function can be used in all function scripts without copying and pasting codes. Instead, the same piece of code has to be copied several times. Second, though the extension implementation has a JIT execution engine for function scripts, it is still slower than the code generated by highly optimised native compilers. It is quite noticeable, especially when the function scripts are evaluated thousands of times. Third, the function scripts do not have the access to the system resources, such as the files on disk or web pages on the Internet. Finally, since there is no debugging facilities provided by the function scripts execution engine, the developers do not have an interactive way to find out bugs in the algorithms.

6.3.2 Native function in function scripts

It is always tempting for the developers to use customised functions in the same way as the pre-defined functions. Since the third party developers will not have the access to the source code of the function script of *Parser* and *Compiler*, it is infeasible to modify the existing modules in order to introduce new functions built by Java or C++. To solve this problem, the implemented function script *Compiler* module is designed to have add-ons containing customised functions. Such scheme is named as *native function*.

In order to introduce the function add-ons to the function scripts, three questions must be answered: how to recognise the customised function in the function scripts, how to verify the number of parameters defined by the customised function, and how to evaluate the customised function. To recognise the customised function in the function scripts, the name of the add-on must be the same name as the customised function. For example, if a customised function “trilinear” is developed by C++, the library name must be “trilinear.fun”. When

Chapter 6. Extensibility of FVRML/FX3D

the *Compiler* module finds a call to an unknown function (i.e. not pre-defined functions or custom functions inside the function script), it will try to load the proper library. The library contains two exposed methods, which gives out the number of parameters defined by the customised function, and the actual function which can evaluate the function value.

Since C++ and Java have the access to the system resources, such as files and web pages, it exposes more opportunities for the developers to create interesting function-based models. In this case, it usually requires strings as the arguments in the native customised functions. Nevertheless, the function script syntax only allows to have real numbers as arguments or variables. To provide the string support, the function source code is pre-processed automatically by converting a string into a series of floating-point numbers according to the length of the string and the ASCII code of the characters. This conversion is transparent to the authors writing function scripts. In the native function implementation, a reverse conversion can be done. Such standard method is provided as a part of the SDK.

6.4 Summary

The FVRML and FX3D implementations can be extended to fit into various applications. The extension can be done by two ways, either by replacing the modules in the visualisation pipeline, or by creating customised function script add-ons. Both these methods can be developed by third parties without the source code of the extension implementation. When replacing the modules, the developers can provide both additional function representations and polygonisation algorithms. Furthermore, the modules can define new mapping functions introduced by the modelling framework. However, the modules are comparably complex to be implemented. If the developers want to just add new functions to the function scripts, they can either write the functions by function scripts language, or develop a library containing the functions using Java or C++. The library would be recognised by the extension

Chapter 6. Extensibility of FVRML/FX3D

implementations, and executed as if they are parts of the pre-defined functions in the function scripts.

Chapter 7

Web-based collaborative modelling tool

In this chapter, a web-based modelling tool is presented. This modelling tool is an interactive collaborative application using FVRML and FX3D. The application can be used to create hybrid function-based models interactively. The created model can be saved for later modification, or exported to FVRML and FX3D source codes. In Section 7.1, the problems for all collaborative modelling systems are analysed. Existing solutions are also presented. To implement the modelling tool, an application framework based on FVRML and FX3D is proposed in Section 7.2. Section 7.3 discusses advantages and disadvantages of the existing multi-user virtual environment systems. The detailed implementation information is illustrated in Section 7.4. Finally, some platform-specific problems and the solutions are proposed in Section 7.5.

7.1 Collaborative modelling

Computer-aided design is often performed as a collaborative work and therefore it requires developing systems which allow distant participants to see, discuss, and change the same geometric model. Hence, computer-aided design systems working over networks were introduced. In such systems, *concurrency* and *synchronisation* are the two basic guidelines which must be complied with. *Concurrency* problems occur when different users try to manipulate

Chapter 7. Web-based collaborative modelling tool

one or more objects simultaneously. *Synchronisation* problems arise when changes to the objects are made by one of the users, and these changes have to propagate to other users to maintain consistency. When a multiple-user distributed system (i.e. cooperative system) is concurrent and synchronised, it is called collaborative. The integration of a networked CAD system and a communication environment is called collaborative modelling system, in which users can not only create and modify models interactively, but can communicate with each other in real-time. Hence, besides the concurrency and synchronisation, connection across different platforms is another issue in collaborative modelling system.

7.1.1 Concurrency

To deal with concurrency, a method of solving conflicts is needed. The most common method is called *floor control*. This scheme decides which user has a floor access, i.e. the right to perform manipulations on one or more objects. There are several ways of implementing the *floor control*, including *token passing*, *locking*, and some build-in mechanisms (e.g. single-threaded service).

Token passing is first seen in token ring networks. The idea of token passing can be described as following. At anytime, only one certain user holds the token, and only this token holder can perform manipulations on objects. If the token holder no longer needs the token, he passes it to another user. This kind of control can be seen in many actual applications. In [134], the FeatureM system employs this idea by implementing an audio channel where the users can discuss and decide who can modify the models. Similar loose implementation can be seen in webSpiff [135]. Instead of using an audio channel, traffic lights have been involved to show the current status of each part. The automated strict token passing implementation is exploited in TOBACO (i.e. Tool Based Cooperation) [136]. Token passing is also one of the concurrency handling methods in the Collaborative Solid Modelling system (CSM) [137].

Locking is a popular method for solving conflicts of critical sections in operating systems

Chapter 7. Web-based collaborative modelling tool

such as Windows and Linux. The basic idea of locking is similar to the token passing. At any time, only one user can successfully lock the model. If a user wants to modify the model, a request for locking the model is sent to the server (usually a session manager). If the locking request is successful, only this user can modify the model during the locking period. Once the user stops manipulation, the lock should be released immediately in order to avoid deadlock. In Advanced Realism CAD Environment (ARCADE) [138], each part of the model has one lock. This enables different users to modify different objects simultaneously, and hence boosts the efficiency of collaborative modelling. In some modelling systems, the locking mechanism is transparent to users. In this case, multiple users may execute modifications concurrently. However, the concurrent modifications are actually applied one by one using a server-side locking approach. For example, the MUG system [139] implemented a ticketing system, which ensures that no two modifications would be executed concurrently. In WSCAD system [140], the *Request Arbitrator* serves as the locking approach. It selects only one modification from the request queue at a time, and then sends the selected modification to modelling kernel. The kernel applies the modification and distributes such modification to the clients using Simple Object Access Protocol (SOAP).

Some build-in mechanisms can also be employed to solve the concurrency problem. For example, if the server is implemented in Java, the synchronised statement can be used to serialise the parallel requests, as mentioned in [135].

There are also mixed schemes. Thus DistView [141] is a toolkit that can be used for developing collaborative applications over WAN. In this system, locking is applied in order to solve the concurrency problem, meanwhile token passing is introduced to minimise the network delay when acquiring and releasing locks. Some of the systems provide more than one floor control policy such as 3dig [142], while only one policy can be activated at one time.

7.1.2 Synchronisation

In a collaborative environment, all clients share the same geometric model. Thus when one user changes the model, the changes must propagate to all the clients. Due to its nature, the synchronisation method is largely dependent on the information sent over network. In the most cases, two approaches, *model transmission* (i.e. sending the whole model) and *event transmission* (i.e. sending only the modifications), are employed. Due to recent rapid development of the broadband network and video streaming technology, certain *image transmission* (i.e. sending the whole rendered images or video streams) approaches can be used in small scale networks as well.

For *model transmission*, the synchronous problem solves automatically since for each modification the model is totally replaced by the new one. This kind of approach can be seen in CyberView [143], in which VRML is used when transferring data over the network by File Transfer Protocol (FTP). However, since the polygonal model, feature model, and rendered images are usually large in size, transferring such data interactively over the Internet is not possible at present. Therefore, CyberView is not a real collaborative modelling system according to its definition, but only an asynchronous modelling system combined with interactive viewing and commenting on the Web.

For *image transmission*, the synchronisation is solved in a very similar way compared to the *model transmission* approaches. In *image transmission*, the only model is kept on the central server, so there is no need to transmit and synchronise the models between all the clients. All modifications and view point changes are sent back to the server as messages. After the changes are applied at the server side, the new rendered image or video stream is distributed to the client. By using this approach, the clients do not require to have 3D rendering capabilities. However, due to the nature that only a few rendered images or video streams can be produced at interactive speed on one server, the clients either have to share

Chapter 7. Web-based collaborative modelling tool

the same view point, or have to use a large scale grid to support a number of users, such as approach used in [6].

For *event transmission*, the synchronisation problem complicates. First, in order to keep all clients consistent, the update events need to arrive at all the clients in the same order. Second, due to instability of the network, some inconsistency may occur which should be detected as soon as possible. If an inconsistency is found, a warning should be given, or even some recovery mechanism should be applied. For client-server networks, synchronisation can be monitored and controlled by the server. In ARCADE, TOBACO and all web-based systems, events are first sent to the server. At the server side, the events are serialised and then distributed to all the clients. This scheme ensures that the events are arrived at all client computers in the same order. For peer-to-peer networks, currently there is no ideal approach. In FeatureM, the inconsistencies can only be detected by comparing the log files after the modelling session. Thus, if any inconsistencies occur during the session, neither recovery nor warning is provided.

7.1.3 Connection

If the collaborative modelling system is specially built for a particular application, connection between the clients will not be a problem – TCP/UDP protocol can be used directly. In Collaborative Objects Coordination Architecture (COCA) [144], multicast technology as well as normal unicast, which are both based on UDP protocol, are employed in order to minimise network traffic. However, in many cases, different platforms are involved. Commonly, there are two solutions to this problem. For web-based systems, the common way is to exploit standard protocols based on HTTP such as Simple Object Access Protocol (SOAP), Common Object Request Broker Architecture (CORBA) as in FeatureM, TOBACO, 3dig, and NetFeature [145], and Java Remote Method Invocation (Java RMI) as in CSM and webSpiff. These protocols make the programmers and users transparent to the location and the type of the

server. Another straightforward way is to use wrappers specially designed for each platform. Such scheme is employed in ARCADE. Compared to the standard web-based protocols, this approach is usually faster and more efficient. However, it takes more time to develop and the extensibility is not good.

7.2 Interactive FVRML/FX3D application framework

Before actually implementing the collaborative modelling tool, an interactive FVRML/FX3D application development framework must be proposed. Such application framework should be used to develop any interactive collaborative applications based on FVRML and FX3D. We will define the application development framework as a hierarchy of three layers. They are: the *model providing layer*, the *modification layer*, and the *application layer*.

The lowest *model providing layer* is the layer of direct coding in FVRML/FX3D. Similar to other programming language, if one wants to make a meaningful model or application, lots of code has to be written. For novice users, it can be difficult to create a serious application only staying at this level. However, models defined by analytical formulae or function scripts can be easily modified on-the-fly in the virtual scene when the VRML/X3D JavaScript is used.

At the *modification layer*, one may interactively define any field of the FVRML/FX3D prototypes. It is proposed as a library which can be used to create interactive applications based on FVRML/FX3D. The general idea of this layer is to provide certain methods which can modify all FVRML/FX3D prototype fields interactively. Any field modification can be either a gradual modification (i.e. the modified value is based on the original field value), or a replacement modification (i.e. the original field value is replaced by a new value). In order to create collaborative applications, we should also include in this layer the ability of synchronising the field values either partially or as a whole. Thus, interactive application

Chapter 7. Web-based collaborative modelling tool

developers can directly employ the methods provided by the *modification layer*, and create the desired functionality in an easy way.

Based on the *modification layer*, which wraps the *model providing layer*, a versatile on-the-fly adjustable application can be created, that is performed at the *application layer*. The function-based models in the application can be even modified and synchronised on-the-fly without using any external software.

The proposed application framework may also work on other modelling methods, provided that the modelling approaches can identify the model modifications easily and can transfer the whole model or the modifications to the model efficiently over the network.

7.3 Development platform

Let's consider what platform could be chosen for developing a collaborative modelling tool. Since there is no native support for collaboration in VRML and X3D, a third party collaboration support must be obtained to develop the modelling tool. Three basic features must be available in the collaboration platform, which are text chatting, exchanging arbitrary data, and providing locking mechanism.

Ability to chat is an important feature of shared virtual environments. When one of the collaborating users types some text into a chat box, the text string is first sent to the server and then it is distributed to all the clients in the same session simultaneously. When several text messages are typed concurrently, the server is responsible to do the synchronisation properly so that all the chat scripts at the client-side will be the same for all users in the same session.

Sending arbitrary data may not be provided by a shared virtual platform, however this ability is crucial for the modelling tool. Though text chatting can be used for sending data over the network, it normally requires an interaction from the user (i.e. the user has to type the data text string into the chat text box). Therefore this is not a feasible solution for the

Chapter 7. Web-based collaborative modelling tool

modelling tool which will send data frequently, as each modification to the shape or appearance results in the model that has to be sent over the network. Thus, the chosen platform must provide a method which can send the data when a certain event occurs.

Besides the data transmission capabilities, certain locking mechanisms must also be available. Locking is a complicated feature provided by the server, and it usually cannot be accomplished by only client-based programming. As we already considered in Section 7.1, locking is the common practical way to resolve the concurrency issue in the collaborative modelling systems. Because of this, locking is also needed when choosing the development platform.

