Adaptive 3D Web-based environment for heterogeneous volume objects

Ali Abdallah

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

Bournemouth University

October 2019

Copyrights Statement

Abstract

The Internet was growing fast on the last decade. Interaction and visualisation became an essential feature online. The demand for online modelling and rendering in a real-time, adaptive and interactive manner exceeded the growth and development of the hardware resources including computational power and memories. Building up and accessing an instant 3D Web-based and plugin-free platform started to be a must in order to generate 3D volumes. Modelling and rendering complicated heterogeneous volumes using online applications requires good Internet bandwidth and high computational power. A large number of 3D modelling tools designed to create complicated models in an interactive manner are now available online, the problem of using such tools is that the user needs to acquire a certain level of modelling knowledge

In this work, we identify the problem, introduce the theoretical background and discuss the theory about Web-based modelling and rendering, including client-server approach, scenario optimization by solving constraint satisfaction problem, and complexity analysis. We address the challenges of designing, implementing and testing an online, Web-based, instant 3D modelling and rendering environment and we discuss some of its characteristics including adaptivity, platform independence, interactivity, and easy-to-use after presenting the theoretical part of implementing such an environment. We also introduce platform-independent modelling and rendering environment for complicated heterogeneous volumes with colour attributes based on client-server architecture. The work includes analysis and implementation for different rendering approaches suitable for different kind of users. We also discuss the performance of the proposed environment by comparing the rendering approaches. As an additional feature of our modelling system, we discuss aspects of securing the model transferring between client and the server.

Table of contents

6 Experiments, tests and results

List of Figures

List of Tables

Acknowledgements

I would like to express my gratitude to my supervisors: Prof. Alexander Pasko, Prof. Peter Comninos, Dr. Oleg Fryazinov and Dr. Valery Adzhiev for their encouragement and patience.

I would like to thank my wife, daughter, and son for their support and encouragement.

Declaration

I declare that this report has been written and created by myself. The report has not been submitted for any degree. The work in this report has been undertaken by myself except where otherwise stated.

Chapter 1

Introduction

1.1 Brief description of Web-based modelling

1.1.1 Web-based modelling concepts

Computer-Aided Design (CAD) helps engineers and designers to efficiently create, modify and analyse geometric shapes resembling real-life objects. It uses mathematical formulas to define a geometrical shape using computer software. Its revolution started early at the MIT in the mid-1960s as a result of early interest in computer graphics, which developed rapidly as computers became more affordable, and the application areas have gradually expanded [Sutherland, 2003].

Visualisation, exploration, and interaction with 3D models became an essential tool for different sectors. Hardware dedicated for computing graphics including graphics adapters and processing power is not growing fast and not meeting the rapid increase of demand from the graphics community, which realised that hardware started to be a bottleneck for the modelling process, and believed that relying on the slow hardware development to generate more complex 3D models is not convincing any more. Alternative solutions were required to overcome hardware bottleneck problems that is why researchers started to focus their research on proposing and developing new rendering algorithms to minimise rendering run-time and reduce hardware storage and processing power. [Wang *et al.,* 2016]

Many CAD systems exited as desktop modelling applications. Such systems are very efficient with high performance and precision. Thus, the need for collaboration and model sharing leads to take the CAD system online. Building up a Web-based collaborative 3D shape-modelling environment requires a deep investigation regarding three major aspects, namely Networking, Modelling and Visualisation. Problems surrounding these topics need to be discussed, resolved and solutions to be implemented. Different parts of the proposed environment need to communicate throughout a predefined network, which constitutes its backbone. 3D volumes need to be modelled using a modeller, which is considered as the core of the environment. Visualisation is the part where rendering and volume displaying takes place and is considered as the output of the environment

The pipeline of the creation of 3D models is composed of three phases: Modelling, Processing, and Rendering. Volume is usually represented using discrete geometric data or procedurally with continuous functions in the modelling phase, the preparation for rendering and visualisation takes place in the processing phase. Security concepts such as watermarking and data compression can be implemented in this phase. Rendering is the last stage, where volumes are displayed after setting up the sampling frequency the camera viewpoint, etc. [Vanhoey *et al*., 2017]. Enough information should be carefully collected about the real object to be modelled. The second step is determining all the necessary functions and parameters to be transformed into model data ready for rendering. Texturing is the last step where visualisation of the model takes place. Model interaction in virtual environments gives a certain level of reality especially when it comes to real-time movements accompanied by shadows reflected from the light source. [Popvski *et al*., 2014].

Web-browsers started to adapt and support 3D graphics thanks for the rapid development of the Graphics processor units (GPUs) with their ability to process 3D graphics. Different 3D graphics applications were introduced as visualisation solutions [Jung *et al*., 2012]. Modern computers, and even smaller devices such as tablets and smart-phones, are supplied with a suitable graphics adapter. Sons *et al*., focused on improving the functionality and performance of the browser when dealing with the GPU at the client side [Sons *et al*., 2010]. They designed an extension to the Web browser as an attempt to increase the performance. Web-based graphical environment resides externally on an application service provider's network and on the GPU, which is managed and controlled at the client-side using visualisation tools.

Online applications should be efficient and characterised by the instant response. In order to keep the environment up-to-date, an information regarding hardware, bandwidth and performance, and memory consumption should be collected. Scenario Optimisation usually provides efficient and accurate instant decision-making depending on the continuous monitoring and tracking for the emerging changes of hardware, connection and software resources. It helps in reducing the time cost by making use of the available resources [Mars and Hundt, 2009]. The scalability and availability of any environment, especially online one, depends on scenario optimisation, which constitutes an essential factor of success.

1.1.2 Web-based geometric representation

Online applications designed for interactive 3D modelling and rendering are developing rapidly, and the need for rendering complicated volumes is growing fast. Rendering interactive high-quality complex models require a big amount of data, wide network bandwidth and a vast amount of computational resources. Hoppe in 1977 introduced the Progressive Mesh (PM) which supports complex rendering and gradual transmission. The problems with PM lies in the huge number of triangle faces that constitute the complex model, which leads to a considerably big data volume up to tens of Megabytes, in addition to the edge collapse process, where big number of edge collapse operations take place and generate a big amount of data added to the original PM volume data be transmitted over the network. Moreover, every time the mesh is adjusted or updated, the request will be sent to the server side, then thousands of rendering operations and calculations should take place on the client-side which affects the instant visualisation process due to the huge number of requested information transmitted over the network and the massive amount of rendering operations that takes place in both the server and the client side. [Chen *et al.*, 2016]

Computational performance and high memory consumption started to become an issue in computer graphics, for those reasons, architectures started to deploy caches to reduce rendering time and extensive computational power. Caches help users when accessing the network, where geometric model access time could be high. Meshes can be compressed by eliminating repeated data from the original mesh. Mesh compression uses triangle strips and can be easily decoded using hardware graphics adapters which uses the mesh in sequential order, for that reason, ray tracing, which access the mesh in an arbitrary order, cannot access such representation, that is why ray-strips was proposed by researchers, to support ray tracing. [Deitrich *et al*., 2007].

Developing and accessing an easy to use 3D browser-based application based on wide-spread application program interface (API), and plugins free environment started to be a must inorder to produce 3D models. All modern browsers started to support WebGL, a JavaScript-based environment for working with computer graphics that is supported in most of the online modelling and visualisation systems. [Fisseler *et al*., 2017]. WebGL becomes a popular graphics API because some browsers, especially on mobile devices, have very limited support of commonly used extensions and limited on processing power and memory storage resources. Some techniques such as geometry compression were added to JavaScript libraries in-order to decrease 3D data files dedicated to complicated models and due to the fact that not all 3D file extensions could be supported on various platforms. Progressive object loading is another technique used to decrease the load on the limited hardware resources and allows different users to use the Web-based platform at the same time. [Fisseler *et al*., 2017].

The increasing demand for 3D models forces online Web-based applications to start using Web technologies such as X3D and WebGL. The excessive usage of artefacts leads to model distortion and negatively affect the rendering and visualisation processes especially when using numerous parameters. In order to evaluate the 3D model visual quality, successive snapshots of the 3D model were taken from different angles, these snapshots then construct the 2D image metrics which was used by researchers to compare the metrics with the original model. [Lavoue *et al*., 2016][Fisseler *et al*., 2017].

1.1.2.1 Client-server approach

Client-server architecture allows for the information exchange between a server and a client, where the server is a large-capacity computer, with a huge amount of information stored on it, and available for sharing with different clients. The clients are smaller computers that are used to perform local computing tasks. Storing big amount of information on servers or clients is a serious problem especially when clients with low connection speeds and/or hardware resources (storage space and memory) are requesting services with big data files. Transmission of 3D scenes still a major problem in spite of the huge efforts and researchers dedicated to resolve such kind of problems [Limper *et al.*, 2013]. Cloud computing including virtualisation and parallel computing could be one of the useful available solutions, where online big data storage and processing on demand can be applied [Wu *et al.*, 2014].

3D scenes are difficult to integrate into HTML files, and Web browsers are not designed to deal with large 3D data files with a vast amount of computational resources. One of the available options is to develop a partially interactive and immersive flash animation from a 3D scene [Rodrigues *et al.*, 2011]. Another option is to play back a reproduced “flat” 3D environment in the browser; such environment is composed of several pictures covering 360 degrees and stitching them together, all the above possible solutions use the Internet as a transmission media to transmit information (images and data) between the environment’s servers and the clients.

Even though the rapid development of the hardware graphics adapters supported by special memories, and despite the widening of the bandwidth, interactive instant visualisation still a challenge and is subjected to visual quality loss. The standard built-in tools outputting the model in the standard format of polygonal meshes have many drawbacks and limitations resulting in insufficient convenience of modelling (especially in a collaborative mode), and inefficiency of communications through the network and visualisation, thus being at odds with the current cutting edge of the technology in terms of hardware and networking abilities. More specifically, in terms of building proper client-server architecture for 3D Web-based systems, the development of the collaborative tools for modelling and visualisation requires flexible and efficient handling of hardware and software resources of each client to achieve the efficient workflow.

1.1.2.2 Rendering approaches and tools

The term rendering is defined as the process of visualising geometric models and transform it into a visible image. Rasterisation and ray-tracing are two of the most applied rendering algorithms. Rasterisation algorithm projects the rendered 3D objects onto the image plane after generating model dots and transforms them into pixels. Hardware components were developed to increase graphics computing performance and was supported with multiple and parallel rasterisation units. Nowadays, GPUs are supported with sub-processors designed to manipulate different kinds of shaders simultaneously rather than stacking them in the pipeline. In Ray-tracing algorithms, the observer shoots ray into the scene through the camera-aligned grid, the closest intersection with the model surface is then selected. Ray tracing can generate high-resolution images and is now available for real-time rendering. [Deitrich et al., 2007]

Many 3D modelling tools designed to create complicated models in an interactive way are now available online, the problem of using such tools is that the user needs to acquire a certain level of modelling knowledge. Most of these modelling systems use polygonal meshes that can be rendered quickly and using low hardware resources. Major weaknesses can be noticed in such systems such as the inability to edit or change parameters during the modelling process and the lack of tracking the construction history of the model during the modelling process. [Lindborg *et al*., 2017]. Some modelling systems use voxels to create complex models, such systems are very slow in the visualisation (rendering) process, and oblige users to store a huge amount of data on their machines.

Object geometry is widely represented by polygonal meshes, such representation is easy to understand and modify. Polygonal meshes representation is restricted to a limited number of predefined consecutive operation. One of the most well-known formats to represent and store 3D models is polygonal meshes using the boundary Representation (BRep) because it can be easily implemented on the Web browser and is supported by most of the application program interfaces (APIs). Polygonal meshes are described as an approximation of the mathematical model, and they still have issues including the loss of the real-model visual properties and precision in addition to the large consumption of memories and difficulties over distributed networks, not mentioning the limitation in accessing the construction history during the modelling process. Scalar fields is an alternative representation which can be used in web-based modelling. One of the main examples is Signed Distance Fields (SDF) representation which is a function-based representation gained a lot of attention because of its characteristics. Such representation proved to be very efficient in applying many operations such as blending, offsetting and others. It can be divided into two types exact and approximates, and both divisions are suitable for display, animation etc… When the field value matches the Euclidean distance, the scalar field is transformed into the distance field, and then we can differentiate between two cases: signed and unsigned distance fields. In the unsigned distance field, the sign is determined by the position of the target point whether it is inside or outside the geometry. In the signed distance field, the iso-value that constitutes the distance to the mesh is negative on the mesh surface and positive everywhere else. [Sanchez *et al*., 2015].

Some complex scenes, could be huge and need a lot of computational power, sometimes may need more than one CPU, Ram and GPU should be implemented on the same machine, or may use different hardware machines to compute one scene. The scene may be divided and distributed among the hardware resources to be rendered, and then combined to perform one rendered scene. In this way, we can make use of multiple hardware to render one massively complex scene. [Deitrich *et al*., 2007].

Complex geometrical models with massive data started to evolve due to the rapid change and evolution that is taking place in 3D modelling applications. Real-time rendering started to put a huge pressure on the hardware platforms that is why researchers started to pay attention on the level of details techniques and tools.

By using the suitable LOD, users during the rendering process started to pay attention to the distance between the viewing point and the viewed geometry in order to reduce the huge load on the hardware resources. As a result, choosing the correct LOD lead to low-cost hardware resources and faster rendering, and improvement in real-time visualisation. LOD is still incapable to run smoothly with massive data models transmitted over the Internet that restricts online real-time visualisation [Chen *et al.*, 2016].

Building up an interactive Web-based architecture, that supports wide range of different hardware and software configurations on both client and server side requires optimisation of modelling and rendering response time for viewing 3D models. Optimisation of the tools and techniques is also required at the rendering level. A certain level of model quality is required in order to meet accurate measurements [Fisseler *et al.*, 2017]. Internet speed with much bandwidth is essential for transmitting large 3D data files over the network and Internet for interaction and rendering. Large data files require the use of data streaming and compression techniques to speed up data transmission process over the Internet. Securing the model is a crucial issue in this aspect, encoding data at the server side could be done in almost no time, and the problem stays at the client side, where the decoding process takes place. The decoding time at the client side must be low to reduce the transmission time. [Fisseler *et al.*, 2017]. In this work we focus on designing, implementing and testing a 3D instant Web-based environment, supported with different rendering approaches suitable for different users. Our scope is to research and develop an adaptive, interactive, easy-to-use and platform independent environment.

1.1.2.3 Proposed approach (Implementation and novelty)

This work proposes a theoretical modelling framework to be designed and implemented, such a framework will include combining different modelling systems (Function and Boundary representations), and will be supported by different rendering approaches. The proposed modelling system will integrate two different systems and combine their characteristics in a hybrid one, thus allowing users to benefit from the characteristics of both modelling systems in one new system. Different rendering approaches will be implemented to support the hybrid modelling framework. Our implementation will include a new method that allows desktop models to be transformed into Web-based models. A model conversion tool will support the proposed framework to allow desktop users to convert their models into Web-based models without the need to acquire and knowledge about Web-scripting language. The proposed framework will also support the creation of heterogeneous volumes with different densities attributes, in our work we will focus on colouring attributes and will propose two different colouring approaches in Chapter 3 section (3.6).