DeepMatrix [40] is a multi-user 3D environment that features chat, shared objects, and shared events. It is written in Java consisting of one server application and one client applet that works with different web3D plug-ins. DeepMatrix can use either existing VRML browser as the client via the Java EAI, or its own VRML client written in Java. The DeepMatrix's functionality includes text chatting facilities and shared events. However, the shared events in DeepMatrix do not provide any support of locking mechanism. This makes DeepMatrix not suitable for developing interactive applications based on the function-defined models.

blaxxun Communication Server can fulfil the three requirements of the modelling tool we proposed. It consists of two main components: the server and the client. The server acts not only as an HTTP server, which holds the VRML scene and HTML web pages, but also some special communication servers, which contain additional information of the shared virtual scene and exchange shared information across different clients. In its SDK, it provides a text chat box and shared events. By using the text chat box, user can type text messages and communicate with other users in the same session. By using shared events normally, it is possible to send any string to other users. By using shared events in the locking mode, the locking feature is provided by the blaxxun Communication Server automatically. The Server also can work together with Bitmanagement BS Contact VRML/X3D browser, however in

that case X3D files cannot be used as root scene files but only can be inlined from VRML files. Our experience of the Virtual Campus project [131] proves that such platform is sufficient for collaboration. Based on these considerations, we have chosen blaxxun Communication Server as the development platform.

7.4 Implementation of the interactive collaborative modelling tool

When designing the interactive web-based collaborative modelling tool, many aspects need to be taken care of. Since it is a collaborative modelling tool, the design of *concurrency*, *synchronisation*, and *connection* must be done. These three aspects are the common issues shared among all collaborative modelling tools as discussed in Section 7.1. In addition to these three common problems, the user interface and the functionality of the tool must also be clarified.

Before proposing any solution to the concurrency, synchronisation and connection problems, we must first understand the functionality of the tool.

According to the requirement, we know that the core part of the application will be applying the tool to the base shape. Depending on the tool selected, it may modify either the geometry or the appearance of the shape. Moreover, it is very common that in different sessions, users might want to use different sets of tools in order to fit the modelling purpose. Therefore, any pre-defined sets of tools will not be sufficient for advanced users. A method which can define customised tools must be provided. Also, within the same modelling session, different users may want to keep their own sets of customised and pre-defined tools as well as share their tools with other users when needed.

Besides the core functionality, we also consider other features which are very crucial to the real collaborative modelling tool. Such features include exporting the model to the

Chapter 7. Web-based collaborative modelling tool

VRML and X3D, saving the current model to a certain format for backing up purposes, loading previously saved shapes to continue modelling, and allowing users to join the current modelling session even after the modelling is started.

Taking the functionality and the platform into consideration, we propose the following solution to the three problems of collaboration.

The concurrency problem is comparably simple when designing this tool. We employ the common floor control mechanism to resolve the modification conflicts in the modelling tool. Since the blaxxun Communication Platform contains a central server, the most suitable approach of *floor control* would be the *locking* mechanism. The platform server can maintain a lock, which users would compete for. Once a user gets the lock, no other user can get the lock concurrently. Only the user acquiring lock can do modifications to the current shape. In order to achieve the fairness (i.e. each user would have the same priority to do the modifications), the lock is acquired and released automatically when a user wants to apply the modification to the shape. If the lock is already obtained by another user, the modification to the shape is ignored. This can avoid the situation that a user who acquired the editing lock then refuses to release it. Furthermore, such acquire-release process can be transparent to the user and can be done automatically by the modelling tool whenever the user wants to apply a modification to the current shape.

Since we are developing a web-based collaborative modelling tool, we can only do *model transmission* or *event transmission* in order to synchronise all clients within the same session.

At the first glance, it seems to be reasonable to use event transmission in order to synchronise all the clients. This sounds plausible, because the modelling process is done by incremental applying modifications to the current shape. Theoretically, if there is no network transmission error and all the users joined the session before the modelling process starts, each modification to the current shape can be propagated to all the clients as one event. How-

Chapter 7. Web-based collaborative modelling tool

ever, in reality the network transmission error occurs frequently if the data is transmitted over the Internet instead of a local area network (LAN). Furthermore, for collaborative modelling tool, it is also possible to have users joining in the middle of the modelling process, or loading a previously saved model. Such events must be detected and a recovery mechanism should be provided however it is quite challenging.

One of the advantages of using the hybrid function-based shape modelling framework is the small size of the models. When we use functions instead of polygons to define a shape, it usually results in much smaller file size. Thus, such model can be transmitted over the Internet in a more efficient way. Unlike the polygon-based models, which cannot make a real collaborative modelling system, the hybrid function-based modelling framework makes the model transmission for the interactive collaborative web-based modelling possible. Since the model transmission exchanges the whole modified model after each modification is applied to the current shape, it is impossible for any client to be continuously desynchronised after joining the modelling session. Moreover, whenever a new user is joining a halfway modelling session, the whole current model can be transmitted to the new client immediately. Thus, the new user can see the current model concurrently. In conclusion, model transmission scheme is the most suitable solution for our function-based collaborative modelling tool.

The solution to the connection problem is rather straightforward. Because we are using the blaxxun Communication Server, it already provides sufficient mechanisms to communicate between the clients, as well as between the clients and the blaxxun communication server, which is called shared events. Because of this, there is no need to create our own protocol or use other existing protocols for this tool. We take the full advantage of the existing protocol provided by the blaxxun Communication Server.

Since the modelling tool uses the hybrid function-based modelling framework, it is intuitive that the tools used to modify the current shape could be defined by functions as well. Because of the fact that the modelling tool uses a gradual modification process, the cus-

Chapter 7. Web-based collaborative modelling tool

tomised tools should not be very complex. Therefore, the most straightforward way to define a simple function-defined tool is by using analytical formulae. Since FVRML/FX3D already supports function definition, we simply employ its syntax.

Figure 7.1: GUI of the modelling tool

In the user interface of the pilot version of the collaborative shape modelling tool (Figure 7.1), there are three parts, which are the 3D modelling scene (top-left), the control panel (top-right) and the command/chatting area (bottom). This configuration is not a must and can

Chapter 7. Web-based collaborative modelling tool

be changed. The 3D modelling scene and the chatting area are shared among the users in the same session, while the control panel is user-specific and not shared. The user can type commands as well as chat in the command/chat pane. When the user wants to change the current tool, pre-defined tools and existing customised tools can be chosen from the drop-down menu in the control panel. The set of predefined tools can be changed by simple HTML editing of the control panel web page. If a new customised tool is desired during the modelling session, the analytical formulae of such tool can be entered in the control panel which will result in immediate introduction of the new tool. The user may also change the existing tools by selecting its name and changing its analytical formula. In the control panel, the user can also enter the colour including transparency of the current tool. There is also a text box in the control panel, which allows the user to load the saved models to the modelling scene.

To apply an interactive tool, the user clicks at any point on the current shape. The current tool will show at the clicked point, initially using the point as its local origin and the normal to the surface at this point as its principal axis. If the position or the orientation is not what the user wants, the user can to scale it, move it along its principal axis, displace along the normal to the shape, and rotate it arbitrarily using roll-pitch-yaw or free rotation. After placing the tool, the user selects the pre-defined operations to modify the geometry or the colour of the current shape. In Figure 7.1 three operations are used, which are removing of material, depositing of material and applying colour. Removing material subtracts the tool from the current shape (Figure 7.2). Depositing material unifies the tool with the current shape. Removing and depositing operations employ the Boolean operations provided by FVRML/FX3D directly. Colouring operation blends the colour assigned to the tool with the current colour of the shape according to the transparency assigned to the tool. The colour blending is performed only at the part of the current object that intersects with the tool. When the tool is not transparent, the colour of the tool overrides the colour of the current shape at the application area (Figure 7.3).

Chapter 7. Web-based collaborative modelling tool

Figure 7.2: Definition of the custom tool by its implicit formula and removing the material with this tool

Chapter 7. Web-based collaborative modelling tool

Figure 7.3: Colouring with different tools and blending values

Figure 7.4: Exporting the design to FVRML source code

Chapter 7. Web-based collaborative modelling tool

Since the modified shape might be complex, function scripts are generated instead of the analytical formulae. The operations mainly based on the mechanisms provided by function scripts and Boolean operations in FVRML/FX3D, so there is no complex internal data structure used. The whole model is represented by a single FVRML/FX3D object, while the tool is another FVRML/FX3D object. When the model is changed, a certain piece of function scripts is appended to the end of the existing model. If undo operation is involved, a part of the function scripts at its end is removed from the existing model. Thus, scripts defining the modified geometry or the modified colours are created on-the-fly and transmitted over the Internet to every client participating in the modelling session. The re-polygonisation is done immediately on each client machine after receiving the whole modified function scripts. By doing this, all users in the same session can see the changes concurrently.

When a new user joins the existing modelling session, his/her modelling tool will send out a message automatically, requesting the current model of the session. One existing modelling tool in the session, which holds a special lock, will answer to this request and will send the whole model to the new user. If there is no answer, the new user must be the first user of the modelling session, and thus some basic shape will be used as the default initial shape of the modelling session.

Besides the basic modelling ability, we also implemented a mechanism to save/load/export the modelled shape. When saving the modelled shape, the defining function scripts and parameters will be printed in the console pane of the browser. The user can copy and save it as a text file. When the user wants to resume the modelling process, by simply copying and pasting it into the model input pane of the modelling tool all users of the same session will be able to see and modify the previously saved shape. When exporting the shape, the corresponding FVRML or FX3D code will be sent to the console window of the browser (Figure 7.4). The user can copy and save it directly into a VRML or X3D file. Such saved

file can be viewed using any of the browsers which with the FVRML/FX3D plug-in installed. The user is also able to edit the saved/exported files with any text or VRML/X3D editors.

7.5 Platform-specific issues

There are several limitations of shared event support provided by blaxxun Communication Platform. First of all, it does not support transmitting CRLF in the string. Second, sometimes it receives parts of the transmitted string instead of the whole string at one time. Third and the most important, the string transmitted by the shared event has a maximum length limit. Any characters beyond the length limit would be ignored. These issues must have been resolved to develop the modelling tool.

Since the modelling tool generates editable function scripts instead of analytical functions, presence of CRLF is essential to make them readable. As we use the shared event to transmit only certain type of text, it is possible to replace the CRLF characters with special characters and restore them after receiving the message. In our modelling tool, we use ‘\’ character to represent CRLF characters in the transmitted messages.

In order to solve the partial transmission problem, we added a prefix and suffix to each message transmitted over the Internet. When a prefix is found in the received data, the modelling tool begins to accumulate the received data until the suffix can be found. By doing this, we can successfully receive the messages.

According to our experiment, the maximum length limit of the string in shared event is around 1,000 characters. Since the shared event can break a message into several chunks, we can also break a long message into several parts. Each part must be less than 1,000 characters to be transmitted over the Internet without any loss. We also used another lock to indicate if any-one is sending a message over the shared event, so that no parts of two different messages can be mixed together.

7.6 Summary

Though FVRML/FX3D is very versatile, it might be difficult for an inexperienced user to define something meaningful just by typing mathematical formulae. Besides this, even a user skilled in mathematics may have a problem with estimating 3D coordinates and parameters of geometric operations. Therefore, like with defining complex VRML and X3D scenes using interactive modelling tools, there is a necessity in such a tool for designing complex FVRML/FX3D objects which then can be included in standard VRML/X3D scenes. A collaborative option would allow the user to concurrently work on the design and discuss it with each other.

To implement such a modelling tool, an interactive FVRML/FX3D application framework consisting of three layers is proposed. Based on this application framework, the web-based collaborative modelling tool is implemented based on blaxxun Communication Platform. The modelling tool itself employs the functionalities provided by FVRML and FX3D. This tool is consistent with the paradigm of the hybrid function-based modelling. It lets the users introduce some basic objects, appearances and their modifications by typing analytical functions, interactively combine them into another function-defined object, and finally save the created object into FVRML/FX3D source code for further use.

Chapter 8

Examples of using FVRML/FX3D

This chapter illustrates using FVRML and FX3D. In Section 8.1, static models which mix the FRep functions and parametric functions are presented. Section 8.2 presents the ability to define time-dependent models using functions. Illustrations of extensions to FVRML and FX3D discussed in Chapter 6 are presented in Section 8.3. Section 8.4 presents some models for real applications created interactively using the web-based collaborative modelling tool. A workflow describing how to create hybrid function-based models interactively for real life applications is presented in Section 8.5. Finally, some discussions about making FVRML/FX3D models are presented in Section 8.6.

8.1 Static models

In FVRML and FX3D, the users are allowed to freely mix FRep functions and parametric functions to define geometries, geometric textures, and colours. As discussed in Chapter 3, there are four different possible ways to mix geometries and geometric textures and four different possible ways to mix geometries and colours. Let's illustrate them by simple analytical definitions of geometry and appearances.

Figure 8.1 shows the VRML code and the image of an FRep shape with FRep textures.

Chapter 8. Examples of using FVRML/FX3D

Figure 8.1: FRep shape and FRep texture

The value of the FRep texture function is interpreted as an amount of the displacement along the surface normal. In this example, the original FRep shape “heart” is defined as:

$$(0.1z^3 + x^2)y^3 - (2z^2 + x^2 + y^2 - 1)^3 \geq 0$$

and the FRep texture function is defined as a noise function based on the Gardner series:

$$0.02(\sin(10\pi x) \sin(10\pi y) + \sin(10\pi x) \sin(10\pi z) + \sin(10\pi y) \sin(10\pi z))$$

Figure 8.2 shows the X3D code and the image of an FRep shape with parametric textures.

Chapter 8. Examples of using FVRML/FX3D

The value of the parametric texture function is interpreted as the absolute displacement of the original shape at every point. In this example, the original FRep shape is “blending of three tori”, and the parametric texture is defined as a normalised noise function:

$$\begin{aligned} x &= \frac{0.02 \sin(10\pi x) \sin(10\pi y) \sin(10\pi z) u}{\sqrt{u^2 + v^2 + w^2}} \\ y &= \frac{0.02 \sin(10\pi x) \sin(10\pi y) \sin(10\pi z) v}{\sqrt{u^2 + v^2 + w^2}} \\ z &= \frac{0.02 \sin(10\pi x) \sin(10\pi y) \sin(10\pi z) w}{\sqrt{u^2 + v^2 + w^2}} \end{aligned}$$

Figure 8.3 shows the X3D code and the image of a parametric shape with FRep textures. In this example, the original parametric shape “star” is defined as:

$$\begin{aligned} x &= (1 - 0.3 \sin(5\pi v)) \cos(\pi v) (4 + \cos(\pi u)) \\ y &= (1 - 0.3 \sin(5\pi v)) \sin(\pi v) (4 + \cos(\pi u)) \\ z &= \sin(\pi u) \end{aligned}$$

and the same FRep texture function used in Figure 8.1 is employed.

Figure 8.4 shows the VRML code and the image of a parametric shape with parametric textures. In this example, the original parametric shape is a unit sphere, and the parametric texture is defined as:

$$x = 0.1 \sin(4\pi u), y = 0.1 \sin(4\pi v), z = -0.1 \sin(4\pi w)$$

Figure 8.5 shows the X3D code and the image of an FRep shape with FRep colours. The value of the FRep colour function is interpreted as a colour by looking up the corresponding value linearly interpolated according to *patternColor* and *patternKey* fields. The FRep colour of this object can be thought of as a 3D density field in the same geometric coordinate space. The function value is then mapped to the actual colour according to the designed colour map. In this example, the FRep shape of the “watermelon” object is defined as a simple sphere. For the colour of the “watermelon”, the uneven shape defined in the colour space can be used to make green patterns. This is achieved by mapping the colour function value from 0.98 to 1 to red, green and blue values ranging from [0,1,0] to [0,0.2,0]. The colour inside the “watermelon” is uniform red.

Chapter 8. Examples of using FVRML/FX3D

```

<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN"
"http://www.web3d.org/specifications/x3d-3.0.dtd">
<X3D version="3.0" profile="Immersive">
  <Scene>
    <!--External Prototype Declaration-->
    <ProtoInstance name="FShape">
      <fieldValue name="appearance">
        <ProtoInstance name="FAppearance">
          <fieldValue name="material"><Material diffuseColor="0 1 1" /></fieldValue>
          <fieldValue name="texture3D"><ProtoInstance name="FTexture3D">
            <fieldValue name="definition"
              value="
function parametric_x(u,v,w,t){return 0.2*sin(10*pi*u)*sin(10*pi*v)*sin(10*pi*w)*u/sqrt(u*u+v*v+w*w);}
function parametric_y(u,v,w,t){return 0.2*sin(10*pi*u)*sin(10*pi*v)*sin(10*pi*w)*v/sqrt(u*u+v*v+w*w);}
function parametric_z(u,v,w,t){return 0.2*sin(10*pi*u)*sin(10*pi*v)*sin(10*pi*w)*w/sqrt(u*u+v*v+w*w);}"/>
            <fieldValue name="type" value="displacement" />
          </ProtoInstance></fieldValue>
        </ProtoInstance>
      </fieldValue>
      <fieldValue name="geometry">
        <ProtoInstance name="FGeometry">
          <fieldValue name="resolution" value="100 100 100" />
          <fieldValue name="definition" value="
exp(-(x^4+2*x*x*(y*y+z*z-3.4)+y^4+2*y*y*(z*z+3.08)
+z^4-2*z*z*3.4+9.4864))
+exp(-(x^4+2*x*x*(z*z+y*y-3.4)+z^4+2*z*z*(y*y+3.08)
+y^4-2*y*y*3.4+9.4864))
+exp(-(y^4+2*y*y*(x*x+z*z-3.4)+x^4+2*x*x*(z*z+3.08)
+z^4-2*z*z*3.4+9.4864))-0.7" />
        </ProtoInstance>
      </fieldValue>
      <fieldValue name="bboxCenter" value="0 0 0" />
      <fieldValue name="bboxSize" value="9 9 9" />
    </ProtoInstance>
  </Scene>
</X3D>

```

Standard colour

Parametric texture

FRep geometry

Figure 8.2: FRep shape and parametric texture

Chapter 8. Examples of using FVRML/FX3D

```

<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN"
  "http://www.web3d.org/specifications/x3d-3.0.dtd">
<X3D version="3.0" profile="Immersive">
  <Scene>
    <!--External Prototype Declaration-->
    <ProtoInstance name="FShape">
      <fieldValue name="bboxCenter" value="0 0 0" />
      <fieldValue name="bboxSize" value="3 3 3" />
      <fieldValue name="appearance">
        <ProtoInstance name="FAppearance">
          <fieldValue name="material"><ProtoInstance name="FMaterial">
            <fieldValue name="diffuseColor" value="r=1;g=1-fabs(sin(u));b=0;" />
          </ProtoInstance></fieldValue>
          <fieldValue name="texture3D"><ProtoInstance name="FTexture3D">
            <fieldValue name="definition"
              value="0.1*(sin(2*pi*x)*sin(2*pi*y)
                +sin(2*pi*x)*sin(2*pi*z)
                +sin(2*pi*y)*sin(2*pi*z))" />
            <fieldValue name="type" value="displacement" />
          </ProtoInstance></fieldValue>
        </ProtoInstance>
      </fieldValue>
      <fieldValue name="geometry">
        <ProtoInstance name="FGeometry">
          <fieldValue name="resolution" value="100 100 100" />
          <fieldValue name="parameters" value="-1 1 -1 1 -1 1" />
          <fieldValue name="definition" value="
            x=(1-0.3*sin(v*5*pi))*cos(v*pi)*(4+cos(u*pi));
            y=(1-0.3*sin(v*5*pi))*sin(v*pi)*(4+cos(u*pi));
            z=sin(u*pi);" />
        </ProtoInstance>
      </fieldValue>
    </ProtoInstance>
  </Scene>
</X3D>

```

Diagram annotations:

- Parametric colour**: Points to the `<fieldValue name="diffuseColor" value="r=1;g=1-fabs(sin(u));b=0;" />` line.
- FRep texture**: Points to the `<fieldValue name="definition" value="0.1*(sin(2*pi*x)*sin(2*pi*y) + sin(2*pi*x)*sin(2*pi*z) + sin(2*pi*y)*sin(2*pi*z))" />` block.
- Parametric geometry**: Points to the `<fieldValue name="definition" value="x=(1-0.3*sin(v*5*pi))*cos(v*pi)*(4+cos(u*pi)); y=(1-0.3*sin(v*5*pi))*sin(v*pi)*(4+cos(u*pi)); z=sin(u*pi);" />` block.

Figure 8.3: Parametric shape and FRep texture

Chapter 8. Examples of using FVRML/FX3D

```

FShape {
  appearance FAppearance {
    material FMaterial {
      diffuseColor "r=1;g=1-fabs(sin(u));b=0;"
    }
    texture3D FTexture3D {
      definition "x=0.1*sin(4*pi*u);
        y=0.1*cos(4*pi*v);z=-0.1*cos(4*pi*u);"
      type "displacement"
    }
  }
  geometry FGeometry {
    definition "x=cos(v*pi);y=cos(u*pi)*sin(v*pi);
      z=sin(u*pi)*sin(v*pi);"
    resolution [100 100 100]
    parameters [0.00001 1.99999 0.000001 0.99999]
    bboxCenter 0 0 0
    bboxSize 3 3 3
  }
}

```

Parametric colour

Parametric texture

Parametric geometry

Figure 8.4: Parametric shape and parametric texture

Chapter 8. Examples of using FVRML/FX3D

```

<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN"
"http://www.web3d.org/specifications/x3d-3.0.dtd">
<X3D version="3.0" profile="Immersive">
  <Scene>
    <!--External Prototype Declaration-->
    <ProtoInstance name="FShape">
      <fieldValue name="appearance">
        <ProtoInstance name="FAppearance">
          <fieldValue name="material"><ProtoInstance name="FMaterial">
            <fieldValue name="diffuseColor" value="
sqrt(x*x+y*y+z*z)+0.01*(sin(12*atan2(x,z+0.04*sin(y*25)))-0.7)" />
            <fieldValue name="patternKey" value="0 0.8 0.96 0.98 1" />
            <fieldValue name="patternColor" value="0.4 0 0,1 0 0,1 1 1,0 1 0,0 0.2 0" />
          </ProtoInstance></fieldValue>
        </ProtoInstance>
      </fieldValue>
      <fieldValue name="geometry">
        <ProtoInstance name="FGeometry">
          <fieldValue name="resolution" value="100 100 100" />
          <fieldValue name="parameters" value="0" />
          <fieldValue name="definition" value="
function frep(x,y,z,t) {
  return (1-x*x-y*y-z*z)&((-x+z)&y&z);
}" />
        </ProtoInstance>
      </fieldValue>
      <fieldValue name="bboxCenter" value="0 0 0" />
      <fieldValue name="bboxSize" value="2 2 2" />
    </ProtoInstance>
  </Scene>
</X3D>

```

FRep colour

FRep geometry

Figure 8.5: FRep object with FRep colour

Figure 8.6 shows the VRML code and the image of an FRep shape with parametric colours. The value of the parametric colour function is interpreted as red, green and blue values of a colour directly. In this example, the FRep shape of the “broken heart” is defined as the original shape in Figure 8.1 intersected with two planes. The colour of the shape changes from the centre to the surface.

Figure 8.7 shows the VRML code, the file on the web, and the image of a parametric shape with FRep colours. In this example, the parametric shape function is defined in a separate file placed on the web. By doing this, only the URL of the file on the web has to be placed in the *definition* field. FVRML and FX3D are able to automatically detect the URL and to retrieve the file as the actual source code.

Figure 8.8 shows the X3D code and the image of a parametric shape with parametric colours. In this example, the parametric shape is defined as a distorted sphere, while the parametric colour function assigns each dent and bump with different colours.

Besides defining geometries, geometric textures, and colours, operations of functions can also be defined using *FTransform* node. For example, the union operation with blending can be analytically defined as $f_1 + f_2 + \sqrt{f_1^2 + f_2^2} + \frac{1}{1+f_1^2+f_2^2}$. Therefore, applying this operation on two cylinders can be defined as shown in Figure 8.9.

8.2 Time-dependent models

Time-dependent functions can be employed in FVRML and FX3D to define sophisticated animations. The animations can be geometry animation, geometric texture animation, and colour animation. Any combination of these three types of animations can be done by FVRML and FX3D using FRep and/or parametric functions. This will be illustrated by models with dynamic changes defined by simple time-dependent analytical definitions of geometry and appearances.

Chapter 8. Examples of using FVRML/FX3D

```

FShape {
  appearance FAppearance {
    material FMaterial {
      diffuseColor "
        function parametric_r(u,v,w,t)
        {return 1-(((2*w^2+u^2+v^2-1)^3
        -(0.1*w^2+u^2)*v^3))^2;}
        function parametric_g(u,v,w,t)
        {return sqrt(fabs(((2*w^2+u^2+v^2-1)^3
        -(0.1*w^2+u^2)*v^3)))};
        function parametric_b(u,v,w,t) {return 0;}
        specularColor "r=1;g=1;b=1;"
        shininess "0.7"
      }
    }
  }
  geometry FGeometry {
    definition "(-(2*z^2+x^2+y^2-1)^3
    -(0.1*z^2+x^2)*y^3))&(-(x&z))"
    type "analytical"
    bboxCenter 0 0 0
    bboxSize 3 3 3
    resolution [100 100 100]
    parameters 0
  }
}

```

Parametric colour

FRep geometry

Figure 8.6: FRep object with parametric colour

Chapter 8. Examples of using FVRML/FX3D

```

FShape {
  appearance FAppearance {
    material FMaterial {
      diffuseColor "function frep(x,y,z,t) {return sqrt(x*x+y*y+z*z);}"
      patternColor [0 1 0 1 0 0]
      patternKey [0.7 1.3]
    }
  }
  geometry FGeometry {
    definition "http://www.ntu.edu.sg/home/assourin/fvrml/shape.txt"
    resolution [200 200]
    parameters [-1 1 -1 1]
  }
}

```

Annotations:

- FRep colour**: Points to the `diffuseColor` property.
- Parametric geometry defined by an URL**: Points to the `definition` property.