1.1.3 Heterogeneous volumes and constructive modelling

The amount of 3-dimensional space needed for an object is called a volume. Volumes represented as point sets with both physical and photometric attributes such as colours and densities are called hyper-volumes [Schmitt *et al*., 2001]. Heterogeneous volumetric objects consist of a point set, where each point contains a set of various attributes referred to it. An attribute can be defined as a numerical measure of the physical property. Using-real valued functions, both geometry (point set) and attributes can be modelled separately. Researchers proposed many representations including boundary, functional and volumetric representation for heterogeneous objects. [Schmitt *et al*., 2001]. Gupta *et al*., defined heterogeneous objects as a set of composed objects with a different set of material and are divided into three categories, objects with multi-marital, objects composed of sub-objects, and clear material boundary objects [Gupta *et al*., 2010]. Kumar *et al*., described heterogeneous objects as objects made up of several non-uniform materials and different compositions with different microscopic characteristics [Kumar *et al*., 1998].

Constructive modelling is the process of decomposing one complex model into a set of simple or primitives, for example, spheres and cubes. These simplest volumes that have scalar attributes representing their colours, densities, temperatures etc., are considered as 3D point sets, where different operations such as difference, intersection, and union in addition to linear transformation functions are applied to them to reconstruct a new complex model forming a constructive tree, where leaves are primitive objects and nodes are the applied functions [Schmitt *et al*., 2004].

1.2 Problem statement

The demand of graphics computer is growing much faster than the hardware dedicated to computer graphics such as graphics adapters and processing powers. This issue leads to what is called hardware bottleneck problem in the modelling process because the demand for complex 3D graphics is much faster than the dedicated hardware. This problem makes it inconvenient to relay on slow hardware and directly affects both complex volume rendering, and real-time concept. To solve this issue, we will focus on proposing and developing different rendering approaches that suit different hardware resources to minimise rendering run-time and to assure instant modelling while making full use of the available hardware storage and processing power.

The principal purpose of this work is to propose the theoretical framework for the adaptive Web-based environment for heterogeneous volumetric objects as well as to develop the practical methods and tools to test the proposed theory. In this document, we explore the problems of building an adaptive 3D Web-based real-time modelling and rendering environment (WRMR). We mean by WRMR, an architecture or environment that is platform independent, reliable, and collaborative. It enables instant users' interaction for the purpose of 3D shape modelling and rendering in a collaborative manner. In this work, we focus on a particular emphasis on rendering aspects, thus defining the most efficient way of communication between the server and the clients depending on the available resources. To validate the proposed framework, we compared it to the available state-of-technology projects, using different research and technical dimensions. We tracked the progress of other similar available online systems, we made a detailed observation and performed data analysis to support the planning processes and to avoid replication of existing online systems. We also highlighted the benefits of the proposed framework in chapter 6, section 4, paragraphs 6.4.1 and 6.4.2.

Building up an instant adaptive environment is one thing, protecting it, and the generated 3D volume is another thing to consider. Delivering 3D models over the Web efficiently and securely is a must as the Internet became widely spread and being accessed by almost everybody. Some Web-based modelling applications are already developed, most of them are not platform independent and are designed to serve certain users with specific platforms, and not designed to serve all kinds of users with different platforms simultaneously.

One of the real challenges is to develop an easy to use 3D browser-based platform independent and plugins free environment supported by a friendly interface. This 3D modelling and rendering environment for heterogeneous volume objects should be accompanied by a solid multi-rendering technique engine wish is capable of visualising different complicated volumes based on different rendering approach. Choosing the most suitable rendering approach, even with 3D complex volumes that usually accompanied by massive data, will make use of the available hardware resources, and will improve rendering time and will assure instant interaction and modelling. The ability to pass parameters (colour, texture, etc…) to models in the modelling stages is a tough work especially when working with heterogeneous volumes with different densities.

1.2.1 Research aim

The aim of the research is to investigate, research, design and implement an instant adaptive 3D Web-based shape modelling and rendering environment that allows instant online modelling for heterogeneous volume objects supported by different parameters and attributes, to allow non-professional and professional users to use efficient modelling tools and rendering techniques, and to deliver online complicated 3D objects, as well as to assure model's security by applying suitable security defences on the proposed environment, in addition to data extraction and delivery which supports model security concept and assures delivering the model safely and efficiently over the Web.

1.2.2 Research questions

Based on the research problem, the following questions should be answered:

1) The demand from graphics community for complex models is increasing, and the development of hardware resources is not meeting their need, the rapid growth of the need for developing complex volumes with vast data is yet facing the limitations of the computation power and storage. Implementing different rendering approaches that require lower storage and processing power started to be in focus for the researchers. Recently they started to implement different rendering algorithms to escape hardware bottleneck problem. The question to be asked is: "How the available rendering algorithms and techniques are efficiently used in an open Web-based user-friendly environment in order to develop heterogeneous volumes while reducing their massive data and decreasing the rendering time and still maintaining high resolution and optimal usage?"

2) Online modelling is another point of interest, such an environment should be user-friendly and must be efficient and available for every single client with any hardware platform available on the Web. Clients with different hardware storage and computational power may exist which makes it hard to design an adaptive environment without noticing all the wide variety of users over the Web. In this work, we will answer the following question "What are the best practices and techniques needed to build up a Web-based environment and how to distinguish the proposed environment from what already exists online? "

3) Two major modelling approaches already exist for desktop and online modelling, Boundary representation (BRep) uses polygonal meshes, it is easy to be implemented and is supported by most of the application program interfaces. Function representation (FRep) is another approach for modelling, it uses continuous functions to represent a volume, it reduces its complexity and keeps its construction history while maintaining low

storage. Combining the two systems in one Hybrid representation modelling system (HRep) allows to benefit from the characteristics of the two systems together, therefore the following question should be answered: “What theoretical and practical methods should be applied to combine two different modelling systems in one hybrid web-based system which benefits from their characteristics.”

4) JavaScript is widely used language in Web-based applications. Desktop modelling uses some other languages such as C++ or HyperFun (HF). HF is a high-level programming language used to present (FRep) objects to do the modelling and generate volumes on desktops using the language compiler [Cartwright *et al*., 2005][Vilbrandt *et al*., 2004][Fryazinov *et al*., 2008]. Our goal is to take the modelling process from desktop level to the Web and make it easy for desktop users to use their own desktop modelling source code over the Web. To solve such an issue, the following question should be answered: “What are the best practices in moving 3D models from desktop to the Web, and what are the necessary tools and techniques needed to change a modelling language from desktop language to Web-based suitable to run on the Web-browsers?”

5) Online modelling applications are usually supported with one rendering technique and serve certain users with defined hardware resources. For example, Tinkercad is a Web-based application designed to generate simple models based on primitive objects [Tinkercad]. It uses Obj file format to save models, and is tailored for users with low hardware resources. Shapesmith is another Web-based application that supports by a limited number of modelling functions and needs more hardware resources. [Shapesmith].

   This issue leads to a limitation of the number of users and forces them to render their complex objects using the only available rendering technique. The question arises here as follows: What if a certain user with specific computational power needs to use the environment, and what if a user decided to visualise his model using another rendering approach?

Using the suitable rendering approach leads to low-cost hardware resources, and to faster rendering, and improvement in instant visualisation. To solve such an issue, the following question: "How to research and implement the right and efficient rendering approaches based on different scenarios?" should be answered.

6) All Web-based applications are open source applications, and thus are vulnerable to huge security risks and threats. Shapesmith for example, prevents normal users to access the application and asks them to send access requests using their personal emails in order to enforce security. Shapeways askes users to use login methods such as Gmail and Facebook, and it forces them to input a considerable amount of personal information including names, emails, addresses etc.., which could be an exhausting process in order to assure privacy, and online protection.

   Ensuring security over the proposed environment while keeping both online and open source concepts alive and well-functioning remains a big question to be answered. Extracting and delivering data files over the Web is another issue to be discussed, and the question about extracting and delivering 3D complex objects data over the Web efficiently, and integrate different models online smoothly and securely should be researched and solved. The question to be answered is "What is the best practices to support Web-based open-source 3D shape modelling environments with a certain level of online security?"

7) Modelling with heterogeneous volumes with attributes such as colour and texture is a must in order to take the volumes into another level. Passing parameters to volumes at the modelling phase stays a challenge especially when we do online modelling, such an issue should be investigated and solved. The question to be answered is: "How to implement online heterogeneous volumes with different densities and attributes?"

1.2.3 Contribution

The novelty of this work consists of:

1- Combining two different modeling approaches (BRep and FRep) in one unique approach (HRep) allowing users to make use of the characteristics of both approaches.

2- Dealing with HF users to transform their models from HF functions into JS functions suitable for Web browsers within a glance using an HFtoJS converter designed and implemented for this purpose. This approach allows desktop models to be available on the Web-browser in an interactive manner.

3- Dealing with heterogeneous volumes with attributes was a challenge especially when dealing with volumes with different densities and colours. This work solved the problem of dealing with heterogeneous volumes with attributes by introducing two different colouring approaches.

The work will address the following problems:

1) Modelling level:

   We address the heterogeneous volume objects modelling where the internal structure of the models must be visualised and presented with the surface. Both function and boundary representations are combined as part of the solution to this fundamental problem [Pasko *et al.*, 1995], Thus introducing a hybrid representation (HRep), which is a combination and integration between both function and boundary representation of models. Heterogeneous volume modelling with attributes such as colour and texture should be discussed and implemented.

2) Rendering Level:

Different rendering approaches will be implemented corresponding to different scenarios; these scenarios will use a special core engine running on the server-side of the environment to collect information about the client requesting the service. The rendering approaches will be distributed among the server and the client sides, depending on the obtained scenario.

3) Data extraction and delivery level:

3D data extraction and delivery using a well-known and predefined file formats allow online users to load, integrate, display and access 3D models efficiently.

This document will discuss and try to solve the problems listed below:

- Researching the most efficient way of communication between servers and clients for efficient modelling services.
- Collecting information about clients' resources and connectivity is a serious problem to be solved, monitoring and detecting the changes at the client side should be included in the solution.
- Clients with different hardware resources may be presented; different scenarios should be introduced to interact with the clients' needs.
- Scenario optimisation should be investigated, discussed and implemented to assure the quality of service and platform independence.
- Clients with different hardware resources and bandwidth require different rendering approaches to be discussed and implemented.
- Investigate and discuss the possibility of implementing the environment in the cloud for better interaction and storage.
- Protect online code and deliver access rights and permissions to authorised users. Implement security procedures to protect both, the server-based 3D models and the low-resolution and high-resolution heterogeneous volume objects rendered on the client-side.

- Implement special engines for 3D data extraction and storage to allow online 3D data delivery and integration.
- Discuss and implement a platform suitable for heterogeneous modelling that allow passing attributes (parameters) at the modelling level.

The main contributions of this work are following:

1) The work successfully combines two different modelling systems in one Hybrid system. Thus, Boundary representation (BRep), are combined and integrated into one Hybrid representation (HRep).

2) A HyperFun-to-JavaScript converter (HFtoJS) was designed and implemented to allow HyperFun (HF) users to use online modelling based on JavaScript. HFtoJS converts HF functions, codes and, libraries into JavaScript (JS) functions, codes, and libraries suitable for Web-based modelling. Using HFtoJS, we managed to solve a major problem of taking HyperFun to the Web and allow HF users to write HF functions and convert them to JS functions ready to be executed on the Web-browser.

3) Heterogeneous volumes with colour attributes were approached from two different perspectives, where two colouring methods were developed (leave and node colouring) to allow users to deal with heterogeneous volumes and to apply different colouring approaches. Those colouring approaches allow both external and internal volume colouring, where different coulours can be applied to the internal structure of the volume as well as its external structure.

Below a list of published contributions.

Problem	**Contribution**	**Publication /Outcome**
Web-based Modelling	Adaptive environment	ACHI 2014
Online 3D environment security	Obfuscation, Authentication/Authorization and data extraction	ICT and Societal Challenges 2017
Model exchange	Data extraction and delivery	FASSI 2017
Models with densities	Heterogeneous volumes with attributes	IEEE: ACIT 2018
Heterogeneous 3D volumes on the Net	Real-time heterogeneous volumes on the Web	Journal to be submitted soon.

Table 1: *list of contributions*

The list of publications is presented in Appendix A.

1.3 Objectives and outputs

The aim of this project is to explore different ways of building a Web-based interactive architecture for a modelling system based on hybrid representation, with different types of adaptations to the clients' needs with particular consideration of rendering techniques. After surveying related works, we describe an adaptive 3D shape modelling environment of a Web-based system, considering in detail a pure client-server one, and then an adaptive one. We will try to identify four of the most probable scenarios to support decision-making using client-server communications. After that, we will use different rendering approach (Marching Cubes, WebGL, Server rendering using C++ and Sign Distance Field) depending on the client infrastructure and try to introduce models with attributes. Finally, we present the experiments and discuss the results as well as some practical recommendations presenting the advantages and disadvantages of the tested techniques.

As an output, the document will put into test four different rendering approaches, and try to apply them to simple and complicated heterogeneous volumes, and to compare the performance of the rendering approaches regarding GPU power needed and time taken. The document will summarise the result obtained in a table showing the time taken for each rendered model using one of the three different rendering approaches taking place on the client-side; it also compares the efficiency of each approach and presents the results using different charts. After considering the time and the GPU power for each technique, the environment will be able to determine what kind of service has to be delivered to the client.

1.4 Summary

In this work, we introduce a new framework for modelling and rendering environment. The proposed framework combines two different modelling systems (BRep and FRep) in a hybrid one HRep, we discuss and put into test four different rendering approaches, we also propose a system to convert desktop functions into Web-based function suitable for online modelling and rendering, and we deal with heterogeneous modelling with attributes while focusing on volumes with different densities and colours using two different colouring approaches implemented to serve the proposed environment.

The research allows users to take benefit from the characteristics of both BRep and FRep in a hybrid representation HRep. Using HRep, users can use functions to represent their models, take advantage of the constriction history of the modelling process, transform their functions to polygonal meshes for rendering, transmit models over the Web using low space data files, and others.

Another benefit of this work, is the ability to take the modelling process into the Web by transforming desktop modelling functions mainly HyperFun functions into Web-based JavaScript functions suitable for online modelling and rendering. This process allows HF users to transform their models into JS models suitable for the Web-browser in an interactive manner.

Heterogeneous volumes with attributes allow users to deal with volumes with different densities, and benefit from applying different colours attributes for both the internal and the external structure of the volume.

1.5 Report structure

This document is organised as follows:

Chapter II will introduce a general overview of the related work related to online modelling and rendering and will discuss some of the researchers' work. A brief introduction to Web-based modelling will be presented and will include relevant research areas and the available non-ideal solutions. The modelling theory will be presented and will focus on the collaborative shape modelling environments in addition to the client-server architecture. A general overview of the available rendering approaches will be presented. We will discuss some aspects of securing online applications and some available 3D geometric models and modelling systems.