Content of `http://www.ntu.edu.sg/home/assourin/fvrml/shape.txt`:

```

x=(0.3*sin(10*(u*pi+v*pi))+1)*cos(u*pi);
y=(0.3*sin(10*(u*pi+v*pi))+1)*sin(u*pi)*cos(v*pi);
z=(0.3*sin(10*(u*pi+v*pi))+1)*sin(u*pi)*sin(v*pi);"

```

Figure 8.7: Parametric object with FRep colour

Chapter 8. Examples of using FVRML/FX3D

```

<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN"
  "http://www.web3d.org/specifications/x3d-3.0.dtd">
<X3D version="3.0" profile="Immersive">
  <Scene>
    <!--External Prototype Declaration-->
    <ProtolInstance name="FShape">
      <fieldValue name="appearance">
        <ProtolInstance name="FAppearance">
          <fieldValue name="material"><ProtolInstance name="FMaterial">
            <fieldValue name="diffuseColor" value="r=0.3*sin(10*u*pi)*sin(10*v*pi)+0.5;
            g=0.3*sin(10*v*pi)+0.5;b=0.3*sin(10*u*pi)+0.5;" />
          </ProtolInstance></fieldValue>
        </ProtolInstance>
      </fieldValue>
      <fieldValue name="geometry">
        <ProtolInstance name="FGeometry">
          <fieldValue name="resolution" value="200 200" />
          <fieldValue name="parameters" value="0 2 0 2" />
          <fieldValue name="definition" value="
            x=(0.05*sin(10*u*pi)*sin(10*v*pi)+1)*cos(u*pi)*((sin(0.5*v*pi))^3);
            y=(0.05*sin(10*u*pi)*sin(10*v*pi)+1)*sin(u*pi)*((sin(0.5*v*pi))^3);
            z=(0.05*sin(10*u*pi)*sin(10*v*pi)+1)*cos(0.5*v*pi);" />
        </ProtolInstance>
      </fieldValue>
    </ProtolInstance>
  </Scene>
</X3D>

```

Parametric colour

Parametric geometry

Figure 8.8: Parametric object with parametric colour

Chapter 8. Examples of using FVRML/FX3D

Figure 8.9: Union of two cylinders with blending

Chapter 8. Examples of using FVRML/FX3D

```

FShape {
  cycleInterval 5
  loop true
  appearance FAppearance {
    material FMaterial {
      diffuseColor "sin(sqrt(x*x+y*y+z*z)*pi)"
      patternColor 0 1 0 1 0 0
      patternKey -1 1
    }
  }
  geometry FGeometry {
    resolution [100 30]
    parameters [0 5 0 2 -1 1]
    timeSpan 0.01 1
    definition "
      x=(1-0.1*cos(v*pi))*cos(u*pi*t);
      z=(1-0.1*cos(v*pi))*sin(u*pi*t);
      y=0.1*(sin(v*pi)+u*t*pi/1.7-10);"
  }
}

```

Annotations in the diagram:

- 5 seconds per cycle loop animation** points to `cycleInterval 5`.
- FRep static colour** points to `diffuseColor "sin(sqrt(x*x+y*y+z*z)*pi)"`.
- Parametric time-dependent geometry** points to the `definition` block.

Figure 8.10: Geometry animation

Figure 8.10 illustrates the ability to define time-dependent geometries using FVRML. In this example, the time-dependent geometry is defined as sweeping a circle along a 3D helical curve. It is done by parametric formulae where one of the parameters is time. In this example, there is no geometric texture, and the colour defined by FRep function does not change through time.

Figure 8.11 illustrates the ability to define time-dependent geometric textures using FX3D. In this example, both the geometry and the colour are static. The geometry is defined using

Chapter 8. Examples of using FVRML/FX3D

parametric function scripts and the colour is defined by parametric formulae. The geometric texture is a time-dependent solid noise. The noise displaces the surface of the object thus achieving the bumpy appearance. The swing animation between a smooth and a bumpy surface is defined.

Figure 8.12 illustrates the ability to define time-dependent colours using FVRML. In this example, the geometry is static and defined by parametric function scripts. There is no geometric texture defined. The colour permutation of the “knot” object is defined by the time-dependent explicit formula, the *patternColor* field, and the *patternKey* field.

Figure 8.13 illustrates the ability to define time-dependent geometries, time-dependent colours and time-dependent geometric textures concurrently using FX3D. In this example, geometry, colours and geometric textures are all defined by time-dependent functions with different time spans. The animated scene shows rippling waves which are distorted by the solid noise similar to the one used in Figure 8.11. The colour of the waves changes through time.

8.3 Other function-defined models

The proposed modelling framework as well as the extension allow any proprietary function-based representations to be used and rendered. According to the discussion in Chapter 6, this can be done either by replacing the proper modules, or creating add-ons to the function scripts.

When replacing the modules in the visualisation pipeline, it is possible to import FRep shapes modelled with the interactive function-based shape modelling system introduced in [84]. This is so because the modelling system requires a new polygonisation and function evaluation algorithm. By saving it in an encoded ASCII data format, the FRep models can be

Chapter 8. Examples of using FVRML/FX3D

```

<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN"
"http://www.web3d.org/specifications/x3d-3.0.dtd">
<X3D version="3.0" profile="Immersive">
  <Scene>
    <!--External Prototype Declaration-->
    <ProtoInstance name="FShape">
      <fieldValue name="cycleInterval" value="5" /><fieldValue name="loop" value="true" />
      <fieldValue name="appearance"><ProtoInstance name="FAppearance">
        <fieldValue name="material"><ProtoInstance name="FMaterial">
          <fieldValue name="diffuseColor" value="r=sin(u*pi*2);g=fabs(sin(u));b=0;" />
        </ProtoInstance></fieldValue>
        <fieldValue name="texture3D"><ProtoInstance name="FTexture3D">
          <fieldValue name="timeSpan" value="0 0.4" />
          <fieldValue name="definition"
            value="fabs(t-0.2)*(sin(2*pi*x)*sin(2*pi*y)+sin(2*pi*x)*sin(2*pi*z)
              +sin(2*pi*y)*sin(2*pi*z))" />
          <fieldValue name="type" value="displacement" />
        </ProtoInstance></fieldValue></ProtoInstance></fieldValue>
      <fieldValue name="geometry"><ProtoInstance name="FGeometry">
        <fieldValue name="resolution" value="50 50" />
        <fieldValue name="parameters" value="0 6.28 0 6.28 -1 1" />
        <fieldValue name="definition" value="
          function parametric_x(u,v,w,t)
          {return sin(u)*(7+cos(u/3-2*v)+2*cos(u/3+v));}
          function parametric_y(u,v,w,t)
          {return cos(u)*(7+cos(u/3-2*v)+2*cos(u/3+v));}
          function parametric_z(u,v,w,t)
          {return sin(u/3-2*v)+2*sin(u/3+v);}  " />
      </ProtoInstance></fieldValue></ProtoInstance></Scene></X3D>

```

5 seconds per cycle
loop animation

Parametric static colour

FRep time-dependent
texture

Parametric static geometry

Figure 8.11: Geometric texture animation

Chapter 8. Examples of using FVRML/FX3D

```

FShape {
  cycleInterval 5
  loop true
  appearance FAppearance {
    material FMaterial {
      @diffuseColor "sin(sqrt(x*x+y*y+z*z)*pi+t)"
      timeSpan -3.1415926 3.1415926
      patternKey -1 1
    }
  }
  geometry FGeometry {
    resolution [200 30]
    parameters [-1 1 -1 1 -1 1]
    definition "
      function parametric_x(u,v,w,t) {u=(u+1)*pi*2;v=(v+1)*pi;
      return .1*cos(v)*cos(u)+0.5*cos(u)*(1+.4*cos(1.5*u));}
      function parametric_y(u,v,w,t) {u=(u+1)*pi*2;v=(v+1)*pi;
      return .1*sin(v)+.4*sin(1.5*u)-1;}
      function parametric_z(u,v,w,t) {u=(u+1)*pi*2;v=(v+1)*pi;
      return .1*cos(v)*sin(u)+0.5*sin(u)*(1+.4*cos(1.5*u))-1;}
    "
  }
}

```

Annotations:

- 5 seconds per cycle
loop animation (points to `cycleInterval 5` and `loop true`)
- FRep time-dependent colour (points to `@diffuseColor "sin(sqrt(x*x+y*y+z*z)*pi+t)"`)
- Parametric time-dependent geometry (points to the `definition` block)

Figure 8.12: Colour animation

Chapter 8. Examples of using FVRML/FX3D

```

<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN"
"http://www.web3d.org/specifications/x3d-3.0.dtd">
<X3D version="3.0" profile="Immersive">
  <Scene>
    <!--External Prototype Declaration-->
    <ProtoInstance name="FShape">
      <fieldValue name="cycleInterval" value="5" /><fieldValue name="loop" value="true" />
      <fieldValue name="appearance"><ProtoInstance name="FAppearance">
        <fieldValue name="material"><ProtoInstance name="FMaterial">
          <fieldValue name="diffuseColor" value=
            "r=cos(v*pi*2+t*pi*6);g=fabs(sin(u));b=sin(v*pi*2+t*pi*4)/2;" />
          </ProtoInstance></fieldValue>
          <fieldValue name="texture3D"><ProtoInstance name="FTexture3D">
            <fieldValue name="timeSpan" value="0 0.4" />
            <fieldValue name="definition" value="fabs(t-0.2)*(sin(2*pi*x)
              *sin(2*pi*y)+sin(2*pi*x)*sin(2*pi*z)+sin(2*pi*y)*sin(2*pi*z))" />
            <fieldValue name="type" value="displacement" />
            </ProtoInstance></fieldValue></ProtoInstance></fieldValue>
            <fieldValue name="geometry"><ProtoInstance name="FGeometry">
              <fieldValue name="resolution" value="50 100" />
              <fieldValue name="parameters" value="0 6.28 0 6.28 -1 1" />
              <fieldValue name="definition" value="
                function parametric_x(u,v,w,t) {return v*5*cos(u);}
                function parametric_y(u,v,w,t) {return v*5*sin(u);}
                function parametric_z(u,v,w,t) {return cos(v*5+t*2*pi);}
              " />
            </ProtoInstance></fieldValue></ProtoInstance></Scene></X3D>

```

5 seconds per cycle
loop animation

Parametric time-dependent
colour

FRep time-dependent
texture

Parametric time-dependent
geometry

Figure 8.13: Combined geometry, colour and geometric texture animation

Chapter 8. Examples of using FVRML/FX3D

(a) FRep "Duck"

(b) FRep "Bird"

(c) FRep "Swan" with an interactively painted colour

(d) The FRep "Swan" coloured with parametric functions

Figure 8.14: Shapes created with an interactive function-based shape modelling programme and imported into VRML/X3D for viewing and further modification with the function-based extension

delivered and displayed by VRML/X3D browsers. Several examples of such FRep shapes as well as FRep geometry coloured by analytical formulae in VRML are given in Figure 8.14.

The add-on can be done via either implementing the customised function in function scripts, or in a native library.

In Figure 8.15, a customised blobby function in function script presented in Section 6.3.1 is used for modelling a body. It is purely written in function script without using any additional modules or native customised functions. The source code of this object can be seen in Figure 8.15(a). It contains 26 bobbies with different locations and sizes. The base blobby function is defined as a customised function 'blobby' in the function script. It takes the centre of the blobby, the size of the blobby and other parameters. Figure 8.15(b) shows these

Chapter 8. Examples of using FVRML/FX3D

blobbies as if they are spheres. Standard union operation is applied to these spheres, thus the model is not smooth. Figure 8.15(c) shows the smooth blobby model. The blobby functions are unified using the addition operation, which is common to smooth blobby models.

Figure 8.16 is an example which uses a native library containing a customised function. The customised function trilinearly interpolates a CT volume data, and returns the interpolated value. When a proper threshold is chosen, the trilinear interpolation function can be regarded as an FRep function representing an object. The surface of the object can thus be reconstructed by the polygonisation algorithm. Since native customised functions are a part of the function scripts, it can either be used along or mixed with other function script features. When mixing other function script features together with the native trilinear interpolation function, it is possible to modify the geometry and the colour of the reconstructed object by applying set-theoretic operations (such as union, intersection, difference, and blending) to the CT volume data.

8.4 Creating models interactively

In Chapter 7, a web-based collaborative modelling tool is built based on FVRML and FX3D. In Figure 8.17, we give an illustration of the example of shape modelling with the developed tool. First, we define the basic shape by an implicit function which defines a distorted sphere with another smaller sphere subtracted from it. This is done by typing the following command:

```
shape: (0.8^2-x^2-y^2-z^2+0.03*(sin(12*atan2(x,z+0.04*sin(y*25)))-0.7))
&(-(0.7^2-x^2-y^2-z^2))
```

The colour of this shape is then defined by an explicit distance function

$$g = \sqrt{x^2 + y^2 + z^2}$$

Chapter 8. Examples of using FVRML/FX3D

```

DEF shape FShape {
  appearance Appearance {
    material Material { diffuseColor .8 .4 0.5 }
  }
  geometry DEF geo FGeometry {
    bboxCenter 0 1 0 bboxSize 2 2 2 resolution [100 100 100]
    definition "
function blobby(x,y,z,x0,y0,z0,r,b) {
rr = sqrt((x-x0)^2+(y-y0)^2+(z-z0)^2);
return exp(b*(r-rr));}
function frep(x1,y1,z1){
x=x1;y=y1;z=z1;fun=-1;b=60;
fun=fun+blobby(x,y,z,0,0.6,0,0.15,b); fun=fun+blobby(x,(y-0.7)/2+0.7,z,0,0.7,0,0.15,b);
fun=fun+blobby(x,y,z,0,1,0,0.05,b); fun=fun+blobby(x,y,z,0,1.15,0,0.15,b);
fun=fun+blobby((x-0.15)/2+0.15,y,z,0.15,0.95,0,0.05,b); fun=fun+blobby(x,y,z,0.22,0.9,0,0.05,b);
fun=fun+blobby(x,(y-0.85)/2+0.85,z,0.25,0.85,0,0.05,b); fun=fun+blobby(x,y,z,0.25,0.75,0,0.05,b);
fun=fun+blobby(x,(y-0.65)/2+0.65,z,0.25,0.65,0,0.05,b);
fun=fun+blobby((x+0.15)/2-0.15,y,z,-0.15,0.95,0,0.05,b);
fun=fun+blobby(x,y,z,-0.22,0.9,0,0.05,b); fun=fun+blobby(x,(y-0.85)/2+0.85,z,-0.25,0.85,0,0.05,b);
fun=fun+blobby(x,y,z,-0.25,0.75,0,0.05,b); fun=fun+blobby(x,(y-0.65)/2+0.65,z,-0.25,0.65,0,0.05,b);
fun=fun+blobby(x,y,z,0.1,0.6,0,0.05,b); fun=fun+blobby(x,(y-0.5)/2+0.5,z,0.1,0.5,0,0.06,b);
fun=fun+blobby(x,y,z,0.1,0.35,0,0.05,b); fun=fun+blobby(x,(y-0.25)/2+0.25,z,0.1,0.25,0,0.06,b);
fun=fun+blobby(x,y,z,0.1,0.1,0,0.05,b); fun=fun+blobby(x,y,(z-0.05)/2+0.05,0.1,0.05,0.05,0.05,b);
fun=fun+blobby(x,y,z,-0.1,0.6,0,0.05,b); fun=fun+blobby(x,(y-0.5)/2+0.5,z,-0.1,0.5,0,0.06,b);
fun=fun+blobby(x,y,z,-0.1,0.35,0,0.05,b); fun=fun+blobby(x,(y-0.25)/2+0.25,z,-0.1,0.25,0,0.06,b);
fun=fun+blobby(x,y,z,-0.1,0.1,0,0.05,b); fun=fun+blobby(x,y,(z-0.05)/2+0.05,-0.1,0.05,0.05,0.05,b);
return fun;}
} }

```

(a) VRML source code

(b) Unblended model

(c) Blended model

Figure 8.15: Blobbies using custom function in function script

Figure 8.16: Bunny using trilinear interpolation of volume data with modifications and colouring

which values are linearly mapped to the colours, based on the key-values defined by pattern keys and colours in patternkey/patterncolor commands. This is achieved by typing commands:

patternkey: 0, 0.7, 0.74, 0.76, 0.78

patterncolor: 0.4, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0.8, 0

*color: sqrt(x*x+y*y+z*z)*

It results in a colour which changes from yellow-green colours on the surface, to white-pink immediately beneath it, and finally to grades of red colour inside the shape.

Next, we select a pre-defined tool “box”, locate it on the surface of the shape, interactively change its size and orientation, and finally cut away the material with it to make “eyes”. Then, we change the tool’s formula to make a cutter to carve piece by piece a “mouth”. Finally, we change the tool’s formula one more time to make a cutter for carving away a “nose”. The final design can be saved either into FVRML or FX3D source code.

When modules are placed, the models recognised by the new modules can still be inter-

Chapter 8. Examples of using FVRML/FX3D

Figure 8.17: Example of web-based interactive shape modelling

Chapter 8. Examples of using FVRML/FX3D

Figure 8.18: Interactive make-up of the NURBS head

actively coloured via the web-based collaborative modelling tool. Figure 8.18 shows such example. In this case, the base object – the human head – is defined by one NURBS surface using the customised NURBS parser and compiler modules in FVRML/FX3D. When the base object is imported into the modelling session, all participants can see the same human head concurrently. Though the modelling tool cannot modify the geometry of the NURBS human head, because it is not defined by function scripts and FRep functions, the users can still colour the human head using various pre-defined and customised tools. Colour blending is also available here. When the modellers are satisfied with the colour, the modified model can be saved and exported as if it were a model defined purely by the modelling tool.

8.5 Working with hybrid function-based models

When working with any modelling technique, no matter function- or polygon-based, it is usually difficult to create a geometry directly, especially when modelling objects in real life. Therefore, for each modelling technique, there are many tools which can be used to help creating 3D models. By combining the tools in a proper order, a workflow can be proposed. The workflow can be a guideline for any user who wants to create 3D models using the modelling technique. In this thesis, several tools have been introduced, including function scripts

Chapter 8. Examples of using FVRML/FX3D

Figure 8.19: Proposed workflow for hybrid function-based modelling technique

with customised functions, analytical formulae, FVRML/FX3D with replaceable modules, standalone interactive modelling software, and the web-based collaborative modelling tool. To use the tools properly when creating hybrid function-based models, a workflow for hybrid function-based modelling technique is proposed. The diagram of the proposed workflow can be seen in Figure 8.19.

The whole proposed workflow contains four stages (shown as dotted round rectangles). These four stages form a whole working pipeline.

The first stage is the *basic model acquisition*. At this stage, the rough basic shape of the final model is created. The process can be done either by defining a model with several simple analytical functions, such as spheres, cylinders, boxes, cones, and planes, or by using a device which converts a model from real life into digital scan data, such as CT data, MRI data, or range scan data. In some cases, these two approaches can be employed concurrently in this stage to achieve some special effects. For example, the scan data of the real object can be cut with a plane in order to reveal the internal structure of the real-life object.

The second stage is the *function model representation*. At this stage, the basic shape defined in the *basic model acquisition* stage is represented by different function representation approaches. As proposed in Chapter 4 and Chapter 6, four different approaches can be used in this stage, i.e. *standard features in function scripts*, *customised functions in func-*

Chapter 8. Examples of using FVRML/FX3D

tion scripts, native customised functions, and representations in customised modules For analytically-defined basic models, usually they can be represented by function scripts in a straightforward way while only using the standard features provided by function scripts. For certain special modelling methods, such as blobby models, customised functions in function scripts may be more efficient. However, when the basic models are acquired by various sensor devices (e.g. CT, MRI, range scan devices) or are created by other function-based modelling applications, it would be too difficult and time-consuming to write function scripts to interpret the model. Thus, in these cases, usually either native customised functions or function representation in customised modules are used.

The third stage is the *model modification*. At this stage, the function-based models produced in the *function model representation* stage is gradually modified in order to produce a new function-based model. The modification can be done by modifying any function-based model defined by proper function representations. This stage is usually done with the help of an interactive software, since it might be very difficult for the users to estimate coordinates and the parameters of transformations for the desired modification. In this thesis, two modification tools, *web collaborative modelling tool* (Chapter 7) and *standalone interactive modelling tool* ([133]), are considered. However, any other third party modelling tool with function scripts support should be able to be integrated into this workflow flawlessly.

The final stage is the *final model visualisation*. At this stage, the modified function-based model produced by the function-based modelling tools is exported as an FVRML/FX3D file. These files are ready to be published on web, and can be visualised online. At this stage, since the function scripts are human-readable, it is still possible for the authors to edit the FVRML/FX3D files manually.

To illustrate the usage of the workflow, we created a function-based “bunny” using images of real rabbits and the Stanford bunny model as references (Figure 8.20). The process strictly follows the stages in the proposed workflow (Figure 8.21).

Figure 8.20: The Stanford bunny model

Figure 8.21(a) represents the *basic shape acquisition* stage. At this stage, the basic model is acquired by CT scanner. The scanned result is represented by three dimensional volume data. The volume data is packed as raw density values of all voxels scanned. The figure is a slice of the whole volume data. Since the CT scanned data only contains density values, there is no colour information. Some details are missing due to the accuracy loss of the CT scanner.

Figure 8.21(b) represents the *function model representation* stage. At this stage, the basic model is converted into a function script. However, since voxel support was not included as a part of the standard features in function scripts, a new native customised function, trilinear interpolation, was added in order to reconstruct a function-based model from the voxels. Employing the trilinear interpolation function, it is straightforward to write a function script representing the CT data once the desired threshold density value is determined.

Figure 8.21(c) represents the *model modification* stage. At this stage, the *web-based collaborative modelling tool* is used to gradually modify and colour the base model. When modelling this bunny, the actual process can be divided into four steps as shown in Figure 8.22.

Chapter 8. Examples of using FVRML/FX3D

(a) Basic model

(b) Function representation

(c) Model modification

(d) Final model in a VRML scene

Figure 8.21: Modelling a function-based bunny

Chapter 8. Examples of using FVRML/FX3D

(a) Original function-based model

(b) Base colour applied

(c) Geometry modifications applied

(d) Colour modifications applied

Figure 8.22: Model modification in web-based modelling tool

Chapter 8. Examples of using FVRML/FX3D

First, the base model is loaded into the modelling environment (Figure 8.22(a)). At this step, the model is shown as a gray object, which is the default colour of VRML and X3D. Second, the base colour is applied to the base model (Figure 8.22(b)). In this case, the base colour of the bunny changes from bright white (left bottom corner) to dark gray (right top corner). Next, the geometry of the bunny mouth and nose is slightly modified (Figure 8.22(c)). The modification is done by applying operations to the function scripts without changing the original CT volume data. Finally, the eyes and the ears of the bunny are coloured interactively as dark red and semi-transparent dark yellow, respectively, by various tools (Figure 8.22(d)).

Figure 8.21(d) represents the *final model visualisation* stage. At this stage, the FVRML file is exported from the *web collaborative modelling tool*. The figure is a snapshot of the model rendered using FVRML/FX3D. The result of the model together with other standard VRML objects is illustrated in Figure 8.23.

The same workflow can also be applied to other applications, such as medical image visualisation. For example, a function-based model can be obtained via CT/MRI/PET scans. It is known that these data are prone to have noises, due to the nature of the scanners. Though many automatic algorithms are proposed, it is still the best to manually do denoising and segmentation. However, editing and colouring voxels would always be challenging. Usually, the volume data can not be modified and segmented interactively and intuitively. However, with the help of the hybrid function-based model and its workflow, it is possible to modify and colour any volume data interactively and intuitively on web. The modification to volume model can be saved as a separate patch file without touching the original voxels. Figure 8.24 shows a human brain reconstructed from CT volume data. The function-based brain model is geometrically modified and coloured according to the density in order to study the cross section. The proposed workflow has already being used on several on-going projects for real medical applications. The above example is from a part of the project “An Integrated Approach to Improving Measurement Accuracy of Magnetic Resonance Brain Images to Support Change

Chapter 8. Examples of using FVRML/FX3D

Figure 8.23: Bunny created according to the workflow together with standard VRML objects

Figure 8.24: Human brain coloured according to density

Detection in Large Cohort Studies (SBIC RP C-012/2006)”. In this project, the large MRI data resides on client machines. In case new MRI data is desired, it will be downloaded once to the client machine automatically. When the MRI data needs collaborative interactive modification and colouring, the modification and colour defined by function scripts are exchanged between clients. By this, the transmission of large data has been successfully avoided.

8.6 Summary

In this chapter, many examples are presented to illustrate all features provided by the FVRML/FX3D extension using the framework proposed in this thesis. The modelling framework together with the extension is proven to be useful. When the modelling tool is not used, the users have to be very knowledgeable about mathematical functions. Usually, a trial-and-error approach has to be adopted, since it is difficult to imagine the exact coordinates and orientation of the models defined by mathematical functions. When using the modelling tool, it is not necessary to know the exact coordinates and orientation, since the modification can be done in a visualised environment.

Chapter 8. Examples of using FVRML/FX3D

Though FVRML and FX3D are very flexible and feature-rich, they still have some limitations when modelling specific objects. For example, volumetric clouds which are clusters of infinite small points are very difficult to be modelled in FVRML/FX3D. This is caused by the fact that VRML and X3D are designed mainly to render surfaces (i.e. polygons). The ability to render points and voxels are very limited.

There are also challenges to convert parametrically defined objects to implicitly/explicitly-defined ones to enjoy all the advantages of using set-theoretic operations, as well as to convert implicitly/explicitly-defined objects to parametrically-defined. Though there are ways how it can be approached mathematically, it usually requires very long computational time which makes it prohibitive for using in shared virtual spaces.

Chapter 9

Conclusion and future work

This chapter summaries the work presented in this thesis. The conclusion is made, and future work is discussed.

9.1 Conclusion

Function-based shape modelling approaches are proved to be feasible for boosting web visualisation by providing compact editable models with any level of detail. This conclusion is supported by the thorough study of the research papers, projects and software for web visualisation and shape modelling. The main contributions of this thesis consist of the general function-based rendering pipeline, the hybrid function-based web shape modelling framework, the function-based extension of VRML and X3D with support of time-dependent models, and the web-based interactive application framework.

To understand the nature of the function-based shape modelling techniques, a general function-based rendering pipeline is proposed in this thesis. It defines the overall structure which all function-based shape rendering system should follow. The proposed rendering pipeline shows that it is possible to use functions to describe all aspects of a virtual object, including visual properties, physical properties, and other properties which can be perceived

Chapter 9. Conclusion and future work

directly or indirectly by human. Furthermore, it allows the free mixture of all kinds of functions to define a single virtual object.

Based on the understanding of the general function-based rendering pipeline, a hybrid function-based approach to web-based shape modelling is designed. Since this thesis focuses on visualisation, the idea of this method is to concurrently use different classes of mathematical functions to define geometry and visual appearances of shapes (i.e. colours and geometric textures) as well as operations on them. When modelling a virtual object with the proposed framework, the geometry and the appearances are defined in their respective domains as independent geometric shapes. These shapes are then mapped and merged into the final shape with colours and geometric textures. The final shape is defined in the same domain as the original geometry. The final shape can be either static shape or time-dependent.

Since it is known that the function-based models are generally much smaller in file size than the commonly used polygon and voxel-based models, it is efficient to exchange these models across the Internet for web visualisation and collaborative interactive applications. In order to develop software tools which are capable of visualising the proposed hybrid function-based models, a general function-based web visualisation pipeline and a web-based interactive application framework using function-based models are proposed.

The general function-based web visualisation pipeline is designed to enable any browsers which can render polygons, points, and/or voxels, to visualise hybrid function-based models. The essential idea of the pipeline is to convert the final shape produced by the hybrid function-based web shape modelling framework into primitives which can be rendered by target browsers. If the shape is time-dependent, the conversion has to proceed continuously. In this case, each conversion produces a snapshot of the time-dependent shape at current time.