Chapter III is the theoretical part; it includes the theory behind the proposed online 3D environment. We start by defining the 3D environment and move to discuss different representations for modelling, we first discuss the Boundary and the Function representations and later we propose the Hybrid representation including the forward-backward transformation. The document will also discuss client-server architecture and adaptive environment concepts. Four different rendering techniques will be presented and discussed, these rendering techniques are as follows: Marching cubes, WebGL, OpenGL C++, and ray-marching using Sign Distance Fields. The complexity of the used algorithms to implement different rendering approaches will be discussed and compared for performance. Modelling with attributes will be also discussed and implemented.

In chapter IV we will cover the engineering and the implementation phases of the project. We start with the platform implementation including the core engine, then we implement some security concepts including authentication and authorisation, code encryption and decryption (obfuscation) and mesh delivery.

Data extraction and delivery will be implemented after and we will focus on the user-server interaction while developing the front end-user interface.

Chapter V will discuss the implementation of our work including the four rendering approaches in addition to the implementation of heterogamous volumes with attributes. The chapter will include a discussion about important tasks and modules. We perform volumes' implementation, testing, and comparison then we summarise the work, present a conclusion, and discuss the limitation and propose some future work.

Chapter VI will be dedicated to experiments as we are going to put into test different rendering approaches, and compare them according to different constraints (availability, bandwidth, and performance). Delivering heterogeneous 3D data models will be discussed and implemented and then compared according to the used rendering techniques

Chapter VII is the finale chapter where we present the conclusion of our work, in addition to future work.

Chapter 2

Literature review

Creating 3D models and scenes can be done using nowadays modern 3D technologies and modelling tools which are available online for public use. Complex models with internal material properties and the ability to define or change certain parameters during the modelling process usually need a certain level of knowledge. Tools that allow user interaction and support the changing parameters process are available online but with considerable limitations, such as the lack of tracing the model development history and the inability to access some operations [Lindborg *et al.*, 2017]. In the literature review, we will introduce some related terms and notations in the background section, then we present some of the commonly used graphical representations including polygonal meshes and voxels. Boundary and Function Representations are also introduced and highlighted in addition to signed distance fields and the HyperFun language. The non-ideal solution section discusses the interactive systems and tools, cloud technologies, distributed environments, collaborative models, and modelling frameworks and pipelines. The client-server architecture section comes next to present what was done on the client-server level using the Web as a network to exchange 3D data. Rendering tools are highlighted next, Marching Cubes, WebGL, ray casting, OpenGL are different approaches for rendering. In the security section, we highlight the watermarking concept, in addition to code obfuscation and mesh compression. And finally, we presented some real modelling systems available online.

2.1 Geometric representations and background

In this section we discuss geometric principles and formats that are used in CAD. We will also outline and define the terms which will be used in the thesis.

In computer graphics, 3D models are commonly represented using polygonal meshes. In spite of the restriction of the number of operations on the geometric object, meshes allow full understanding and modification of the 3D models and scalar fields can fill up the gaps and allow the manipulation with a huge number of additional restricted operations. Polygonal meshes have many problems such as loss of visual properties and precession for real models, they consume a large amount of memory, still face difficulties over networks and have no construction history [Sanchez *et al*., 2015]. High-resolution meshes can be generated using modelling shapes through part-based models and labelled objects are essential. This technique lacks the ability of any modification since it relies on gathering and combining parts retrieved from the database. Soltani *et al.*, extracted models from multi-view depth maps using generative models, the obtained 2D images are then used to render 3D novel and detailed objects [Soltani *et al*., 207].

Modern modelling systems deal with wide range of different geometric representations. One of these representations is polygonal meshes using Boundary Representation (BRep) which consists of set of vertices and the information on their connectivity into set of connected polygons. This representation is an approximation of the real models because of discrete nature of the polygonal data [Kobbelt et al., 2000]. Bounding the edges of the faces and vertices introduce us to the concept of Boundary representation (BRep). Solid modelling is considered as an advanced way of 3D modelling through which solid parts of the 3D volume can be represented using wire frames.

Another alternative representation to BRep is scalar fields representation. Geometric representation with scalar fields is based on a function that maps from point coordinates to scalar values for each point in 3D space. An example of representation with continuous scalar fields is Function Representation (FRep), where the mapping is done by a continuous function which defines the geometry of the object. FRep allows to efficiently describe *the exact shape of the volume object without using high computational resources [Pasko et al.*, 1995]. A special high-level programming language called HyperFun (HF) was introduced to present FRep objects in order to do the modelling and to generate volumes after rendering, HF is a desktop environment and it uses C++ or Java language compiler [Cartwright *et al*., 2005]. Signed Distance fields (SDF) is another well-known modelling representation and is defined by Canelhas *et al*., SDF is a surface representation where a 3D volume is transformed and represented as a scalar field where the field is negative inside the geometry, zero at the boundaries, and positive elsewhere [Canelhas *et al*., 2016].

Another discrete representation of the volume objects is by using voxels. Voxels are set of small boxes of the same size; in some way it is an extension of pixels into 3D space. Voxelisation is a process of using voxels to represent 3D volumes [Laine, 2013]. Volumetric convolutional networks are defined as deep networks used to present 3D volumes [Wu *et al*., 2015]. 3D shapes can be represented using voxels that can be used to build up complex models but with certain limitation such as high consumption of memory. Using volumetric convolutional networks, 3D shapes can be represented based on novel Voxelisation. 3D models can also be generated using 2D depth maps or silhouettes, where 3D models can be generated from a multi-view presentation. In comparison, the multi-view technique produces much higher resolution models than voxels. Soltani *et al*., concluded that detailed real-world objects can be generated from the multi-view representation [Soltani *et al*., 2017]. Great interest was shown from both designers and 3D CAD users in conceptual design.

Rendering with Marching Cubes (MC) is one of the most applied rendering techniques in 3D modelling, it iterates over a grid of cubes to generate a polygonal mesh [Lorensen and Cline, 1987]. OpenGL is a set of libraries suitable for 3D graphics based on C++ compiler [Rodrigues and Robinson, 2009]. WebGL is defined as an application program interface for low-level 3D graphics which is independent of any platform over the Web. It enables Web applications to take advantage of 3D graphics hardware acceleration in a standard way [Evans *et al*., 2014], [Khronos]. Ray casting iterates over pixels to produce 3D volumes, where rays are shot from a camera to the pixel to detect the intersection of the ray with that pixel [Congote *et al*., 2011].

Lindborg *et al.*, used graph-based representation, which allows users to change parameters and request changes any time they need. In their work, they used sphere tracing, which is a direct rendering technique that uses geometric distance, to render implicit surfaces. Their system was based on the transformation from a scene graph into functions stored inside the shader for fast representation. They developed a node editor to generate a scene by connecting the provided nodes to the root node. The resulted node tree is converted into shader code which constitutes the geometry and then compiled using sphere tracing algorithm inside the fragment shader to be rendered and visualised. [Lindborg *et al.*, 2017].

2.2 Relevant shape modelling systems and tools

Relevant Web-based 3D modelling application must be able to handle a visualisation-specific representation that consists of registered and merged points, surface, and even volume data as well as the corresponding meta-information in order to provide important features for visualisation. The information is re-formatted in a structured way after data acquisition and before filtering the data accordingly. After that, all data sets (scalar values and vectors) are transformed into a certain representation and ready for rendering step [Jung *et al.*, 2012]. In this section we will present the related work related to different modelling systems and the relevant geometric representations. We start by presenting different types of geometric representation including polygonal meshes and voxels, then we move to different modelling approaches including BRep, FRep and SDF and we present some different tools and techniques designed for modelling purposes including HyperFun and Constructive solid geometry tools.

2.2.1 Shape and volume modelling systems.

Three main shape representations were discussed including BRep, FRep and SDF. BRep uses polygonal meshes in the form of inter-related faces and vertices to represent a model, FRep is a mathematical representation of the model and SDF uses signed distances between points and surfaces.

Most of the collaborative modelling tools adopt BRep. It mainly uses polygonal meshes (set of faces and vertices) to represent a model in a 3D virtual world and can use parametric surfaces inside [Asghari, 2013].

One of the most known rendering techniques for implicit surfaces using FRep is polygonisation with polygon rendering, where three-dimensional objects (made up of flat polygons) are used to approximate shapes. In their work Sanchez *et al.*, dealt with signed Euclidean distance as a continuous real function that can evaluate polygonal meshes very efficiently. They took the convolution product of the distance field with the kernel in order to smooth it. [Sanchez *et al.*, 2015]. SDF uses a function to represent object geometry. In contrast to the discrete data structure that represents inaccurate and sometimes deformed model and is represented by polygonal meshes, SDF functions assure the continuous representation by returning the Euclidean distance to an object and can be used for real-time parameterised and interactive modelling. [Lindborg *et al.*, 2017]. SDF is represented by a procedural tree which contains both: primitive objects or models as leaves having distance property and operations as nodes that assure real-time modelling. The authors introduced a modelling prototype that uses heterogeneous objects that can add, change or even connect nodes together, which is implemented in C++. They used sphere tracing based on direct rendering to assure quick visualisation using direct hardware resources implemented in OpenGL. The goal of their prototype is to attain parameterisation over an interactive environment. Lindborg et al., succeeded in proposing and developing heterogeneous volumes that can be edited at the modelling and rendering levels using sphere tracing. The benefits of the work include the ability to deal with parametrised heterogeneous volumes using SDF in an interactive manner. Their work has some drawbacks such as restriction to set of predefined nodes and leaves and the inability to visually represent moving objects

The Web-ready modelling systems, which are based on FRep are quite rare and generally based either on HyperFun language [Cartwright *et al.*, 2005] or BlobTree structure [Galbraith *et al.*, 2004]. Cartwright *et al.*, used HyperFun (HF) to implement shape modelling applications based on the Web and rely on open system architecture [Cartwright *et al.*, 2005]. HF depends on experimental systems to achieve interaction and relies on Java applet for Web collaboration [Fayolle *et al.*, 2005], FVRML/FX3D [Liu and Sourin, 2006], XISL [Parulek *et al.*, 2006], Hyperfox plug-in for Firefox [Vilbrandt *et al.*, 2010] and Websockets [Grasberger *et al.*, 2013]. HF files can be of small sizes even for complex geometric objects [Vilbrandt *et al.*, 2004]. The size of the HF file is not dependent on accuracy or mathematical precision, which allows for efficient implementation of a client-server modelling system. The main issue with HF is that it is not designed to work on the Web-browser and it needs a mediator to load HF models (after rendering) on the Web-browser using Java applets. HF can deal with parameterised hyper-volume objects based on their Function Representations. Parameters including colour and texture can be passed to the function in the modelling phase before rendering. Objects are defined as functions followed by necessary attributes represented by scalar functions. HF function is evaluated during the construction of the FRep tree, where operations constitute the nodes and the simple volumes occupied the leaves [Schmitt *et al.*, 2004]. Lindborg *et al.,* used SDF representation to represent their models [Lindborg *et al.*, 2017].

Another technique using constructive modelling approach is the Constructive Solid Geometry (CSG), where shapes (geometry) and attributes are combined in an organised manner and are presented as 3D models using an array of voxels and scalar fields (attributes) [Schmitt *et al.*, 2004]. Schmitt *et al.,* discussed another approach called interactive volume sculpting, where the sculpting process focuses on the deformed model by adding-removing material. They mentioned that when a complex model consists of both normal and deformed parts, it can be broken apart into two parts: primitives and non-primitives. The primitives can be processed with CSG or Function Representation, while the deformed parts can be sculpted and then added to the CGS tree. CSG suffers from some limitations such as the limited set of primitives and operations, this makes it difficult to work with complex models when decomposing these models into primitive volumes. The alternative is using a constructive FRep tree which can easily deal with complex models [Schmitt *et al.*, 2001].

As a conclusion for what was mentioned before, BRep using polygonal meshes has certain limitations including lack of model precision, large memory consumption, and no construction history. Voxels face large memory consumption when dealing with complex models. HyperFun is not designed to be implemented on the browser and needs a mediator to load HF models in the browser, the mediator makes it impossible to allow model interaction. Major problems and difficulties will arise when using the above-mentioned approaches over the Web and those problems may go more explicit when dealing with heterogeneous volume objects. A huge amount of data to be stored especially for complex 3D objects makes the process of saving information, parameters, and operations of 3D objects into a low cost and small size file is a real challenge.

2.3 Non-ideal Web-based 3D modelling solutions

Many CAD systems are available as Web-based applications, either as server-based or cloud-based solutions. Such systems are designed for online modelling and rendering but with certain limitations. Some lack interactivity, others use pre-defined and limited tools for modelling and rendering, and few are designed to serve a considerable number of different users with different platforms. In this section we will highlight the growth of available CAD systems and their performance in the cloud.

Co-CAD, a “multi-user CAD prototype system” [Gisi and Sacchi, 1994] was limited to interaction between two people. In order to avoid conflicts between the designers, sort of coordination policy was proposed by Klein [Klein, 1991]. The cPAD system [Shyamsundar and Gadh, 2001] was developed to support Web-based collaborative object design with assembly features that allow designers to perform real-time geometric modifications. COCADCAM was presented as a CAD/CAM system with collaborative concepts that contains geometry editing, processing and modelling [Ramani *et al.*, 2003], [Santos and Strok, 2004]. In order to apply interactive detection and follow-up of improvement model during download, Schwartz *et al.,* discussed a “novel progressive streaming approach” and employed it for the huge BTF data set [Schwartz *et al*., 2013]. Using a similar method, and by using image geometries, X3DOM compressed and transmitted lightweight geometry [Jung *et al.*, 2012]. The Web-based environment can be adopted by distributed design environments because it is easy to use, with a common interface, universal standards and available for real-time access [Qin and Wright, 2004]. Nowadays, different frameworks can support Web-based systems in a collaborative way. Distributed object modelling environment (DOME) was first introduced by MIT, and was designed to deal with modelling problems [Abrahamson *et al*., 2000].

Technologies such as visualisation and parallel computing can be empowered with cloud computing, especially when we deal with rendering on demand, efficiency and availability. Users can make use of the high computational power such as the high speed and the huge storage available over the Web, when using cloud computing for real-time 3D rendering. Wu *et al*., discussed the concept of collecting information about clients in real-time. They also discussed the distribution, share and access of big data files of 3D models on the cloud [Wu *et al*., 2014].

2.4 Approaches to collaborative shape and volume modelling pipeline

A 3D modelling framework presented by Tsai *et al.*, is based on three major stages, 3D point clouds segmentation, bare-bones model generation, and surface structure addition [Tsai *et al.,* 2017]. The cloud-based segmentation and processing were based on a special available algorithm used to segment the data into different groups discussed by Grilli *et al.,* [Grilli *et al*., 2017]. The bare-bone volume model phase composes the components of the model together by identifying the necessary parameters of stage one. To form the bare-bone, a CSG algorithm was used to connect the components of the model together to form the skeleton structure. Stage three represents the decoration phase, where additional surface structures were added after filtering the point clouds, and finally, the mesh objects were added into the model [Tsai *et al*., 2017]. As a result, Tsai *et al.,* developed a frame to build up heritage buildings by combining mesh objects with polyhedral models using the above three phases. Their results showed that data points were reduced and detailed elements were preserved in addition to offering a multi-level detailed object.

Branko and Leitao aimed to employ the features of different tools to cover all the stages of the design process using single-script approach. They designed a procedure to describe the required analysis as well as the model itself by proposing an algorithmic approach without affecting the workflow. Their methodology includes model analysis as well as the model description and is subdivided into three different phases: CAD modelling, Building Information Modelling (BIM) and analysis integration consequently [Branko and Leitao, 2017]. Phase one uses programming tools for modelling and visualising 3D objects using Rosetta, which is a programming environment suitable for the 3D design processes [Leitao and Lopes, 2011]. In phase two, the geometric model is then visualised using BIM software. The BIM model is then supported by decorative elements as a final step [Branko and Leitao, 2017].

Vanhoey *et al*., focused on the 3D object creation pipeline which starts by modelling through processing and ends up with rendering. They discussed in their work the level of distortion a geometric model may acquire under the influence of light-material interaction. They concluded the strong influence of light and material on the object distortion. As a result, they proposed a simple metric that helps in regulating geometry processing by measuring distortions [Vanhoey *et al*., 2017]. Cahyawiajya and Supriana proposed a three stages solution based on model generation stage proceded by an image processing stage and followed by model rendering stage. [Cahyawiajya and Supriana, 2015].

2.5 Web-services and interaction using client-server architecture

In a client-server architecture, polygonal meshes can be generated from the server, and transmitted over the network as vertices and faces to be rendered and visualised on the client's browser [Limper at al., 2013] [Evans *et al*., 2014] [Lavoue *et al*., 2013]. Web-based 3D modelling systems can make use of the services available online using what is known as a Service-Oriented Architecture

(SOA). SOA uses many Web services such as XML and JSON as Web-service protocols and plays a major role in controlling and managing the collaboration between different running applications and platforms [Jung *et al.*, 2012], [Erl, 2005].

Some researchers used a stream of images to view the model from different angles, and they displayed them on the browser [Rodrigues *et al*., 2011]. Koller *et al*., transfer images to clients and include a number of active defence methods to guard against 3D reconstruction attack by providing an interesting proposal to the protection system with a remote rendering service [Koller *et al*., 2004].

The main question to be asked in Web-based modelling is the client-server interactions, i.e., the information that the server sends to the client to render the model. In the case of volume data, the amount of information transmitted between a server and a client can be significantly large and the client often requires installing an additional software tool. Thus, in X3D format 3D objects supporting point, surface and volume primitives are described, but additional plug-ins for the browser needs to be installed.

2.6 Rendering concepts

Different rendering approaches are available and are supported with different rendering tools. In this section we discuss the rendering approaches for volume models represented by representations other than BRep including Marching Cubes (MC), ray casting, volume rendering and image streaming.

Marching cubes is used to create a triangle mesh using a simple computer graphics algorithm which iterates or marches over a grid of cubes and extracts a polygonal mesh of an iso-surface from a three-dimensional scalar field (voxels) [Lorensen and Cline, 1987]. Rendering with ray casting or ray tracing is another widespread technique, which iterates over pixels rather than objects and produces better effects such as shadowing, transparency, texturing and reflections. Rays are shot from a camera to the pixel; the ray intersects with existing objects, and the closest intersection is selected. Volume rendering is another technique, where

RGBA volumes are formed and continuous functions are reconstructed out of discrete data set to be projected onto the 2D plane. Web-based direct volume rendering with ray-casting was presented in [Congote *et al*., 2011], the purpose of discussing direct volume rendering was to investigate in medical imaging as well as in radar meteorology. Reflectancd information can be obtained by using a certain framework based on WebGL.

WebGL was introduced as an extension so that JavaScript can interface with OpenGL [Rodrigues and Robinson, 2009]. It is using OpenGL libraries, and is defined entirely within an HTML document and loaded into a Web browser. 3D graphics are generated by OpenGL engine (libraries) and being accessed by the browser using WebGL [Rodrigues *et al*., 2011]. WebGL interface has the ability to communicate, to access and to control the client's graphics hardware GPU. It allows direct access to the Graphics Process Unit (GPU) hardware, where a new set of objects and functions were established to support the HTML specifications for 3D graphics. The advantage of this application program interface API is the accelerated adoption, where no plug-ins installation is needed any more. It is well-known that WebGL is slow when rendering a large amount of 3D data because of the slow operation speed of the script especially when rendering complex objects with a huge amount of data. Another issue is that WebGL causes network traffic load and delay in transmitting repeated 3D data. In order to reduce rendering transmission and computational complexity, Kang and Lee proposed a way to organise 3D modelling data into unit-based data using tile-based rendering as an attempt to increase WebGL rendering speed [Kang and Lee, 2017]. Most 3D services need special plug-ins in-order to display 3D models on Web browsers, plug-ins come with some advantages such as good quality of service with Web sites using the same plug-ins and great functionality. However, we can also recognise some disadvantages of the plug-ins such as applications' duplication when using different browsers, sudden emerging errors, cross platforms limitation, and security threats, where malware could be embedded with the plug-in [Kang and Lee, 2017].

HTML5 was created in 2014 as an alternative to standard HTML 4.01 to support multimedia including audio and video contents in the browser without the need for any additional plug-ins. WebGL is one of the features of HTML5 standard allowing to bring 3D graphics into Web applications. Kang and Lee used HTML5 and WebGL to display 3D data related to city modelling and tried to discover the requirements and the limitations by attempting to a use bin-packing algorithm, which allows modelling data to tile in units [Kang and Lee, 2017].

One of the useful ways to render 3D models at the server side is by using C++ compiler using OpenGL libraries which maintains full control at the CPU level and produces 3D objects rendered at high resolution and considerable speed. OpenGL has become the graphics engine of choice due to its sheer power and ease of integration. It supports 3D models and provides functionality to 3D modelling applications. Since Web browsers do not understand OpenGL, JavaScript is used as a wrapper in order to interface with the browser and translate OpenGL graphics outputs into statements that the Web browser understands and displays [Rodrigues and Robinson, 2009]. Adopting a strip triangulation algorithm using Matlab based on C++, Wang *et al.,* introduced a lightweight design to connect the outlines in neighbouring sections and construct the inner surface of voids. They used 32 GB of random access memory and a core i7-3770 processor to generate hollowing models in an optimised manner [Wang *et al.*, 2017].

2.7 Web-based 3D modelling security

Securing 3D models and protecting them from theft and piracy were put into research. The majority of methods are for BRep data that is why we focused on three different security concepts including digital watermarking, code obfuscation and mesh compression.

Digital watermarking was one of the techniques designed to hide information of the 3D model. Only authorised users can detect watermarking as an attempt to enforce authentication and copyright [Xiao and Shih, 2010]. Digital watermarking can provide a sort of protection for digital files that constitutes 3D models, and can enforce for copyrights [Zeki *et al*., 2013]. Lin and Wu proposed in 2002 a way of extracting watermark rendering conditions transformation [Lin *and Wu*, 2012]. Spectral watermarking framework accompanied by a blind two-way parametric digital data was introduced by Lui *et al.,*. They used a method called optical fringe projection encoding method to encode 3D models into encrypted 2D referenced fringes using random keys [Li *et al*., 2012].

Code obfuscation, which is a way to encrypt the source code of 3D models, was put into focus by Satoshi [Satoshi, 2000]. Canetti *et al.* proved that when using cryptography, access into black boxes turned to be weak, when compared with 3D functions. Canetti mentioned insecure protocols when using hash functions [Canetti *et al*., 2004]. Barak *et al.,* confirmed the immunity of some obfuscatory against algebraic attacks [Barak *et al*., 2014]. Watermarking still has its limitation when dealing with large number of vertices, it also has no defined representation [Chou *et al*., 2009].

Mesh compression technique started to be a point of interest due to the huge demand of transmitting 3D Web-based scenes over the Internet, in addition the wide range of different platforms and devises with different GPU powers and memories [Limper *et al*., 2013], [Peng *et al*., 2005], [Maglo *et al*., 2012]. Hoppe introduced Progressive Meshes (PM), which is a successive and continuous format of polygonal meshes and have the capability to display 3D objects with great details by making use of the client's resources [Hoppe,1996], [Lavoue *et al*., 2013]. PM can minimise the size of the 3D scene and speed up the transmission over the Internet and can do compression associated with colour and texture attributes without losing the details [Lee *et al.*, 2012]. Lee *et al*., discussed the above-mentioned issues and proposed an algorithm that allows for quick compression of 3D data and generates a (P3DW) binary compressed file. The curvature prediction was another 3D data compression techniques presented by Maglo *et al*., The authors discussed a wavelet formulation method in order to improve the efficiency of the rate distortion (R-D). They also presented a quantisation method as an attempt to increase the compression rate [Malgo *et al.*, 2012]. Dong *et al*., suggested CRYPTON, an intra-origin data control system, which can monitor 3D data at the client browser [Dong *et al.*, 2013]. Kang and Lee discussed the slow rendering problem, when using high-resolution textures, and worked on improving the rendering speed using tile units to represent 3D data [Kang and Lee, 2017].

Developing an independent 3D system with collaborative platform and able to share information over the Web-browser may face many threats. One of these threats is packet sniffing over the network that allows sniffers to watch and access classified information such as account names and passwords. Another kind of threats is password attacks, stealing passwords may allow the attackers to gain access over the environment and allow them to create an illegal back door to the environment [Yu *et al.*, 2003].

2.8 Relevant Web-based 3D modelling systems

Web browsers are now capable of handling 3D objects in an interactive way allowing 3D manipulation techniques such as object scaling, rotation, shadowing and translation in its development phase. Using a browser plug-in, Google released its O3D API in April 2009. Opera released their 3D canvas two years earlier in 2007. Web browsers need access to clients' hardware, mainly GPU and memory, in order to handle 3D graphics. OpenGL, which was developed by Silicon Graphics Inc, in 1992, is an Application Program Interface (API), with high performance and portability so-called "Device independent". Applications depending on OpenGL are designed to run on all platforms, providing a set of useful graphics tools and functions

3D search engines have recently been deployed on the Web [Chen *et al.*, 2003], [Corney *et al.*, 2002], [Varnic, 2003]. Several systems allow users to download 3D models stored in their databases. Google SketchUp provides a small but well-defined set of modelling tools that are easy to learn [Googleskechup]. A general shape modelling description language for procedural models, the Generative Modelling Language (GML), was introduced in order to provide modelling operations for polygonal shapes, which are tessellated on-the-fly adaptively [Havemann, 2005]. Berndt *et al.*, noted that "GML allows for very compact model descriptions, especially useful in a Web context" [Berndt *et al*., 2005]. Its suitability for 3D modelling by non-expert users was also assessed [Greth *et al*., 2005]. Krottmaier *et al*., introduced PROBADO as a co-operative digital library project funded by the German Science Foundation (Deutche Forschungs Gemeinschaft, DFG) [Krottmaier *et al*., 2007]. Its main purpose is to integrate generalised documents, in particular, music and 3D models of architectural buildings, into the workflow of existing libraries [Krottmaier *et al*., 2007].

Recently, many browser-based 3D modelling programs started to appear on the Web. Developers started to integrate primitive 3D Web-enabled applications; such that applications enable ordinary users create their own object models. 3DTin is a browser-based modelling tool developed by Jayesh Salvi [Blog.3dtin1]. It feels like a virtual Lego application that provides a non-technical user with primitive tools and introduces him/her to the possibility to create simple 3D designs. In 3DTin, models are built from cubes with a few primary colours, primitive tools exist to convert the cubes to rounded–formed shapes with a considerable level of smoothness and this leads to the increase of the polygon count and slows down the application. In order to allow people to design more interesting and detailed objects, Salvi added new shapes including cylinders, cones, wedges, spheres, and several variations of these shapes. Another 3D solid modelling software in the browser is Tinkercad [Tinkercad], which requires WebGL. It combines the simplicity of the Web with the basics of the 3D-design process. It has a user-friendly Web service that offers variously shaped 3D brushes and tools for easy drawing and allows for 3D designs aimed at 3D printing. Tinkercad models can be exported in the STL format and shared with different users providing a good and efficient way to share projects. Shapesmith is a recently issued 3D object browser-based modelling system [Shapesmith], which aims to be a powerful parametric application that is open and extensible. Using Shapesmith, one can export designed models for printing, even though it is still in an early stage of development. ShapeJS, a generative geometric modelling language, developed by Shapeways, is based on JavaScript and 8-bit voxel models [Shapeways]. Applied Shapes Limited uses geometric modelling for the aim of jewellery design, 3D printing, education and healthcare, and accessibility, computer games and other applications [Applied Shapes]. The company relies on Uformia's Uformit (abbreviation for You Form It), which allows designers to customise their designs and have full control in choosing and editing their models [Uformit]. 3D CAD tools, such as AutoCAD 360 and Fusion 360, are available on the cloud and allow users to draw different geometric shapes with the ability to save and edit objects using a Web-browser. Such tools use cloud-based rendering and delivering. NX 3D software developed by Siemens relies on a virtual desktop infrastructure to conduct 3D graphics using a privet cloud environment. Another collaborative Web-based and cloud-based CAD system is OnShape, which allows different users to edit and modify the same model at the same time [OnShape]. 3D software packages and services allow users to design and perform 2D and 3D models rendering using Web-browser cloud-based environments in a remote collaborative manner. Cloud-based environments support most of these software packages and services either fully or partially [Wu *et al*., 2016].

2.9 Conclusion

In this chapter, we presented some related terms and notations of modelling and rendering, and we discussed some of the works done by researches. We highlighted the problems, difficulties, and limitations of their work. We researched the concepts of designing, developing and implementing Web-based modelling and rendering solutions for heterogeneous volumes, and we discussed all the limitations of being interactive, real-time, platform independent and collaborative.

After highlighting the problem of different modelling and rendering approaches, we can figure out that the problem become even more obvious when the Web is involved. Heterogeneous volumes objects make the problem even more complex and explicit. That is why a new representation is needed, and this work will try to research, design, implement and put into test such a new representation needed for the new CAD tool.

Chapter 3

Theoretical aspects of 3D environment

Online 3D modelling and rendering methods and tools are developing rapidly, and both users and developers started to show a great interest in online modelling due to the available solutions that allow rapid rendering of high-quality complex models. Recent online applications allow rapid response and a convincing level of interactivity. Users of these applications can interact with 3D models including editing, saving, and exchanging of models over the Web. Modelling environments use the clients' hardware to do the rendering and visualisation, therefore, users should be provided with considerable hardware resources as well as a reasonable internet bandwidth. Most of the modern browsers allow handling of 3D graphics without extra plug-ins. JavaScript became the most common and well-known online scripting language that uses WebGL, which provides direct access to the GPU for rendering. Online 3D environments need to be platform independent and are able to adapt to users with different software configurations. Some users may not have enough hardware resources or may not be able to go rendering in the browser that is why different rendering techniques should be implemented taking into consideration different scenarios. In order to exchange models and information between different 3D environments and users, client-server architecture should be designed and implemented to allow data exchange and to decide the rendering side (server or client). The client-server architecture is divided into three main parts: Networking, Modelling, and Rendering. The server-side allows server based rendering with high definition models using the server hardware power, but with major limitation, the model will not be sent to the client, it sends a stream of images of the rendered model to the client instead. The client-side allows instant rendering using the client's computational power, with the ability to instant model editing, saving and updating. One example of the server-side approach is a server using C++ with OpenGL while a client uses conventional web technologies (HTML and JavaScript) to display stream of images. Since the 3D environment is an online and open source based architecture, it needs a certain level of security to assure certain level model protection. In addition to data extraction and exchange between different environments that allows model's exchange.