The web-based interactive application framework is proposed for developing real-life collaborative applications. Because of the small size of the function-based models, it is possible

Chapter 9. Conclusion and future work

to synchronise the model modifications on the client computers. As the function-based models can provide arbitrary level of detail, the users are able to select the required level of detail without changing the model. With the framework, it is also possible to use the function-based models together with large data sets coming from data acquisition devices such as CT, MRI, or range scanners. In these cases, only the modifications to the models are exchanged across the network while the original data are kept on the client computers or shared on the server.

To solve real-life application problems, a workflow is designed to create function-based models with various kinds of tools is presented in this thesis. The workflow includes data acquisition and initial model definition, interactive modification, and publishing on the web.

To prove the feasibility of the proposed approaches, the Function-based Extension of the Virtual Reality Modelling Language and Extensible 3D (FVRML/FX3D) has been implemented. In the extension, function scripts with analytical formulae can be used to define hybrid function-based models, including geometries, appearances, and operations. The implementation of the FVRML/FX3D plugin strictly follows the proposed hybrid function-based web shape modelling framework and the general function-based visualisation pipeline.

The implemented FVRML/FX3D plugin has been proved to be useful, since it serves well as a basic function-based web visualisation software tool. However, many real-life applications require more complex features and more realistic models than the standard features which the plugin can provide. Hence, several extensions have been implemented following the feedback from the potential users. The plugin can be extended by either module replacement or writing native customised functions, which can be used to work with any appropriate function-based models in FVRML/FX3D. To illustrate this, a module which can define NURBS surface and a native customised function which can import CT/MRI voxel data are implemented.

Based on FVRML/FX3D, a collaborative modelling tool has been implemented as an

Chapter 9. Conclusion and future work

illustration of the proposed web-based interactive application framework. The modelling tool can help the users to create hybrid function-based models by interactive modifications. It allows the users to interactively modify the geometry and the appearance of a basic shape. Together with the FVRML/FX3D plugin, they illustrate the workflow used to create function-based models which can be rendered by FVRML/FX3D. The modelling tool also serves as a foundation for other collaborative web-based FVRML/FX3D applications. New features and user interface can be added to the implemented modelling tool to create new function-based interactive applications. This concept has already been used in the on-going project “An Integrated Approach to Improving Measurement Accuracy of Magnetic Resonance Brain Images to Support Change Detection in Large Cohort Studies (SBIC RP C-012/2006)”, which requires to colour and modify CT/MRI data interactively and collaboratively on web. The tool is also used in the project of web-based virtual orthopaedic training and in teaching undergraduate students computer graphics in Nanyang Technological University.

9.2 Future work

While working on the applications of the proposed modelling framework and tools and developing new plugins to the FVRML/FX3D extensions via the interface provided, several areas of future research have been identified.

9.2.1 Advanced modelling tools

In this thesis, two function-based modelling tools have been mentioned which have the capability of exporting function-based models as FVRML/FX3D files. One is the web-based collaborative modelling tool presented in Chapter 7, the other one is the modelling tool developed within the scope of the Interactive Function-based Shape Modelling project [133]. However, these tools may still be difficult for those who do not have enough mathematical

Chapter 9. Conclusion and future work

knowledge, since they require the users to type in the base shape as mathematical formulae. This limitation can be removed by applying the sketch-up technology.

The essence of sketch-up technology is to allow the users to create 3D models from 2D drawings with intuitive user interactions. Recently, the sketch-up technology is introduced to create 3D function-based models in ShapeShop [146]. The ShapeShop modelling system employs skeleton-based models, blending operation, and Boolean operations to create function-based models. Since all these features are already available in the existing implementation, it is feasible to develop a new module for FVRML/FX3D to support the models created and exported by such modelling system. Furthermore, by combining the ideas presented in ShapeShop system and the web-based collaborative modelling system presented in this thesis, a sketch-based collaborative modelling system may be built using FVRML/FX3D.

9.2.2 Fast polygonisation

In the current implementation, processing of the function definitions relies only on the CPUs of the client computer. Because of the development of the parallel computing technologies and the increasing processing power of Graphics Processing Unit (GPU), it is feasible to propose fast parallel polygonisation algorithm not only using client computer CPUs, but employing the processing power of GPUs on the client computer and CPUs/GPUs on other computers in the same network as well.

Due to the rapid evolution of the graphics hardware, GPUs are no longer a chip dedicated to the graphics processing features. Nowadays, modern GPUs are built with programmable visualisation pipelines, including the programmable vertex process and pixel process. These features are called programmable vertex and fragment shaders respectively. Furthermore, compared to the processing power of traditional GPUs, which have only 8 bits for each colour component (red, green, blue, and possible alpha channel), modern GPUs are equipped with more accurate vectorised computation facilities. It is now possible to support 16, 24 or even

Chapter 9. Conclusion and future work

32 bits per colour component. Each colour component can be represented by a floating-point number. This makes the general processing on GPU possible. The most significant advantages of GPU programming would be the extreme computation power and the parallel processing ability. Since many polygonisation algorithms (e.g. Marching Cubes) are naturally parallel, it is feasible to implement them using GPU programming. Some progress has already been achieved in this direction, such as the GPU-based Marching Cubes algorithm proposed by Johansson and Carr [147]. However, all the works use either density values (e.g. CT/MRI scanned data) or rely on CPU to evaluate the function. In such case, GPU is only responsible to generate and render the polygons according to the function values provided by the CPU. Moreover, since the memory of GPU is usually limited compared to the main memory of a computer, for high resolution rendering it is impossible to fit all function values into the GPU memory. Therefore, an on-the-fly GPU-based function evaluation approach should be proposed together with a GPU-based polygonisation algorithm. According to our experiments, function evaluation process is the most time-consuming step for complex function definitions. Since function evaluation procedure is naturally parallel, it can maximise the performance of GPU and lower the consumption of GPU memory.

Employing grid computing is another trend in many research areas including computer visualisation[148]. Grid computing uses a large number of commodity personal computers to form a cluster. Though the processing power of one personal computer is limited, by using thousands of them, the processing power would be enough to do complex computation. The major problem of this direction would be the accessibility of grids to the end users. It is impossible for each and every end-user to gain access to a local high performance computing grid. If a remote grid is provided by the vendor, the network delay would be unaffordable. Based on these considerations, the grid computing can only be a complementary approach.

9.2.3 Haptic device integration

In the hybrid function-based web shape modelling framework, there is no limitation imposed on the type of the properties defined by functions. Thus, the properties can be both visual properties and physical properties. In traditional visualisation area, only visual properties would be effective, since the final result of the visualisation would be images. With the development in the human computer interaction area, there exist haptic devices which can be used for touching the virtual objects.

There are several issues to implement physical properties in FVRML and FX3D. First, no existing VRML/X3D browser supports haptic rendering. There is no meaning to introduce the physical properties to FVRML and FX3D without proper force feedback ability. Certain force feedback mechanism must be implemented as an extension to the existing VRML/X3D browsers. Such extension should work not only for function-defined objects, but polygon-based objects as well. Second, the physical model should be built in order to calculate the direction and the amount of force given to the user. The force and the direction should be estimated according to the physical properties defined by the models and the position/direction of the haptic device. Finally, the visualisation pipeline of the function-based extension should be revised. New modules implementing new mapping functions should be introduced into the pipeline. The new visualisation pipeline should also communicate with the haptic rendering mechanism provided by other extension.

Acronyms

API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
BIFS	Binary for Scenes
BNF	Backus Naur Form
CAD	Computer-Aided Design
COLLADA	Collaborative Design Activity
COM	Component Object Model
CSG	Constructive Solid Geometry
DLL	Dynamic Link Library
EAI	External Authoring Interface
FBA	Facial and Body Animation
FVRML	Function-based Extension of Virtual Reality Modelling Language
FX3D	Function-based Extension of Extensible 3D
GPU	Graphics Processing Unit
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
JIT	Just-In-Time
JRE	Java Runtime Environment
LAN	Local Area Network
NURBS	Non-Uniform Rational B-Spline
SAI	Scene Access Interface
SDK	Software Development Kit
SIMD	Single Instruction, Multiple Data
SSE	Streaming SIMD Extensions
U3D	Universal 3D
VRML	Virtual Reality Modelling Language
X3D	Extensible 3D
XML	Extensible Markup Language
YACC	Yet-Another Compiler Compiler
XSLT	Extensible Stylesheet Language Transformation

Author's Publications

Journal papers:

1. Qi Liu and Alexei Sourin, "Function-based Extension of VRML and X3D," *Computer Graphics & Geometry*, 2006, 8:3, pp. 21–37.
2. Qi Liu and Alexei Sourin, "Function-defined Shape Metamorphoses in Visual Cyber-worlds," *The Visual Computer*, Springer, 2006, 22:12, pp. 977–990.
3. Qi Liu and Alexei Sourin, "Function-based Shape Modelling Extension of the Virtual Reality Modelling Language," *Computers and Graphics*, Elsevier, 2006, 30:4, pp. 629–645.

Conference papers:

1. Qi Liu and Alexei Sourin, "Function-based Shape Modeling and Visualization in X3D," *Proceedings of Web3D 2006 Symposium*, 2006, United States, pp. 131-141.
2. Qi Liu and Alexei Sourin, "Function-defined Shape Metamorphoses in VRML," *Proceedings of GRAPHITE 2005*, 2005, New Zealand, pp. 339-346.
3. Qi Liu and Alexei Sourin, "Function-based Time-dependent Shape Modeling on the Web," *Proceedings of Cyberworlds 2005*, 2005, Singapore, pp. 35–42.

AUTHOR'S PUBLICATIONS

4. Qi Liu and Alexei Sourin, "Function-based Representation of Complex Geometry and Appearance," *Proceedings of Web3D 2005 Symposium*, 2005, United Kingdom, pp. 123–134.
5. Alexei Sourin, Konstantin Levinski, and Qi Liu, "Virtual Campus—It is Fun and Educational," *Proceedings of MAAP 2004*, 2004, Singapore.
6. Qi Liu and Alexei Sourin, "Analytically-defined Collaborative Shape Modeling in VRML," *Proceedings of Cyberworlds 2004*, 2004, Japan, pp. 70–77.

Bibliography

- [1] Virtual Reality Modeling Language,
<http://www.web3d.org/x3d/specifications/vrml/ISO-IEC-14772-VRML97/>.
- [2] Extensible 3D,
<http://www.web3d.org/x3d/specifications/ISO-IEC-19775-X3DAbstractSpecification/>.
- [3] C. Upson, T. Faulhaber, D. Kamins, D. Laidlaw, D. Schlegel, J. Vroom, R. Gurwitz, and A. van Dam, “The application visualization system: A computational environment for scientific visualization,” *IEEE Computer Graphics & Applications*, vol. 9, no. 4, pp. 30–42, 1989.
- [4] M. Bender, R. Klein, A. Disch, and A. Ebert, “A functional framework for web-based information visualization systems,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 6, no. 1, pp. 8–23, 2000.
- [5] HyperText Markup Language,
<http://www.w3.org/MarkUp/>.
- [6] I. R. Holmes and R. S. Kalawsky, “The RealityGrid PDA and smartphone clients: Developing effective handheld user interfaces for e-science,” in *Proceedings of the UK e-Science All Hands Meeting 2006*, 2006.

BIBLIOGRAPHY

- [7] K. Engel, T. Ertl, P. Hastreiter, B. Tomandl, and K. Eberhardt, “Combining local and remote visualization techniques for interactive volume rendering in medical applications,” in *Proceedings of the conference on Visualization '00*. IEEE Computer Society Press, 2000, pp. 449–452.
- [8] M. Zyda, J. Hiles, A. Mayberry, C. Wardynski, M. V. Capps, B. Osborn, R. Shilling, M. Robaszewski, and M. J. Davis, “Entertainment R&D for defense.” *IEEE Computer Graphics and Applications*, vol. 23, no. 1, pp. 28–36, 2003.
- [9] W. Qi, Z. Song, and G. Wu, “Volume rendering on internet,” in *the First International Symposium on 3D Data Processing Visualization and Transmission*, 2002, pp. 70–74.
- [10] D. Malandrino, G. Meo, G. Palmieri, and V. Scarano, “3WPS : a 3D web-based process visualization framework,” in *the First International Symposium on 3D Data Processing Visualization and Transmission*, 2002, pp. 48–51.
- [11] B.-Y. Chen and T. Nishita, “Multiresolution streaming mesh with shape preserving and QoS-like controlling,” in *Proceeding of the seventh international conference on 3D Web technology*. ACM Press, 2002, pp. 35–42.
- [12] M. Grabner, “WebCAME: a light-weight modular client/server multiresolution rendering system,” in *Proceeding of the eighth international conference on 3D web technology*. ACM Press, 2003, pp. 63–72.
- [13] S. Rusinkiewicz and M. Levoy, “Streaming QSplat: a viewer for networked visualization of large, dense models,” in *Proceedings of the 2001 symposium on Interactive 3D graphics*, 2001, pp. 63–68.
- [14] Hypertext Transfer Protocol,
<http://www.w3.org/Protocols/>.

BIBLIOGRAPHY

- [15] M. Deering, "Geometry compression," in *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*. ACM Press, 1995, pp. 13–20.
- [16] G. Taubin and J. Rossignac, "Geometric compression through topological surgery," *ACM Trans. Graph.*, vol. 17, no. 2, pp. 84–115, 1998.
- [17] C. Touma and C. Gotsman, "Triangle mesh compression," in *Graphics Interface'98 Conference Proceedings*, 1998, pp. 26–34.
- [18] J. Li and C. C. Kuo, "A dual graph approach to 3D triangular mesh compression," in *Proceedings of the 1998 IEEE International Conference on Image Processing (ICIP-98)*, B. Werner, Ed. IEEE Computer Society, 1998, pp. 891–894.
- [19] S. Gumhold and W. Straßer, "Real time compression of triangle mesh connectivity," in *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. ACM Press, 1998, pp. 133–140.
- [20] J. Rossignac, "Edgebreaker: Connectivity compression for triangle meshes," *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, no. 1, pp. 47–61, 1999.
- [21] A. Szymczak, D. King, and J. Rossignac, "An Edgebreaker-based efficient compression scheme for connectivity of regular meshes," in *Proceedings of 12th Canadian Conference on Computational Geometry*, 2000, pp. 257–264.
- [22] M. Isenburg and J. Snoeyink, "Mesh collapse compression," in *Proceedings of the fifteenth annual symposium on Computational geometry*. ACM Press, 1999, pp. 419–420.
- [23] D. King, J. Rossignac, and A. Szymczak, "Connectivity compression for irregular quadrilateral meshes," Georgia Tech, Technical Report TR-99-36, 1999.