In this chapter, we aim to present the theoretical problems of our work, will discuss and analyse them from different perspectives. Dealing with different rendering approaches leads to different scenarios that are based on different constraints. Scenario optimisation analysis should be performed to identify the problem, generate the scenario optimisation tree, and present the theoretical part of the scenario optimisation using the constraint satisfaction problem (CSP) analysis which is a mathematical way used to formulate the problem, reveals the optimal solution and satisfies certain constraints using predefined mathematical questions. The problem of using FRep and SDF will be discussed and will present the theoretical analysis for both. Different rendering approaches will be used based on the scenario optimisation process. Complexity analysis will take place to measure the performance and make comparisons between different rendering approaches. Heterogeneous volume objects are 3D complex models with different materials and densities distributed in a non-uniform way among the model and can handle different attributes such as colours, textures, etc. [Schmitt, 2008] [While, 2006]. The problem of using heterogeneous volumes with attributes will be discussed; the problem will be approached from two different points (node and leaf colouring approaches) accompanied by the theoretical analysis for both simple and complicated heterogeneous volume colouring.

3.1 Characteristics of 3D Web-based real-time modelling and rendering environment (WRMR)

Being online, 3D environments should possess some of the basic characteristics necessary to their survival and continuity. Since such environments completely relay on the Web-browser which is not designed to handle 3D volume objects by nature, this makes them vulnerable to different issues such as additivity, interactivity and real-time processing. This work is concerned with a 3D modelling and rendering environment. The proposed environment is a Web-based environment, it makes use of the rapid development of the Web-browser and their ability to access the graphics adapter GPU at the client-side. The 3D Web-based real-time modelling and rendering environment (WRMR) aims to be an adaptive one, which can adapt to all users' needs and requirements, it is able to deal with different modelling needs and rendering requirements, and is designed to serve a wide range of users with different hardware and platforms. 3D volume modelling and rendering take place in real-time, and it depends on the clients' computational power to do the rendering (Fig 1).

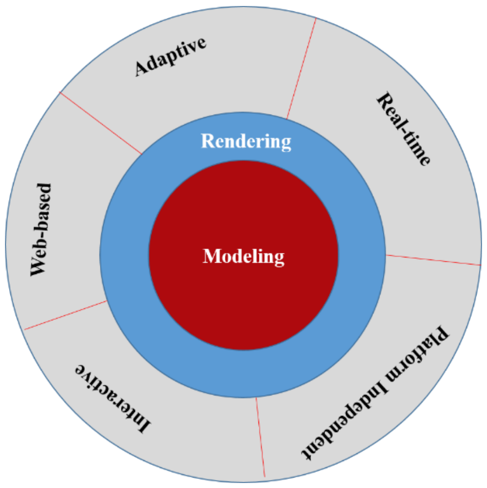


Figure 1: *Web-based, adaptive, real-time, platform independent and interactive are the main characteristic of the proposed 3D Web-based environment*

3.1.1 Function, Boundary, and Hybrid representations

As it was discussed above in 2.1, different representations are available in 3D modelling. BRep, which is an approximation for the exact model, is one of these representations which uses polygonal meshes in the form of inter-related faces and vertices. The mesh can be rendered to generate 3D volumes to be displayed on the Web using modern browsers. One of the major issues when dealing with BRep is the absence of the model's construction history during the modelling process. Another issue is the huge size of the mesh file when dealing with complex volumes. Mesh files may face serious problems when transferring them over the internet. The size of HF files are relatively small compared to BRep files, while representing shapes with similar complexity. [Cartwright *et al*., 2005], [Vilbrandt *et al*., 2004].

It is clear that both BRep and FRep have advantages and problems on modelling and rendering stages. Thus, BRep is widely supported by graphics hardware and therefore suitable for rendering purposes, while FRep allows greater flexibility on the modelling stage yet not being properly supported on the rendering stage. To combine advantages of both representations a hybrid approach should be used. In the context of this work it is not just a hybrid representation, but a hybrid environment, that can adapt to both BRep and FRep modelling and has the ability to use different rendering approaches without the need to change any of its characteristics. As a result it allows to use their features in order to implement a collaborative and adaptive environment which is able to deal with different modelling needs and rendering requirements. (Fig 2).

$$\begin{tikzcd} & \text{HRep} \arrow[dl] \arrow[dr] & \\ \text{FRep} \arrow[rr, leftrightarrow] & & \text{BRep} \end{tikzcd}$$

Figure 2*: Hybrid representation, a combination of both BRep and FRep*

Hybrid representation (HRep) embraces both BRep and FRep by employing the idea that the model can be stored in either representation and converted to another if necessary. In this context HRep is a combination of these two representations in a way that the appropriate representation is chosen given the current task, such as modelling or rendering.

FRep can be converted to BRep by using surface extraction methods, such as polygonization [Lorensen and Cline, 1987]. This process takes a continuous function which defines a scalar value in every point in the domain and results the finite set of triangles which approximates the surface of the zero-level set of this function with a certain precision. The efficiency of this method decreases with increased precision and the extraction process [Pantaleoni, 2011]. Alternatively, a voxelisation process for a scalar field can be used, i.e. extraction of the voxel set that belong to the interior of FRep object followed by extraction of BRep from the voxel grid (Fig 3).

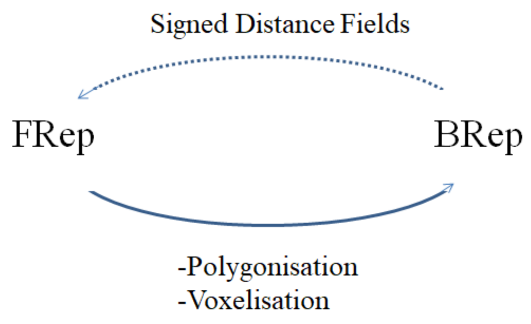


Figure 3: *Forward and backward conversion between FRep and BRep*

The conversion from BRep to FRep can be done with a signed distance field (SDF). Here for a polygonal mesh (boundary in BRep) the function is defined as a signed distance function, where the value of this function is the Euclidean distance to the polygonal mesh multiplied by the sign [Sanchez *et al*., 2012]. The distance can be found by finding the closest element in the polygonal mesh, while the sign is defined differently with respect to the interior of the polygonal mesh. Note that we have to clearly distinguish interior points and exterior points, i.e. the boundary should be well-defined. Combining FRep and BRep together allows to benefit from both representations for modelling and rendering purposes. This two-way integration (Fig. 4) allows transferring any model from FRep to polygonal representation (the mesh), and vice-versa.

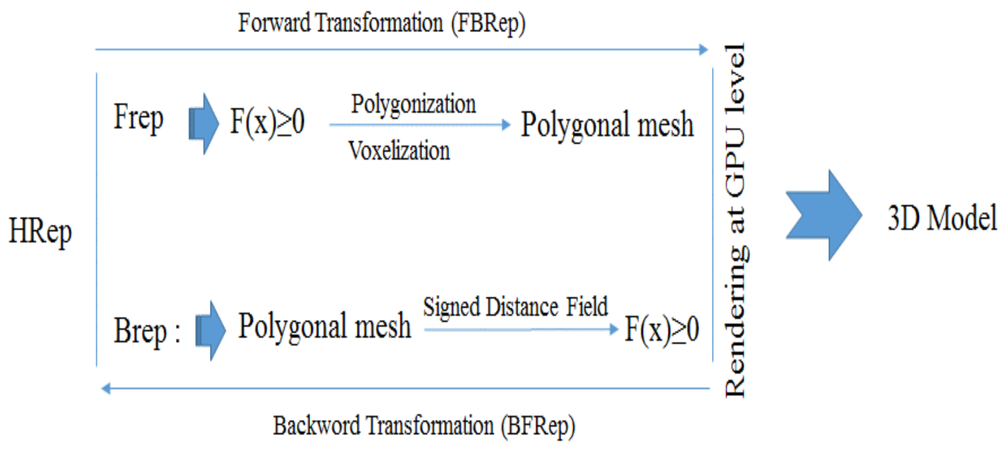


Figure 4: *HRep bi-directional conversion*

The main encouragement of using Hybrid represented (HRep) models is the reduced complexity of the models, this allows us to get rid of all the problems resulted from dealing with large 3D data files and huge resources. However, the major drawback of these models is difficulties of controlling and implementing them inside a Web browser because non-polygonal objects are not supported inside the browsers for rendering 3D scenes [Fryazinov *et al*., 2008]. Modern browsers use polygonal mesh to load and display 3D volume objects, they also allow the user to edit the scenes (translate, rotate, scale) inside the Web browser in an interactive manner.

The proposed environment may acquire a hybrid representation or HRep. It includes all the characteristics of both representations and can deal with the internal structure of the volumes; it can also track and save the constructive tree of the model during the modelling process.

3.2 Pure client-server architecture

Client/server architecture is a network environment for information exchange and services share between servers from one side and among clients from the other side. The clients are smaller computers that users use to perform their computer-based responsibilities. The client-server architecture reduces the multiple iterations of a single file and allows computers to access the same files and applications by turning the organisation into a centralised point.

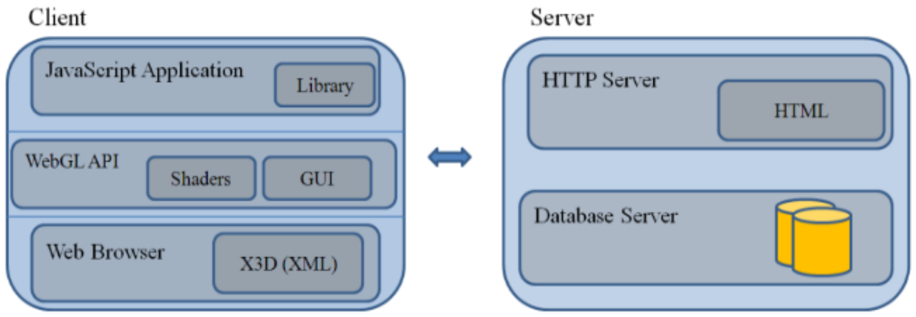


Figure 5: *Client-server architecture [Ramani et al., 2003]*

A fat client is a computer with high functionality and is totally independent of the main server, while a thin client is totally dependent on a server's applications. Even though the fat client can perform many functions without being connected, periodic connection to the server is still needed. In contrast, the thin client totally depends on the server and performs little processing each time it needs to treat or validate input data. A decision should be made in order to design a client-server application. This decision directly affects both clients and servers, and the task distribution as well. Decisions on adaptive system design are usually forced by clients’ infrastructures and hardware characteristics.

Web applications started to take advantage of the client-server architecture using Web technologies for dynamic content and interactive environment (Fig. 5) [Ramani *et al.*, 2003]. 3D immersive environment started to be adapted to the Web browsers and 3D documents started to be generated and retrieved for 3D objects exchange. Graphical capability to create 3D virtual environments is specific to Web3D applications. Web3D is an interactive web-based 3D environment which has the ability to integrate different technologies and tools including programming languages, file formats and protocols [Chourio *et al.*, 2011]. Combining the above elements in one Web3D application is a challenge for both designers and developers; sometimes, additional software should be installed, in order to allow 3D Web-based applications to use their visualisation on the client's GPU. Since such applications are independent of the content of the hosting Web page they usually cause weak interaction with the elements of the page.

Web-based modelling depends on the performance of the client’s hardware, the Web also allows exchange information between clients and servers. Client-server architecture could be a good solution for Web-based modelling and rendering because users can make use of both the clients’ and the servers’ performance. Modelling complicated volumes may require high hardware performance which can be available on the server-side, rendering the model could be quick and efficient when using the clients' machine. Furthermore, securing the model and the environment needs a client-server architecture to assure authentication and authorization in addition to data protection and control.

WRMR, is an online Web-based and platform-independent environment, which allows users to create, edit and delete heterogeneous volume objects in a collaborative manner. WRMR is a system developed and implemented on a local personal server for the purposes of testing heterogeneous volumes with attributes using different rendering approaches. In the context of this environment, the server and the client are defined as following. The server is responsible for performing most of the complicated computational functionalities including solid modelling, rendering, and constraint solving tasks so that the client's size can be reduced. The user can perform tasks on the geometry and can import primitive geometrical objects stored in the database, or on a local storage device. The client-side may have a copy of the main CAD model, loaded to the Web browser using JavaScript and WebGL techniques. The Web browser GUI loads basic tools from the server and is able to perform and generate 3D geometries using the user’s hardwae resources (GPU and local memory) (Fig. 6).

The system we discuss here is platform-independent from the client point of view. The client may have very small hardware resources to process 3D data, we consider the server to be responsible for performing most of the tasks using its powerful computational resources. When the 3D object is processed, rendered and images are generated, all the other operations in a shape modelling system based on HRep, including editing the functions and changing or adding primitives and tasks to them, can be done on the client-side using its low resources.

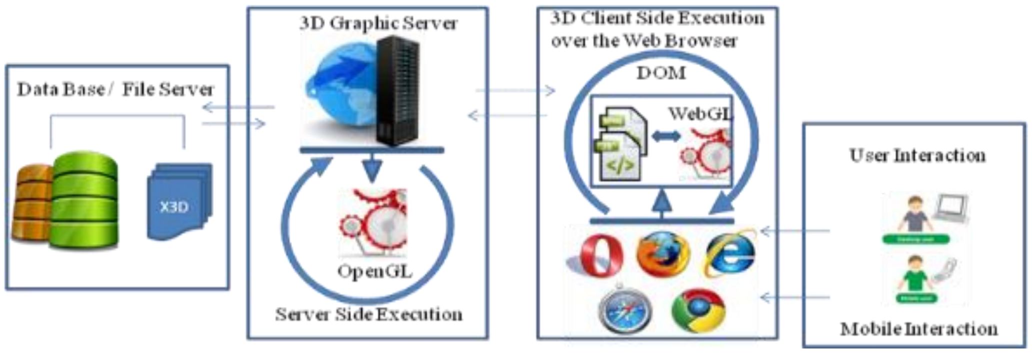


Figure 6: *Client-server 3D Web-Based modelling and rendering architecture [Ramani et al., 2003]*

Information is transformed from the server to the client in different forms, depending on the capabilities of the client, the data can be as WebGL texture objects, polygonal mesh or images. The work on the server and on the client is connected by a code written in JavaScript. We would like to stress that the proposed WRMR should be scalable, where a client can be anything, including desktop, mobile or a pad. In the case of mobile and pad, the server should be able to generate images after performing the rendering, instead of generating 3D data.

The proposed client-side architecture stores the files downloaded from the server, in the client-side cache as WebGL texture objects, and these textures are managed by JavaScript code (Fig. 6). Textures arriving from the server are encapsulated in the image object that can be passed to OpenGL without requiring any further processing that allows for achieving instant speeds. JavaScript plays the role of a director for the work happening in other parts of the browser and graphics driver. The client-side rendering is done by utilising X3DOM, which uses JavaScript with WebGL. WRMR will support both thin and fat clients and will be able to decide where to make the rendering (client or server-side) depending on the bandwidth, hardware capability and memory available at the client (Figure. 7).

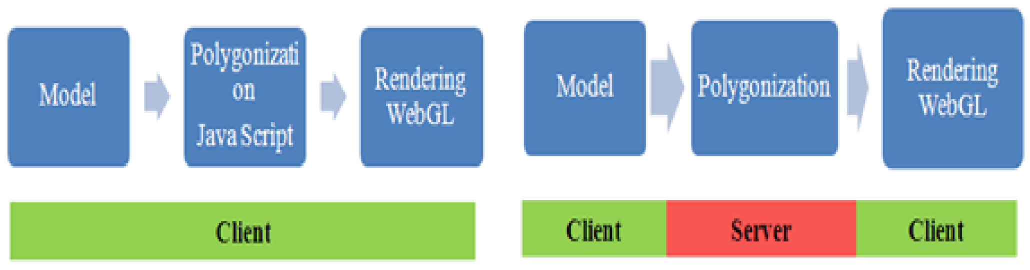


Figure 7: *Proposed client-side architecture (thin and fat clients)*

The server-side of the whole system will be able to do the rendering job for a model loaded from the primitive objects stored in the server database or sent to the server from the client-side. The server performs iso-surface polygonisation, while rendering will be held by client-side WebGL. The server-side is responsible for performing most of the computationally intensive functionalities, where solid modelling and constraint solving operations take place. The server is composed of a static file hosting that serves HTML, JavaScript, and CSS in addition to the Tile Server Web Application. Both mobile and desktop machines with different hardware and software computing potentials and 3D rendering capabilities are served in the same and equal manners. Using a hybrid approach, low-end mobile clients can be served by the servers (Fig. 8), while desktops are served using direct Web-based 3D rendering using X3DOM [Jung *et al*., 2012].

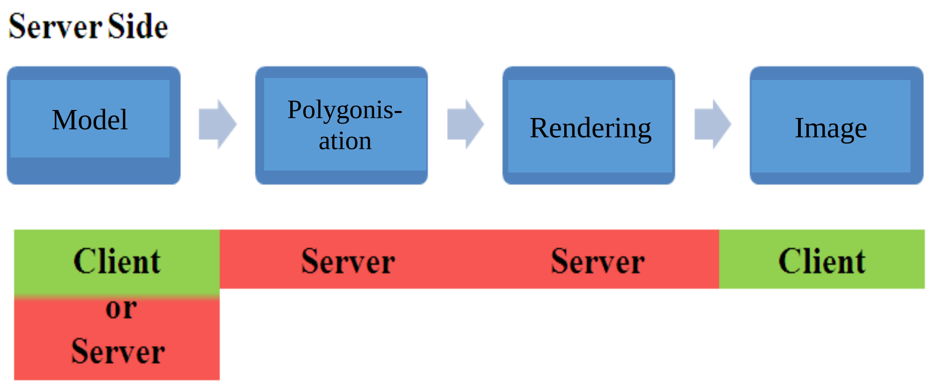


Figure 8: *Proposed server-side architecture with polygonisation and rendering*

3.3 Adaptive environment

The proposed WRMR will consider the bandwidth at the clients, the machine used (desktop or mobile), and the machine GPU and memory, and it will decide where to do the polygonisation, what type of rendering must be done and on what side (client or server). It will also decide what type of Hybrid representation (HRep) should go for (if necessary). The key point here is how to integrate the concept of the HRep with WebGL tools and techniques and with JavaScript engines.

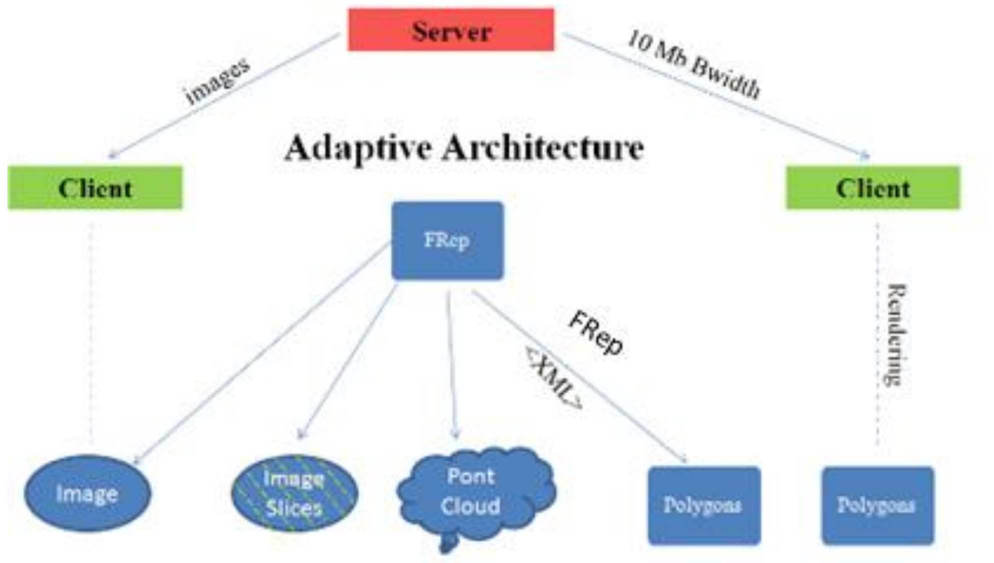


Figure 9: *HRep modelling architecture showing different modelling techniques*

WRMR will include objects processed by the server and delivered to the client as a stream of images; objects may be represented as a point cloud, polygons using <XML> for FRep or polygons generated by WebGL rendering techniques (Fig. 9). The hardware on the client-side can be very different. Different parameters of the client should be taken into account to choose the best possible way to deliver the rendered model from the server to the client. These parameters include the type of the client machine (desktop or mobile), CPU and GPU availability and power, amount of available memory, and the existence of the supported software (such as WebGL support in the browser). WRMR should take all the above-mentioned parameters into account, and choose the best possible way to deliver the model to the client.

The proposed environment is divided into three main parts. The networking part is responsible for determining the most efficient way of communication between the clients and WRMR, and for choosing the best scenario or group of scenarios among all possible available ones in order to achieve the best practice. The modelling section is the one responsible for holding all the modelling processes, it is also responsible for loading and saving 3D volume objects from and into the cloud, and/or the physical data storage in the form of 3D data files. The modelling section also assures the interaction and integration among different users. The visualisation part is responsible for visualising the objects generated from the modelling part, for that purpose, different rendering techniques are available. The best practice can be chosen from a wide range of available rendering techniques supported by WRMR (Fig. 10).

In the client-server architecture, it may happen that the client may have powerful hardware resources; in this case, the environment should transfer not the rendered objects but the model itself where the rendering takes place. Models generated from the server-side can be delivered as follows:

- Images obtained after direct rendering;
- Objects delivered as image slices (voxel array);
- Objects delivered as discrete data structures, for example, point clouds or polygonal meshes;

WRMR should be able to adapt, react and interact with clients according to their capabilities. It uses different scenarios depending on the clients' needs and available resources (Fig. 10). To make a decision regarding the appropriate scenario, WRMR collects information about a specific client's machine and retrieves data related to the available resources and bandwidth. The collected data or information is used to analyse the resources at the client-side, and by using scenario optimisation, the best-selected scenario or group of scenarios will be put into action (Fig. 11).

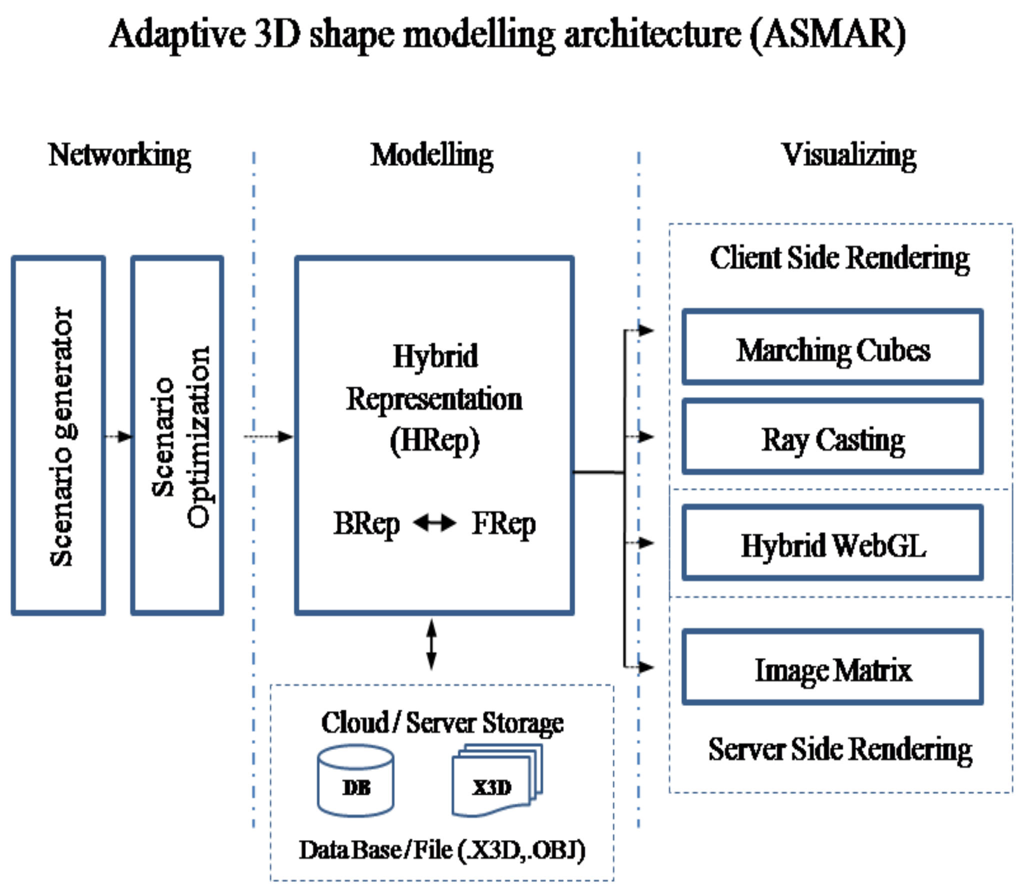
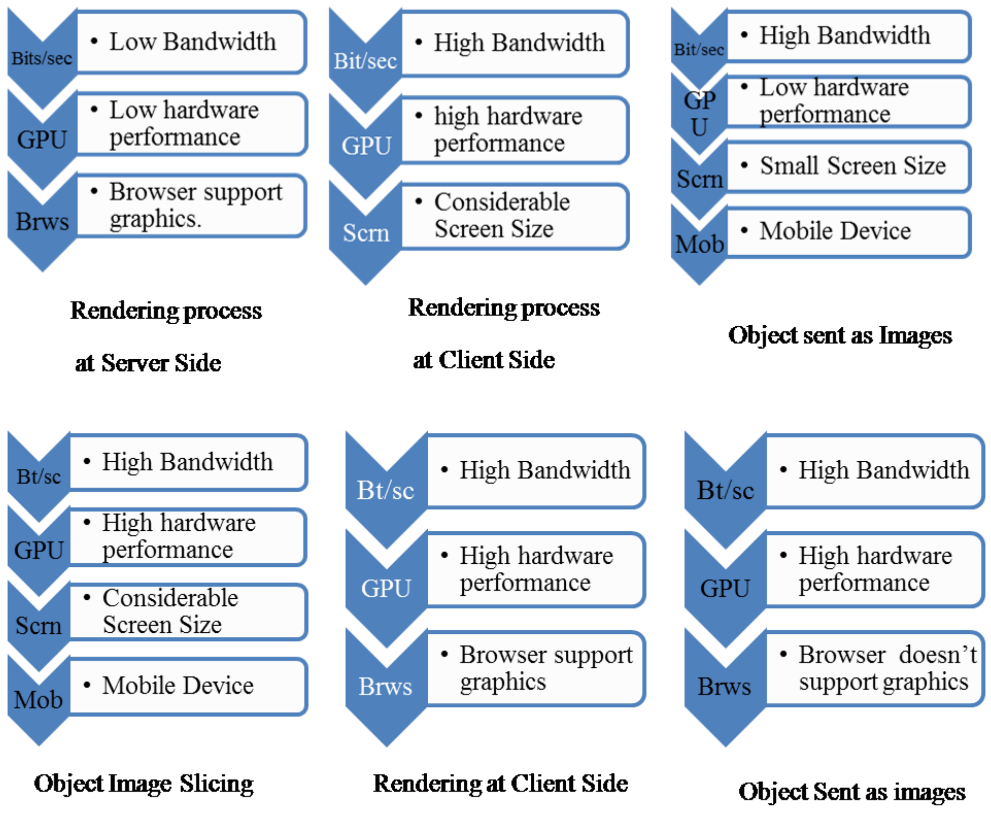


Figure 10: *Proposed adaptive shape modelling architecture (WRMR)*

3.4 Scenario Optimisation

We mean by scenario optimisation the process of finding the best solution for the problem based on different parameters or variables. The problem we are trying to solve or optimise is about choosing the correct rendering approach based on the changing input parameters, which are in our case the client bandwidth, the available computational power or hardware power, and the available browser parameters. Optimising the problem helps in choosing the right rendering approach based on the available parameters, and the ability to switch from one rendering approach to another when a change in the parameters is detected.



Bits/sec
• Low Bandwidth
GPU
• Low hardware performance
Brws
• Browser support graphics.
Rendering process at Server Side
Bit/sec
• High Bandwidth
GPU
• high hardware performance
Scrn
• Considerable Screen Size
Rendering process at Client Side
Bit/sec
• High Bandwidth
GPU
• Low hardware performance
Scrn
• Small Screen Size
Mob
• Mobile Device
Object sent as Images
Bt/sc
• High Bandwidth
GPU
• High hardware performance
Scrn
• Considerable Screen Size
Mob
• Mobile Device
Object Image Slicing
Bt/sc
• High Bandwidth
GPU
• High hardware performance
Brws
• Browser support graphics
Rendering at Client Side
Bt/sc
• High Bandwidth
GPU
• High hardware performance
Brws
• Browser doesn't support graphics
Object Sent as images

Figure 11: *Different types of scenarios and decision-making*

In practice, we identified four most common scenarios as follows:

1) The server detects a low bandwidth and low computational resources client; the model is rendered at the server and sent to the client as images.

2) A client with good bandwidth is detected, but its browser does not support 3D rendering; the server does the rendering from different angles and sends the result as a stream of images. The images are loaded to a slider, which allows the transformation of the object on the client.

3) A client with good bandwidth is detected and its browser supports 3D rendering; however, it has low computational resources. The polygonisation of the model is done on the server who sends the polygonal mesh to the client. The client uses its own resources to render the mesh.

4) A client with a high-performance machine is identified by the server; the complete model is sent to the client, where the rendering process takes place

using the client local resources. In this scenario, the server transfers the model data only.