BIBLIOGRAPHY

- [24] M. Isenburg and J. Snoeyink, “Face fixer: compressing polygon meshes with properties,” in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 2000, pp. 263–270.
- [25] B. Kronrod and C. Gotsman, “Efficient coding of non-triangular mesh connectivity,” in *Proceedings of Pacific Conference on Computer Graphics and Applications 2000*, 2000, pp. 235–242.
- [26] Universal 3D,
<http://www.ecma-international.org/publications/standards/Ecma-363.htm>.
- [27] Collaborative Design Activity,
<http://collada.org/>.
- [28] M. Alexa, J. Behr, and W. Müller, “The morph node,” in *Proceedings of the fifth symposium on Virtual reality modeling language (Web3D-VRML)*, 2000, pp. 29–34.
- [29] Humanoid Animation,
<http://h-anim.org/Specifications/H-Anim200x>.
- [30] T. Naka, Y. Mochizuki, T. Hijiri, T. Cornish, and S. Asahara, “A compression/decompression method for streaming based humanoid animation,” in *VRML '99: Proceedings of the fourth symposium on Virtual reality modeling language*. New York, NY, USA: ACM Press, 1999, pp. 63–70.
- [31] Y.-L. Liu and T.-Y. Li, “A multi-user virtual environment system with extensible animations,” in *Web3D '03: Proceeding of the eighth international conference on 3D Web technology*. New York, NY, USA: ACM Press, 2003, pp. 129–134.
- [32] I. S. Pandzic, “Facial animation framework for the web and mobile platforms,” in *Web3D '02: Proceeding of the seventh international conference on 3D Web technology*. New York, NY, USA: ACM Press, 2002, pp. 27–34.

BIBLIOGRAPHY

- [33] M. Reddy, L. Iverson, and Y. G. Leclerc, “Under the hood of GeoVRML 1.0,” in *Proceedings of the fifth symposium on Virtual reality modeling language (Web3D-VRML)*. ACM Press, 2000, pp. 23–28.
- [34] J. Behr and M. Alexa, “Volume visualization in VRML,” in *Proceedings of the sixth international conference on 3D Web technology*. ACM Press, 2001, pp. 23–27.
- [35] E. Fogel, D. Cohen-Or, R. Ironi, and T. Zvi, “A web architecture for progressive delivery of 3D content,” in *Proceedings of the sixth international conference on 3D Web technology*. ACM Press, 2001, pp. 35–41.
- [36] M. Isenburg and J. Snoeyink, “Coding polygon meshes as compressable ASCII,” in *Proceeding of the seventh international conference on 3D Web technology*. ACM Press, 2002, pp. 1–10.
- [37] M. Hosseini and N. D. Georganas, “MPEG-4 BIFS streaming of large virtual environments and their animation on the web,” in *Web3D '02: Proceeding of the seventh international conference on 3D Web technology*. New York, NY, USA: ACM Press, 2002, pp. 19–25.
- [38] ParallelGraphics Cortona,
<http://www.parallelgraphics.com/>.
- [39] Blaxxun Contact,
<http://www.blaxxun.com/>.
- [40] Geometrek DeepMatrix,
<http://www.geometrek.com/products/deepmatrix.html>.
- [41] Y. Araki, “VSPLUS: A high-level multi-user extension library for interactive VRML worlds,” in *Proceedings of the third symposium on Virtual reality modeling language*. ACM Press, 1998, pp. 39–47.

BIBLIOGRAPHY

- [42] Bitmanagement BS Contact VRML,
<http://www.bitmanagement.de/>.
- [43] J. Behr, P. Dähne, and M. Roth, “Utilizing X3D for immersive environments,” in *Web3D '04: Proceedings of the ninth international conference on 3D Web technology*. New York, NY, USA: ACM Press, 2004, pp. 71–78.
- [44] P. Figueroa, O. Medina, R. Jiménez, J. Martínez, and C. Albarracín, “Extensions for interactivity and retargeting in X3D,” in *Web3D '05: Proceedings of the tenth international conference on 3D Web technology*. New York, NY, USA: ACM Press, 2005, pp. 103–110.
- [45] L. P. Soares and M. K. Zuffo, “JINX: an X3D browser for VR immersive simulation based on clusters of commodity computers,” in *Web3D '04: Proceedings of the ninth international conference on 3D Web technology*. New York, NY, USA: ACM Press, 2004, pp. 79–86.
- [46] D. Brutzman, “X3D-edit authoring for extensible 3D (X3D) graphics,” in *SIGGRAPH '03: Educators program from the 30th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press, 2003, pp. 1–1.
- [47] G. N. M. de Carvalho, T. Gill, and T. Parisi, “X3D programmable shaders,” in *Web3D '04: Proceedings of the ninth international conference on 3D Web technology*. New York, NY, USA: ACM Press, 2004, pp. 99–108.
- [48] Extensible Markup Language,
<http://www.w3.org/TR/2004/REC-xml11-20040204>.
- [49] Extensible Stylesheet Language Transformation,
<http://www.w3.org/TR/xslt>.
-

BIBLIOGRAPHY

- [50] P. McIntosh, M. Hamilton, and R. van Schyndel, “X3D-UML: enabling advanced UML visualisation through X3D,” in *Web3D '05: Proceedings of the tenth international conference on 3D Web technology*. New York, NY, USA: ACM Press, 2005, pp. 135–142.
- [51] R. E. Hetherington and J. P. Scott, “Adding a fourth dimension to three dimensional virtual spaces,” in *Web3D '04: Proceedings of the ninth international conference on 3D Web technology*. New York, NY, USA: ACM Press, 2004, pp. 163–172.
- [52] N. F. Polys, “Stylesheet transformations for interactive visualization: towards a Web3D chemistry curricula,” in *Web3D '03: Proceeding of the eighth international conference on 3D Web technology*. New York, NY, USA: ACM Press, 2003, pp. 85–ff.
- [53] Active Worlds,
<http://www.activeworlds.com/>.
- [54] Second Life,
<http://www.secondlife.com/>.
- [55] D. R. Peachey, “Solid texturing of complex surfaces,” in *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*. ACM Press, 1985, pp. 279–286.
- [56] J. P. Lewis, “Algorithms for solid noise synthesis,” in *Proceedings of the 16th annual conference on Computer graphics and interactive techniques*. ACM Press, 1989, pp. 263–270.
- [57] K. Perlin and E. M. Hoffert, “Hypertexture,” in *SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press, 1989, pp. 253–262.
-

BIBLIOGRAPHY

- [58] C. Loop and T. D. DeRose, "Generalized b-spline surfaces of arbitrary topology," in *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*. ACM Press, 1990, pp. 347–356.
- [59] H. Grahm, T. Volk, and H. J. Wolters, "NURBS in VRML," in *Proceedings of the fifth symposium on Virtual reality modeling language (Web3D-VRML)*. ACM Press, 2000, pp. 35–43.
- [60] D. R. Forsey and R. H. Bartels, "Hierarchical b-spline refinement," in *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*. ACM Press, 1988, pp. 205–212.
- [61] T. W. Sederberg, J. Zheng, A. Bakenov, and A. Nasri, "T-splines and T-NURCCs," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 477–484, 2003.
- [62] T. W. Sederberg, D. L. Cardon, G. T. Finnigan, N. S. North, J. Zheng, and T. Lyche, "T-spline simplification and local refinement," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 276–283, 2004.
- [63] A. Ricci, "A constructive geometry for computer graphics," *The Computer Journal*, vol. 16, no. 2, pp. 157–160, 1973. [Online]. Available: <http://comjnl.oxfordjournals.org/cgi/content/abstract/16/2/157>
- [64] A. A. Pasko, V. Adzhiev, A. Sourin, and V. V. Savchenko, "Function representation in geometric modeling: concepts, implementation and applications," *The Visual Computer*, vol. 11, no. 8, pp. 429–446, 1995.
- [65] B. Wyvill and K. van Overveld, "Polygonization of implicit surfaces with constructive solid geometry," *International Journal of Shape Modeling*, vol. 2, no. 4, pp. 257–274, 1996.

BIBLIOGRAPHY

- [66] J. F. Blinn, “A generalization of algebraic surface drawing,” *ACM Trans. Graph.*, vol. 1, no. 3, pp. 235–256, 1982.
- [67] H. Nishimura, M. Hirai, T. Kawai, T. Kawata, I. Shirakawa, and K. Omura, “Object modeling by distribution function and a method of image generation,” *Transactions on IECE*, vol. 68-D, no. 4, pp. 718–725, 1985.
- [68] G. Wyvill, C. McPheeters, and B. Wyvill, “Data structure for soft objects,” *The Visual Computer*, vol. 2, no. 4, pp. 227–234, August 1986.
- [69] J. Bloomenthal and K. Shoemake, “Convolution surfaces,” in *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*. ACM Press, 1991, pp. 251–256.
- [70] A. Angelidis and M.-P. Cani, “Adaptive implicit modeling using subdivision curves and surfaces as skeletons,” in *Proceedings of the seventh ACM symposium on Solid modeling and applications*, 2002, pp. 45–52.
- [71] S. Hornus, A. Angelidis, and M.-P. Cani, “Implicit modelling using subdivision-curves,” *The Visual Computer*, vol. 19, no. 2-3, pp. 94–104, May 2003. [Online]. Available: <http://www-imagis.imag.fr/Publications/2003/HAC03>
- [72] A. Angelidis, P. Jepp, and M.-P. Cani, “Implicit modeling with skeleton curves: Controlled blending in contact situations,” in *Proceedings of Shape Modeling International 2002*, ACM. IEEE Computer Society Press, 2002, banff, Alberta, Canada. [Online]. Available: <http://www-imagis.imag.fr/Publications/2002/AJC02>
- [73] V. Savchenko and N. Kojekine, “An approach to blend surfaces,” in *Proceedings of CGI’2002*, 2002, pp. 139–150.

BIBLIOGRAPHY

- [74] C. L. Bajaj and G. Xu, “A-splines: Local interpolation and approximation using Gk-continuous piecewise real algebraic curves,” *Computer Aided Geometric Design*, vol. 16, pp. 557–578, 1999.
- [75] C. L. Bajaj, J. Chen, and G. Xu, “Modeling with cubic A-patches,” *ACM Transactions on Graphics*, vol. 14, no. 2, pp. 103–133, 1995.
- [76] C. L. Bajaj, G. Xu, R. J. Holt, and A. N. Netravali, “NURBS approximation of A-splines and A-patches,” *International Journal of Computational Geometry and Applications*, vol. 13, no. 5, pp. 359–389, 2003.
- [77] S. F. Frisken, R. N. Perry, A. P. Rockwood, and T. R. Jones, “Adaptively sampled distance fields: a general representation of shape for computer graphics,” in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, 2000, pp. 249–254.
- [78] R. Zonenschein, J. Gomes, L. Velho, and L. H. de Figueiredo, “Controlling texture mapping onto implicit surfaces with particle systems,” in *Proceedings of the Third International Workshop on Implicit Surfaces*, 1998, pp. 131–138.
- [79] M. Tigges and B. Wyvill, “A field interpolated texture mapping algorithm for skeletal implicit surfaces,” in *Proc. CG International 99*. IEEE Computer Society ISBN: 0-7695-0185-0, 1999, pp. 25–33.
- [80] G. Wyvill, B. Wyvill, and C. McPheeters, “Solid texturing of soft objects,” in *CG International ’87 on Computer graphics 1987*. Springer-Verlag New York, Inc., 1987, pp. 129–141.
- [81] A. Pasko, V. Adzhiev, B. Schmitt, and C. Schlick, “Constructive hypervolume modeling,” *Graphical Models*, vol. 63, no. 6, pp. 413–442, November 2001.

BIBLIOGRAPHY

- [82] B. Schmitt, A. A. Pasko, V. Adzhiev, and C. Schlick, “Constructive texturing based on hypervolume modeling,” *Journal of Visualization and Computer Animation*, vol. 12, no. 5, pp. 297–310, 2001.
- [83] A. Sourin, A. A. Pasko, and V. V. Savchenko, “Using real functions with application to hair modelling,” *Computers & Graphics*, vol. 20, no. 1, pp. 11–19, 1996.
- [84] K. Levinski and A. Sourin, “Interactive function-based shape modeling for cyber-worlds,” in *Proceedings of 2004 International Conference on Cyberworlds*, November 2004, pp. 54–61.
- [85] G. Turk, H. Q. Dinh, J. F. O’Brien, and G. Yngve, “Implicit surfaces that interpolate,” in *Proceedings of International Conference on Shape Modeling and Applications*, May 2001.
- [86] J. C. Hart, E. Bachta, W. Jarosz, and T. Fleury, “Using particles to sample and control more complex implicit surfaces,” in *Proceedings of Shape Modeling International 2002*, May 2002, pp. 129–136.
- [87] H. Du and H. Qin, “Interactive shape design using volumetric implicit PDEs,” in *Proceedings of the eighth ACM symposium on Solid modeling and applications*, 2003, pp. 235–246.
- [88] N. Kojekine, V. Savchenko, and I. Hagiwara, “Surface reconstruction based on compactly supported radial basis functions,” *Geometric modeling: techniques, applications, systems and tools*, pp. 218–231, 2004.
- [89] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans, “Reconstruction and representation of 3D objects with radial basis functions,” in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 2001, pp. 67–76.

BIBLIOGRAPHY

- [90] Y. Ohtake, A. Belyaev, and H.-P. Seidel, “3D scattered data approximation with adaptive compactly supported radial basis functions,” in *Proceedings of Shape Modeling International 2004*, 2004, pp. 31–39.
- [91] N. Kojekine, I. Hagiwara, and V. Savchenko, “Software tools using CSRBFs for processing scattered data,” *Computers & Graphics*, vol. 27, no. 2, pp. 309–317, 2003.
- [92] R. Allègre, A. Barbier, E. Galin, and S. Akkouche, “A hybrid shape representation for free-form modelling,” in *Proceedings of the Shape Modeling International 2004 (SMI’04)*. IEEE Computer Society, 2004, pp. 7–18.
- [93] T. Whitted, “An improved illumination model for shaded display,” *Commun. ACM*, vol. 23, no. 6, pp. 343–349, 1980.
- [94] A. S. Glassner, “Space subdivision for fast ray tracing,” *IEEE Computer Graphics and Applications*, vol. 4, no. 10, pp. 15–22, October 1984.
- [95] I. Wald, P. Slusallek, C. Benthin, and M. Wagner, “Interactive rendering with coherent ray tracing,” in *EG 2001 Proceedings*, A. Chalmers and T.-M. Rhyne, Eds., vol. 20(3). Blackwell Publishing, 2001, pp. 153–164. [Online]. Available: citeseer.nj.nec.com/wald01interactive.html
- [96] J. Schmittler, I. Wald, and P. Slusallek, “SaarCOR – a hardware architecture for ray tracing,” in *Proceedings of the conference on Graphics Hardware 2002*, Saarland University. Eurographics Association, 2002, pp. 27–36, available at <http://www.openrt.de>.
- [97] J. G. Cleary, B. M. Wyvill, G. M. Birtwistle, and R. Vatti, “Multiprocessor ray tracing,” *Computer Graphics Forum*, vol. 5, no. 1, pp. 3–12, 1986.

BIBLIOGRAPHY

- [98] I. Wald, T. Kollig, C. Benthin, A. Keller, and P. Slusallek, “Interactive global illumination using fast ray tracing,” in *Proceedings of the 13th Eurographics workshop on Rendering*. Eurographics Association, 2002, pp. 15–24.
- [99] I. Wald, C. Benthin, and P. Slusallek, “A simple and practical method for interactive ray tracing of dynamic scenes,” Computer Graphics Group, Saarland University, Tech. Rep., February 2002.
- [100] D. Ghazanfarpour and J.-M. Dischler, “A beam tracing method with precise antialiasing for polyhedral scenes,” *Computers & Graphics*, vol. 22, no. 1, pp. 103–115, 1998.
- [101] M. Wand and W. Straßer, “Multi-resolution point-sample raytracing,” in *Graphics Interface*, CIPS, Canadian Human-Computer Communication Society. A K Peters, June 2003, pp. 139–148, iISBN 1-56881-207-8, ISSN 0713-5424.
- [102] M. Attene, B. Falcidieno, M. Spagnuolo, and G. Wyvill, “A mapping-independent primitive for the triangulation of parametric surfaces,” *Graph. Models*, vol. 65, no. 5, pp. 260–273, 2003.
- [103] P. Crossno and E. Angel, “Isosurface extraction using particle systems,” in *Proceedings of the 8th conference on Visualization '97*. IEEE Computer Society Press, 1997, pp. 495–498.
- [104] Z. J. Wood, P. Schröder, D. Breen, and M. Desbrun, “Semi-regular mesh extraction from volumes,” in *Proceedings of the conference on Visualization '00*. IEEE Computer Society Press, 2000, pp. 275–282.
- [105] K. van Overveld and B. Wyvill, “Shrinkwrap: An efficient adaptive algorithm for triangulating an iso-surface,” *The Visual Computer*, vol. 20, no. 6, pp. 362–379, 2004.

BIBLIOGRAPHY

- [106] B. T. Stander and J. C. Hart, “Guaranteeing the topology of an implicit surface polygonization for interactive modeling,” in *SIGGRAPH '05: ACM SIGGRAPH 2005 Courses*. New York, NY, USA: ACM Press, 2005, p. 44.
- [107] W. E. Lorensen and H. E. Cline, “Marching cubes: A high resolution 3D surface construction algorithm,” *ACM SIGGRAPH Computer Graphics*, vol. 21, no. 4, pp. 163–169, July 1987.
- [108] E. Hartmann, “A marching method for the triangulation of surfaces,” *The Visual Computer*, vol. 14, no. 3, pp. 95–108, 1998.
- [109] J. Bloomenthal, “An implicit surface polygonizer,” *Graphics gems IV*, pp. 324–349, 1994.
- [110] G. M. Treece, R. W. Prager, and A. H. Gee, “Regularised marching tetrahedra: improved iso-surface extraction,” *Computers and Graphics*, vol. 23, no. 4, pp. 583–598, 1999. [Online]. Available: citeseer.nj.nec.com/treece98regularised.html
- [111] B. Crespín, “Dynamic triangulation of variational implicit surfaces using incremental delaunay tetrahedralization,” in *Proceedings of the 2002 IEEE symposium on Volume visualization and graphics*. IEEE Press, 2002, pp. 73–80.
- [112] Y. Livnat and C. Hansen, “View dependent isosurface extraction,” in *Proceedings of the conference on Visualization '98*. IEEE Computer Society Press, 1998, pp. 175–180.
- [113] K. Levinski and A. Sourin, “Interactive function-based artistic shape modeling,” in *Proceedings of International Symposium Cyber Worlds: Theory and Practice*, November 2002, pp. 521–528.

BIBLIOGRAPHY

- [114] M. Kazakov, A. Pasko, and V. Adzhiev, “Fast navigation through an FRep sculpture garden,” in *Shape Modeling International 2001*. IEEE Computer Society, 2001, pp. 104–113.
- [115] B. Gregorski, M. Duchaineau, P. Lindstrom, V. Pascucci, and K. I. Joy, “Interactive view-dependent rendering of large isosurfaces,” in *Proceedings of the conference on Visualization '02*. IEEE Computer Society, 2002, pp. 475–484.
- [116] Y. Ohtake and A. G. Belyaev, “Dual/primal mesh optimization for polygonized implicit surfaces,” in *Proceedings of the seventh ACM symposium on Solid modeling and applications*, 2002, pp. 171–178.
- [117] Y. Ohtake, A. Belyaev, and A. Pasko, “Dynamic mesh optimization for polygonized implicit surfaces with sharp features,” *The Visual Computer*, vol. 19, no. 2-3, pp. 115–126, 2003.
- [118] A. Hilton, A. Stoddart, J. Illingworth, and T. Windeatt, “Marching triangles: Range image fusion for complex object modelling,” in *Proceedings of IEEE International Conference on Image Processing*, vol. 2, 1996, pp. 381–384.
- [119] S. Akkouche and E. Galin, “Adaptive implicit surface polygonization using marching triangles,” *Computer Graphics Forum*, vol. 20, no. 2, pp. 67–80, June 2001.
- [120] T. Karkanis and A. J. Stewart, “Curvature-dependent triangulation of implicit surfaces,” *IEEE Comput. Graph. Appl.*, vol. 21, no. 2, pp. 60–69, 2001.
- [121] E. Hartmann, “A marching method for the triangulation of surfaces,” *The Visual Computer*, vol. 14, no. 3, pp. 95–108, 1998.
- [122] S. Rusinkiewicz and M. Levoy, “QSPat: a multiresolution point rendering system for large meshes,” in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, 2000, pp. 343–352.

BIBLIOGRAPHY

- [123] G. Schaufler and H. W. Jensen, "Ray tracing point sampled geometry," in *Proceedings of the Eurographics Workshop on Rendering Techniques 2000*. Springer-Verlag, 2000, pp. 319–328.
- [124] A. Bærentzen and N. Christensen, "Hardware accelerated point rendering of isosurfaces," *The Journal of WSCG*, vol. 11, no. 1, pp. 41–48, February 2003.
- [125] T. K. Dey and J. Hudson, "PMR: point to mesh rendering, a feature-based approach," in *Proceedings of the conference on Visualization '02*. IEEE Computer Society, 2002, pp. 155–162.
- [126] P. Reuter, I. Tobor, C. Schlick, and S. Dedieu, "Point-based modelling and rendering using radial basis functions," in *Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, 2003, pp. 111–118.
- [127] B. Wyvill and A. Guy, "The blob tree: Implicit modeling and VRML," in *Proceedings of International Conference From the Desktop to the Webtop: Virtual Environments on the Internet, WWW and Networks*, 1997, pp. 193–206.
- [128] F. M. Lai and A. Sourin, "Function-defined shape node for VRML," in *Eurographics 2002, Short Presentations*, 2002, pp. 207–215.
- [129] FreeWRL VRML/X3D browser,
<http://freewrl.sourceforge.net/>.
- [130] Xj3D project,
<http://www.xj3d.org/>.
- [131] A. Sourin, "Nanyang Technological University virtual campus," *IEEE Computers Graphics & Applications*, vol. 24, no. 6, pp. 6–8, 2004.

BIBLIOGRAPHY

- [132] A. Krall, “Efficient JavaVM just-in-time compilation,” in *PACT '98: Proceedings of the 1998 International Conference on Parallel Architectures and Compilation Techniques*. Washington, DC, USA: IEEE Computer Society, 1998, p. 205.
- [133] K. Levinski and A. Sourin, “Interactive function-based shape modelling,” *Computers & Graphics*, vol. 31, no. 1, 2007.
- [134] A. Stork, U. v. Lukas, and R. Schultz, “Enhancing a commercial 3D CAD system by CSCW. functionality for enabling co-operative modelling via WAN,” in *Proceedings of the 1998 ASME Design Engineering Technical Conferences*, September 1998.
- [135] R. Bidarra, E. van den Berg, and W. F. Bronsvoort, “A collaborative feature modeling system,” *Journal of Computing and Information Science in Engineering*, vol. 2, no. 3, pp. 192–198, 2002.
- [136] U. Dietrich, U. v. Lukas, and I. Morche, “Cooperative modeling with TOBACO,” in *Proceedings of the TeamCAD97 Workshop on Collaborative Design*, May 1997, pp. 115–122.
- [137] S. Chan, M. Wong, and V. Ng, “Collaborative solid modeling on the WWW,” in *Proceedings of the 1999 ACM symposium on Applied computing*. ACM Press, 1999, pp. 598–602.
- [138] A. Stork and U. Jasnoch, “A collaborative engineering environment,” in *Proceedings of the TeamCAD97 Workshop on Collaborative Design*, May 1997, pp. 25–33.
- [139] C. Foster, Y. Shapirshteyn, C. Cera, and W. Regli, “Muti-user modeling of NURBS-based objects,” in *Proceedings of ASME Design Engineering Technical Conferences & Computers and Information in Engineering Conference*, 2001.

BIBLIOGRAPHY

- [140] Y. Ouyang, M. Tang, J. Lin, and J. Dong, “WSCAD: a collaborative CAD system based on Web service,” in *Proceedings of 8th International Conference on Computer Supported Cooperative Work in Design, Volume I*, 2004, pp. 272–277.
- [141] A. Prakash and H. S. Shim, “DistView: support for building efficient collaborative applications using replicated objects,” in *Proceedings of the 1994 ACM conference on Computer supported cooperative work*. ACM Press, 1994, pp. 153–164.
- [142] L. G. Silveira Jr and S.-T. Wu, “An object-oriented groupware framework for developing collaborative 3D-modelers,” in *Proceedings of Third Eurographics Workshop on Parallel Graphics & Visualisation*, T. P. e Jamie Painter, Ed., September 2000, pp. 103–114.
- [143] C.-Y. Kim, N. Kim, Y. Kim, S.-H. Kang, and P. O’Grady, “Internet-based concurrent engineering: An interactive 3D system with markup,” in *Proceedings of the 1998 ASME Computers in Engineering Conference*, September 1998.
- [144] D. Li and R. Muntz, “COCA: collaborative objects coordination architecture,” in *Proceedings of the 1998 ACM conference on Computer supported cooperative work*. ACM Press, 1998, pp. 179–188.
- [145] J. Y. Lee, S. B. Han, H. Kim, and S. B. Park, “Network-centric feature-based modeling,” in *Proceedings of the 7th Pacific Conference on Computer Graphics and Applications*. IEEE Computer Society, 1999, p. 280.
- [146] R. Schmidt, B. Wyvill, M. C. Sousa, and J. A. Jorge, “ShapeShop: Sketch-based solid modeling with BlobTrees,” in *Proceedings of Eurographics Workshop on Sketch-Based Interfaces and Modeling*, 2005, pp. 53–62.

BIBLIOGRAPHY

- [147] G. Johansson and H. Carr, “Accelerating marching cubes with graphics hardware,” in *CASCON '06: Proceedings of the 2006 conference of the Center for Advanced Studies on Collaborative research*. New York, NY, USA: ACM Press, 2006, p. 39.
- [148] A. Chong, K. Levinski, and A. Sourin, “Interactive grid-based free-form shape modeling,” in *Proceedings of IEEE CCGrid 2006*, 2006, pp. 253–256.