In order to benefit from the identified scenarios, Scenario Optimisation (SO) can be applied to improve services delivered to the clients. SO provides online optimisation based on the continuous and online information collected from the clients' machines, and due to instant monitoring of the hardware performance, and internet connection speed to reduce time cost and make use of the available hardware resources [Mars and Hundt, 2009]. The main idea that stands behind SO is that rendering techniques can be optimised and improved to support a particular defined scenario. For that reason, SO framework can be designed and implemented to take advantage of instant hardware and bandwidth monitoring and to support WRMR and improve the rendering service.

Different scenarios are based on three different input parameters, these parameters determine what scenario to be used. Since it is a client-server based architecture, the bandwidth (BW in the Table 2) at the client-side plays a major role in determining what scenario to be issued, thus user bandwidth is considered the first parameter in the scenario issuing process. Both GPU and memory at the client-side are responsible for rendering and delivering the 3D volume objects, and constitute the Hardware parameter (HW). The last parameter is the available Web-browser (WB) at the client-side, this parameter is a major parameter in determining the rendering side, if the Web-browser allows 3D rendering and displaying then the rendering will take place at the client-side, else it will take place at the server-side. Table 2 contains the three above mentioned scenario parameters, each parameter is assigned one of two values, 1 for reasonable availability, and 0 for low availability or absence. The letter "P" represents the percentage of the occurrences of each parameter. The "output" represents the rendering scenario to take place and "P(out)" represents the percentage of the used rendering approach depending on the input parameters. Table 2 shows that each parameter has only two different values (True, False) or (1, 0), and the occurrence of each parameter is 50% for each value. Since we have three different parameters, then we can conclude the following number of scenarios

Scenario	BW	P(BW)	HW	P(HW)	WB	P(WB)	P(Scenario)	output	P(out)
1	1	50%	1	50%	1	50%	1/8	WebGL	1/8
2					0	50%	1/8	OpenGL	1/2
3			0	50%	1	50%	1/8	MC	1/4
4					0	50%	1/8	OpenGL	1/2
5	0	50%	1	50%	1	50%	1/8	SDF	1/8
6					0	50%	1/8	OpenGL	1/2
7			0	50%	1	50%	1/8	MC	1/4
8					0	50%	1/8	OpenGL	1/2

Table 2: *Scenario-based on three different parameters: Bandwidth, Hardware, and Web-browser*

$$n = v^p \tag{3.1}$$

$$n = 2^3 \tag{3.2}$$

Where n is the number of scenarios, v is the number of values (1, 0) for each parameter and p is the number of available parameters. Thus, the number of predicted scenarios (n) is 8.

Since we are working with four different rendering techniques, Table 2 shows that half of the scenarios go for one rendering technique, which is the server-based and using C++ and OpenGL. 1/4 of them uses the MC rendering techniques, and both WebGL and SDF are 1/8 each. (See descriptions for the four different rendering approaches in 2.6)

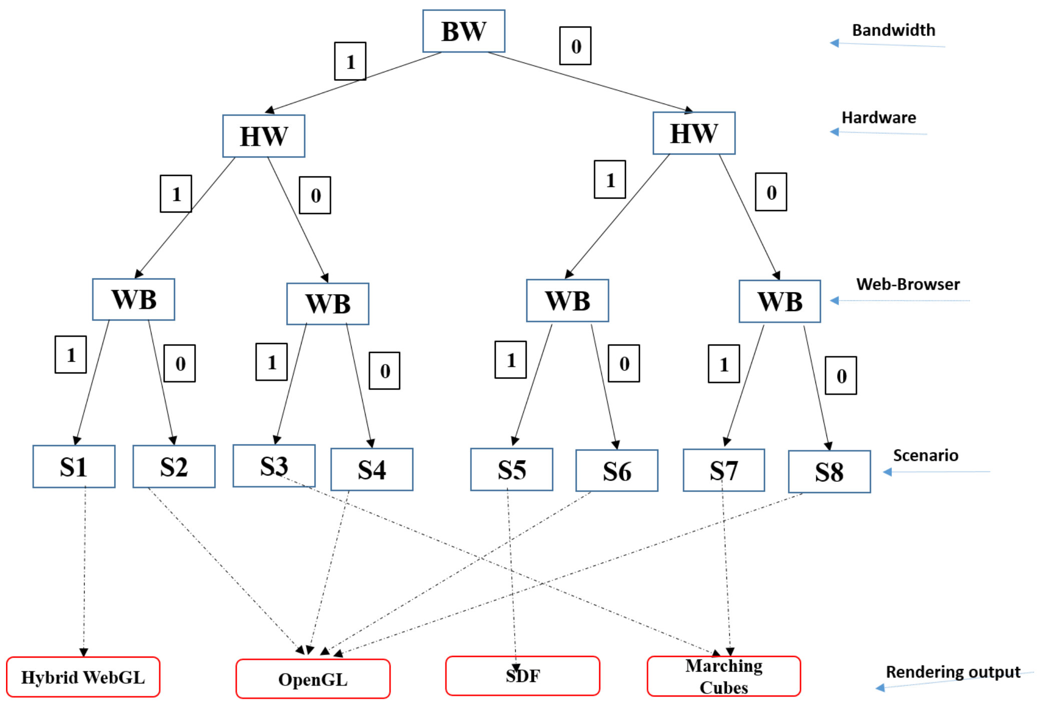


Figure 12: *Scenario decision tree showing the three different parameters (BW, HW, and WB), and the four resulting outputs (WebGL, OpenGL, SDF and MC)*

The above scenario (Table 2) can be represented as a decision binary tree where each node represents one of the three parameters (Bandwidth, Hardware and Web Browser), each node has two branches (1, 0), as a result, eight leaves, each leaf representing a rendering technique. From the decision tree, we can notice that the leaves are of four values, 1/2 of them are for OpenGL, 1/4 for MC, 1/8 for WebGL and 1/8 for SDF (Fig 12). Starting with the BW parameter, if the bandwidth is good enough then move left else move right, the second node is the HW, if the available hardware is powerful enough then move left, else move right, and the last parameter is the WB, if the browser allows rendering, then move left else move right.

The following recursive algorithm can be applied to implement the decision tree in Fig. 12:

- *Step1: An empty leaf as a start*
- *Step2: Split data by selecting a feature*
- *Check for every split*
    - *Step 3: no more splitting? Make a decision*

    - *Step 4: else, use recursion on the split by going back to step 2*

- *End*

The decision tree in Fig. 12 showed the following:

1- If WB is false (Does not support rendering), then the decision is Server-based rendering using OpenGL.

2- If HW is false (low hardware) and WB is true, then the decision is MC rendering at the client-side.

3- Else the decision is divided into between WebGL an SDF depending on the result of BW, if true WebGL, else SDF.

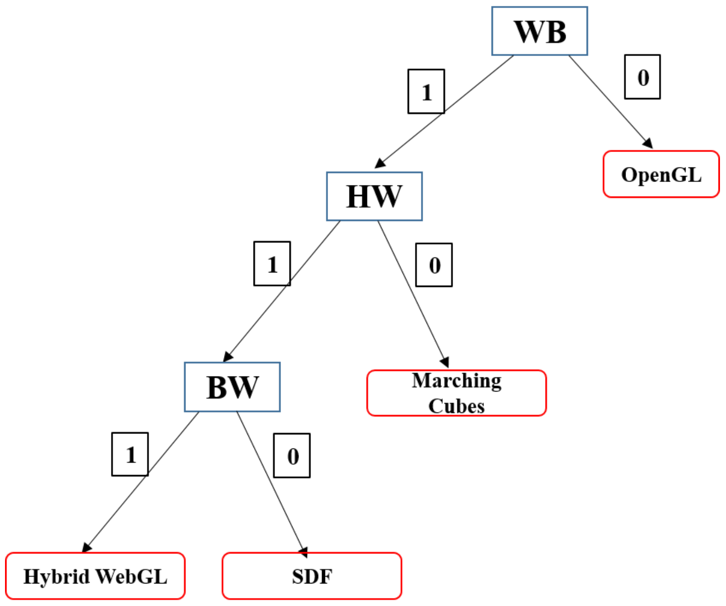


Figure 13: *Optimised decision tree showing the minimum number of nodes and leaves*

The conclusion above allows us to optimise the decision tree as shown in Fig. 13 by applying the followings:

1- Starts with the WB as a node, then follows the HW and ends up with the BW.

2- Use the same recursive algorithm discussed earlier.

In order to solve the above decision tree, constraint satisfaction problem (CSP) will be applied to formulate the problem and reveals the optimal solution for each state. CSP is defined by a set of variables, set of constraints and set of domains. Each variable (X_1, X_2… $X_{n,}$) is represented by X_i, each constraint (C_1, C_2… C_m) is represented by C_j and represents a subset of the set of variables with a valid combination of values by giving values to some or all the available variables. The domain D_k represents the possible values of the variable X_i:

$$csp = < X , D , C > \tag{3.3}$$

$$X, a\ set\ of\ varibales, \{X_1 , \ldots , X_n\} \tag{3.4}$$

$$D, a\ set\ of\ domains\ for\ each\ x.\ \{D_1, \ldots , D_n\} \tag{3.5}$$

$$D_i = \{V_1, \ldots , V_n\}\ for\ X_i \tag{3.6}$$

$$C, a\ set\ of\ constraints\ , C_i = < scope, relationship > \tag{3.7}$$

By applying CSP model to our values and constraints using equations (3.3, 3.4, 3.5, 3.6 and 3.7) we obtain the following:

$$X = \{BW, HD, WB\} \tag{3.8}$$

$$D = \{0 , 1\} \tag{3.9}$$

$$C = \{(\forall x_i), (C_i(x_i) \leq \ \varDelta_i)\} \tag{3.10}$$

Where the variable x includes the following parameters: Bandwidth (BW), Hardware (HW) and Web Browser (WB). Δ_i is a finite possible value. The domain for each variable is either true or false (1 or 0) and the relationship of the constraint is that for every x, there is a finite set of all possible values.

$$csp = < X , D , C > \tag{3.11}$$

Where:

$$x = \{x_i \ \ / \ \ 1 \leq i \leq 3\} \tag{3.12}$$

x: is the set of variables $\{ x_1, x_2, x_3 \}$

$$D = \{D(x_1), D(x_2), D(x_3)\} = \{\{0,1\}, \{0,1\}, \{0,1\}\} \tag{3.13}$$

D: is the set of domains of the variables of x.

$$c = \{c_j \ / \ 1 \leq j \leq m\} \tag{3.14}$$

C: is the finite set of constraints.

For each constraint C_j, there is a relation defined by:

$$c_j(x_j) \subseteq \Delta_j \tag{3.15}$$

Where:

$$x_j = (x_{j1}, x_{j2}, x_{j3}) \tag{3.16}$$

and

$$\Delta_j = D(x_{j1}) * D(x_{j2}) * D(x_{j3}) \tag{3.17}$$

Let R be the finite set of rendering techniques of constraint C, which means for each instance of $x_{j,}$ there exists a unique mode $r_j \in$ R. The same mode corresponds to several instances.

$$R = \{ HW, MC, SDF, OGL\} \tag{3.18}$$

Therefore, WebGL rendering (HW), for example, corresponds to the display mode where all constraints are satisfied (1,1,1). Signed Distance Field rendering (SDF) corresponds to the display mode where all constraints are satisfied (0,1,1). MC rendering (MC) corresponds to the display mode where all constraints are satisfied {(1,0,1), (0,0,1)}. Server-side rendering using OpenGL (OGL) corresponds to the display mode where all constraints are satisfied {(1,1,0),(1,0,0),(0,1,0),(0,1,0)}

The selection and the transition between the different rendering techniques R can be defined as deterministic automata.

$$\Sigma = (R , \Delta , \delta) \tag{3.19}$$

Where: R is the finite set of rendering techniques, Δ is the finite set of all possible values and δ is the transition function:

$$\delta : R * \Delta \rightarrow R \tag{3.20}$$

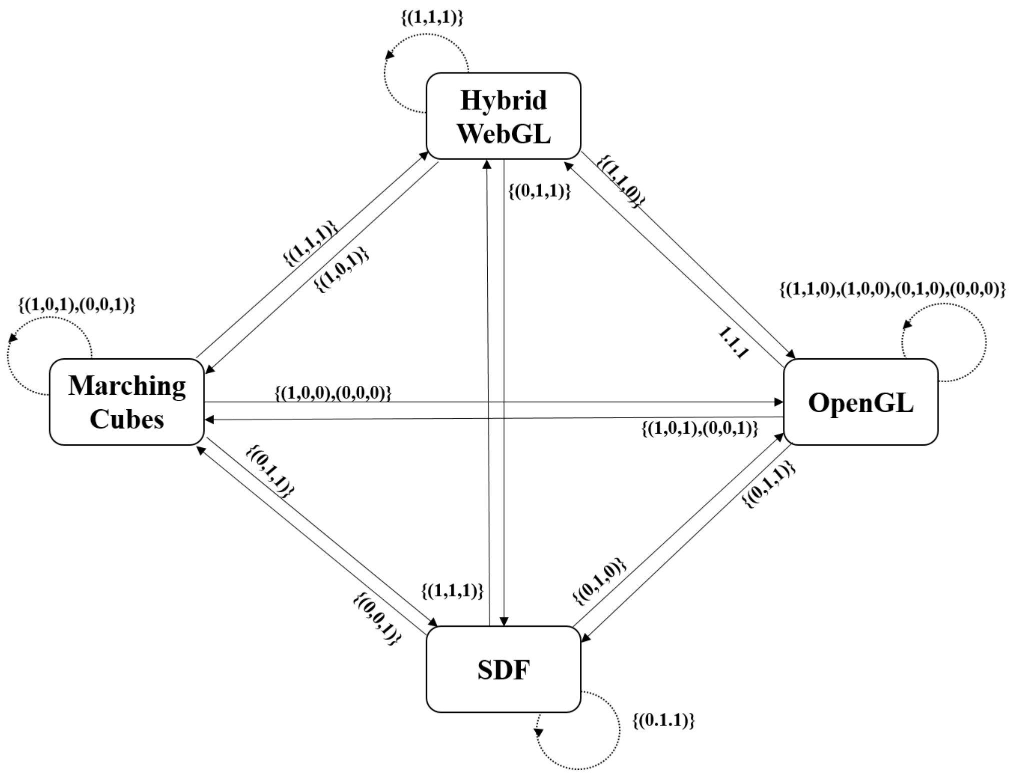


Figure 14: *Constraint satisfaction diagram showing the transition from state to state depending on variable constraint*

The deterministic automata are represented in Figure 14 and allow to graphically represent the transition between different rendering modes depending on the changes of the variables where all constraints are satisfied. The set of variables is represented by BW, HW, and WB, where each variable has two data sets as follows: $\{\{0,1\}, \{0,1\}, \{0,1\}\}$

$$HW * (1,1,1) \rightarrow HW \quad where\ \Delta\ is\ (1,1,1) \tag{3.21}$$

$$HW * (1,0,1) \rightarrow MC \quad where\ \Delta\ is\ (1,0,1) \tag{3.22}$$

3.5 3D Environment analysis

The goal is to create a Web-based heterogeneous volume modelling and rendering environment suitable for a wide range of online users. The modelling pipeline for the proposed environment is composed of three major phases as discussed above: Modelling, Processing, and Rendering. FRep and BRep are both used in the modelling part. Such representations allow us to track the modelling process using the constructive tree, and they can generate volumes having internal structures. Polygonal meshes approximate the precise model and are used to visually represent them and HF functions are used to store models.

3.5.1 Web-based instant modelling and rendering core parts analysis

Modelling and rendering are the major parts necessary for the modelling and development process as shown in Figure 15.

3.5.1.1 Modelling of heterogeneous volume objects with FRep

Complex geometry can be represented using a continuous real-valued function, i.e. with Function Representation (FRep). Three different values are assigned to this function as point coordinates. The term implicit surface of the geometry is used when assigning a zero set to the function in order to represent its surface [Pasko *et al*., 2004**]**. The continuous function that represents the geometry is defined as follows (eq. 3.23):

$$f(x, y, z) \geq 0 \tag{3.23}$$

Where: the corresponding implicit surface of the geometry is defined as follows:

$$f(x, y, z) = 0 \tag{3.24}$$

Where x, y, and z are three variable values assigned to the function of point coordinates.

Since geometric models can be described by a continuous function, this allows for applying different algebraic and geometric operations. The process of disintegrating the complex geometry into a set of primitive shapes combined using different operations is a constructive approach that resembles Constructive Solid Geometry (CSG). In other words, CSG is used to build up a complex geometry starting from a set of primitive ones using algebraic operations. FRep makes use of the same concept of representing primitives and operations with a continuous real-valued functions of point coordinates. FRep uses the concept of CSG to keep the construction history of the model. It is composed of primitive objects supported by a set of relations and operations including intersection, union, subtractions, etc. [Pasko *et al.*, 2004] [Gupta *et al.*, 2010].

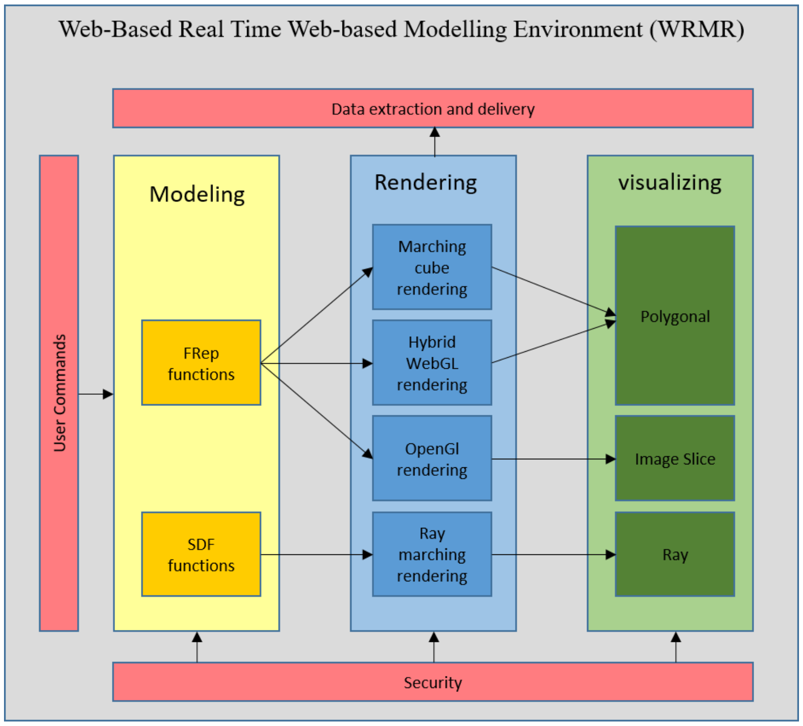


Figure 15: *Structure of the Real-time modelling and rendering (WRMR) environment*

HyperFun (HF) is a self-contained programming language designed to interpret FRep models. It is a specialised high-level programming language that allows for building up complicated models from scratch, it contains all the necessary operations suitable to define and implement FRep models. The language contains a library that includes a wide range of different geometric operations such as blending, rotating, intersecting, and many other operations. It is platform independent and allows Web-based participation by delivering models in the form of Java Applets suitable for Web-browsers [Pasko *et a*l., 2004], [Pasko *et al*., 2001].

Signed Distance Field SDF is described as an implicit scalar function [Oleynikova *et al*., 2016] and is represented as follows (eq. 3.25):

$$f(x) = \begin{cases} d(x, \partial\Omega) & if\ x \in \Omega \\ -d(x, \partial\Omega) & if\ x \in \overset{c}{\Omega} \end{cases} \tag{3.25}$$

Where: $\partial\Omega$ is the set of boundary points of Ω.

The primitive type of geometric model can be referred to a point set. At any point of the point set, there exist a mathematical model of the object property called point set attribute, where each point set and its attribute can be assigned a real-valued function and a tree structure. Attributes assigned to homogenous solids allow for editing the solid as a whole but not its components. A scalar field is a real-valued function and when combined with a set or a subset of 3D space is then transformed into heterogeneous volume. The scalar field in a heterogeneous volume may represent the volume attribute, its density field, or its distance field [Pasko *et al*., 2001].

As discussed in chapter II, FRep is defined as a uniform representation for 3D geometry, and includes a set of set points (objects) O, operations P, and relations R: $< O, P, R >$. O is represented as a continuous function F(x) where the points with F(x) $\geq$ 0 constitute the geometry. Heterogeneous objects can be mathematically represented as follows (eq. 3.28):

$$H = (O, M) \tag{3.26}$$

$$H_i = (O_i, M_c) \tag{3.27}$$

$$H = \sum_{i=1}^{n} G_i \ and\ G_i = \sum_{j=1}^{m} G_{ij} \tag{3.28}$$

Where Heterogeneous objects are represented by H, which is also a set of n cells, O is the Object information and M is the material composition, H_i refers to the ith point in the Object O_i, accompanied by m sub-objects with a specific material (M_c) [Gupta *et al*., 2010].

In general, a hyper-volume object, which is a point set accompanied by attributes, can be represented as follows [Schmitt *et al*., 2004] (eq. 3.39)

$$o = (G, A_1, \dots, A_k) : (G, S_1(X), \dots, S_k(X)) \quad (3.29)$$

The above formula (eq. 3.29) representing the hyper-volume objects is called constructive hyper-volume model. X is an Euclidean space point, G is a point set and S_i is a function standing for an attribute A_i and G and S_i are represented using FRep as real-valued definition function *F* called constructive geometry tree. *F* is evaluated during the construction of the tree, by traversing the geometry tree structure, where operations constitute the nodes and the simple volumes occupied the leaves [Schmitt *et al*., 2004], [Schmitt *et al*., 2001]. The attribute functions are also constructed with trees, where primitives are leaves and operations are nodes. The attribute is evaluated in a similar manner as *F* by traversing throughout the attribute tree.

HF deals with parameterised hyper-volume objects represented with FRep. Objects are defined as functions followed by necessary attributes represented as scalar functions. Users can build 3D volume objects from scratch using FRep library which allows adaptation and customisation and the can make use of the existing functions for primitives such as blocks, spheres, and other basic primitives as well as more complex primitives such as convolution surfaces with different skeletons, metaballs, soft and blobby objects in addition to a wide range of deformation functions including twisting, blending, scaling, etc. [Schmitt *et al*., 2001].

3.5.1.2 Rendering approaches analysis

One of the major roles for any instant and Web-based modelling and rendering environment is to be platform independent thus serve different users with different hardware resources. To do so, different rendering approaches should be taken into consideration in order to deliver different models to different clients based on their available hardware power. In this work, we suggested four different rendering approaches, three of them are client-side and the last one is server-side rendering. The client side rendering approaches are Marching Cubes (MC) (the algorithm is discussed in details in 4.5.1.1), WebGL (the algorithm is discussed in details in 4.5.1.2) and Signed Distance Fields (SDF) (the algorithm is discussed in details in 4.5.1.3) while the server-side is rendering using C++ (the algorithm is discussed in details in 4.5.1.4). Information collection process should take place in order to determine what kind of rendering approach should take place based on the client's hardware, available Internet bandwidth, and the graphics adapter power.

Below we describe in more detail four rendering techniques, three client-based and one server-based. MC is a surface rendering originally developed to study medical images. It is an isosurface extraction algorithm where volume data are subjected to a division approach and are processed using cells or cubes. The intersection between the cube edges and the isosurface is detected and classified as inside or outside the isosurface [Cirne et al., 2013], [Parmar *et al.*, 2016]. The server-based rendering approach uses OpenGL, a rendering library, which is a platform independent and supports 3D graphics. OpenGL has an application programming interface (API) and supported by a utility ToolKit library (GLUT) that makes it possible to visualise 3D scenes on different platforms. The server uses OpenGL to generate a stream of images to be displayed on the client browser using image-slides technique. WebGL is based on a scripting language emerged from OpenGL API and is written in JavaScript suitable for 3D graphics rendering on the Web-browser using programmable shaders. WebGL is supported with different libraries such as tools modelling and rendering, one of the most well-known libraries is THREE.js [Congote *et al*., 2012]. Signed distance field (SDF) is an alternative representation over mesh-based ones in terms of efficiency with respect to visualisation while keeping advantages of implicit modelling. SDF rendering can be accelerated by using sphere tracing, a type of ray-tracing which utilises a distance property of the scalar field. [Zollhofer *et al*., 2015].

The process of measuring the efficiency of an algorithm is called algorithm complexity analysis, which depends on the number of steps including operations, conditions and loops, and the time needed to execute them. Big O notation O(g(n)) is used to measure the run-time using the worst-case algorithm scenario and is represented as follows (eq:3.30):

$$f(n) = O(g(n)) \tag{3.30}$$

The list of sequence statements in the algorithm is represented by C and the loops (for, while, do etc…) are represented by n

$$f(n) \le C * g(n) \tag{3.31}$$

Below, we will try to apply the complexity analysis to the three rendering approaches starting from the MC algorithm, followed by WebGL and SDF rendering. The main loops and the sequence statements of the three algorithms will be taken into account in their most general forms in order to calculate the complexity of the algorithm in terms of n.

1) MC algorithm.

The complexity analysis for the MC algorithm is $O(n^3)$:

$$f(n) = \big((c+1)n - (c+1)\big) * \ (n-1) * \ (n-1) \rightarrow O(n^3) \tag{3.32}$$

2) WebGL algorithm.

The complexity analysis for WebGL algorithm is $O(n^4)$:

$$f(n) = \Big(\big((c+1)n - (c+1)\big)\Big) * (n-1) \ * \ (n-1)) + (\big((c+1)n - (c+1)\big) * \ (n-1) \ * (n-1) \ ((c+1)n - (c+1))) \ \rightarrow O(n^4) \tag{3.33}$$

3) SDF algorithm.

The complexity analysis for SDF algorithm is $O(n^2)$:

$$f(n) = \Big[\Big(\big((c_1+1)n-(c_1+1)\big)+\big((c_2+1)n-(c_2+1)\big)\Big)*(n-1)\Big]+$$
$$\Big[\Big(\big((c_4+1)n-(c_4+1)\big)+\big((c_5+1)n-(c_5+1)\big)\Big)*(n-1)\Big]$$

$$f(n) = O(n^2)+O(n^2) \rightarrow O(n^2) \qquad (3.34)$$

By comparing equations (3.32), (3.33) and (3.34) we can conclude that the level of complexity of the MC algorithm is O(n^3) while WebGL is O(n^4) and SDF is O(n^2) that means SDF is less complex and much faster than WebGL and MC. MC is less complex than WebGL and therefore faster.

The detailed explanations of the above algorithms are found in appendix B

3.6 Heterogeneous volumes with attributes.

Heterogeneous volumes are objects composed of different internal materials. In this work we are using heterogeneous models that are based on FRep, which is composed of a set of volumes and set of operations on them. FRep benefits from a constructive approach, where the resulting function is composed of basic functions for primitives and operations. This approach is one of the most used approaches in heterogeneous modelling. The constructive approach applies different operations on primitive models to integrate them into a more complicated model by generating a constructive tree where leaves contain primitive models defined by a real-valued function *f* and nodes are operations as shown in Figure 16.

Shcmitt *et al*., discussed the concepts of heterogeneous volumes which are point sets and functions representing attributes [Schmitt *et al*., 2004] as follows:

$$o = (G, A_1, \ldots, A_k): (G, S_1(x), \ldots, S_k(x)) \qquad (3.35)$$

Where x=(x_1, … x_n) Euclidean space point, S_i is a function standing for an attribute A_i and G is a point set.

Figure 3.16 shows a constructive tree of primitive objects combined to construct a volume, in our case a tap. The leaves are primitive objects subjected to operations (union operation). As a result, both primitive objects and operations constitute the whole volume. Three operations were applied to generate the final volume (Fig 16).

FRep functions are used to model heterogeneous volumes with different densities. Primitive models are used to construct a more complicated volume using HF operations.

$$M = (P, O, C) \tag{3.36}$$

Where M is the model, P is the set of primitive objects, O is the set of operations applied, and C is the set of attributes.

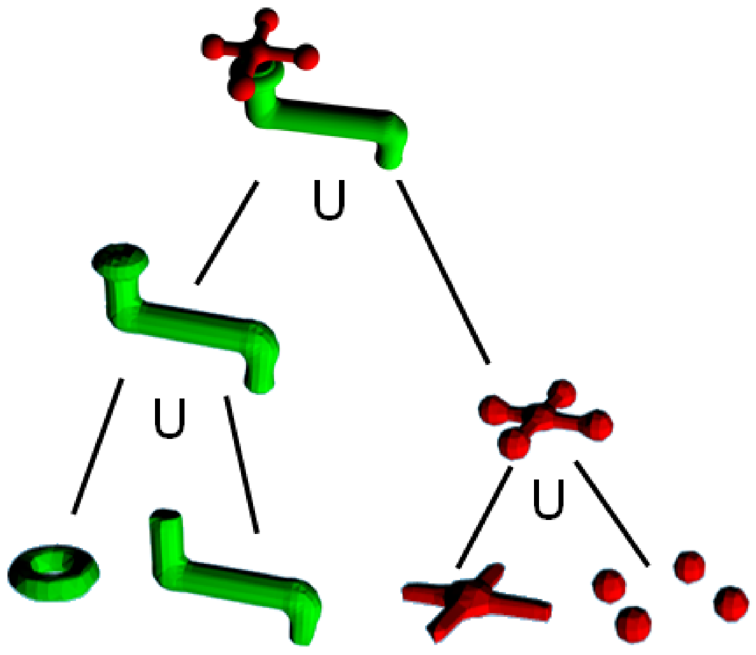


Figure 16: *Constructive tree showing simple models as leaves and operations as nodes*

The equation above (eq. 3.36) shows that a heterogeneous model is composed of set of primitive objects subjected to a set of different operation, HF functions in our case, and different colour attributes. A constructive tree can be built and contains different successive heterogeneous volumes with their attributes and functions. The constructive tree for a hemisphere with microstructure model is shown in Figure 17, it is composed of six consecutive levels, each level carries different models, operations and attributes. The primitive models are consequently transformed into more complicated one as the layers of the tree goes high. Different operations were applied including intersection, blending, rotating and union. The operations are HF based operations and the colour attributes are applied all along the modelling process on different tree levels.

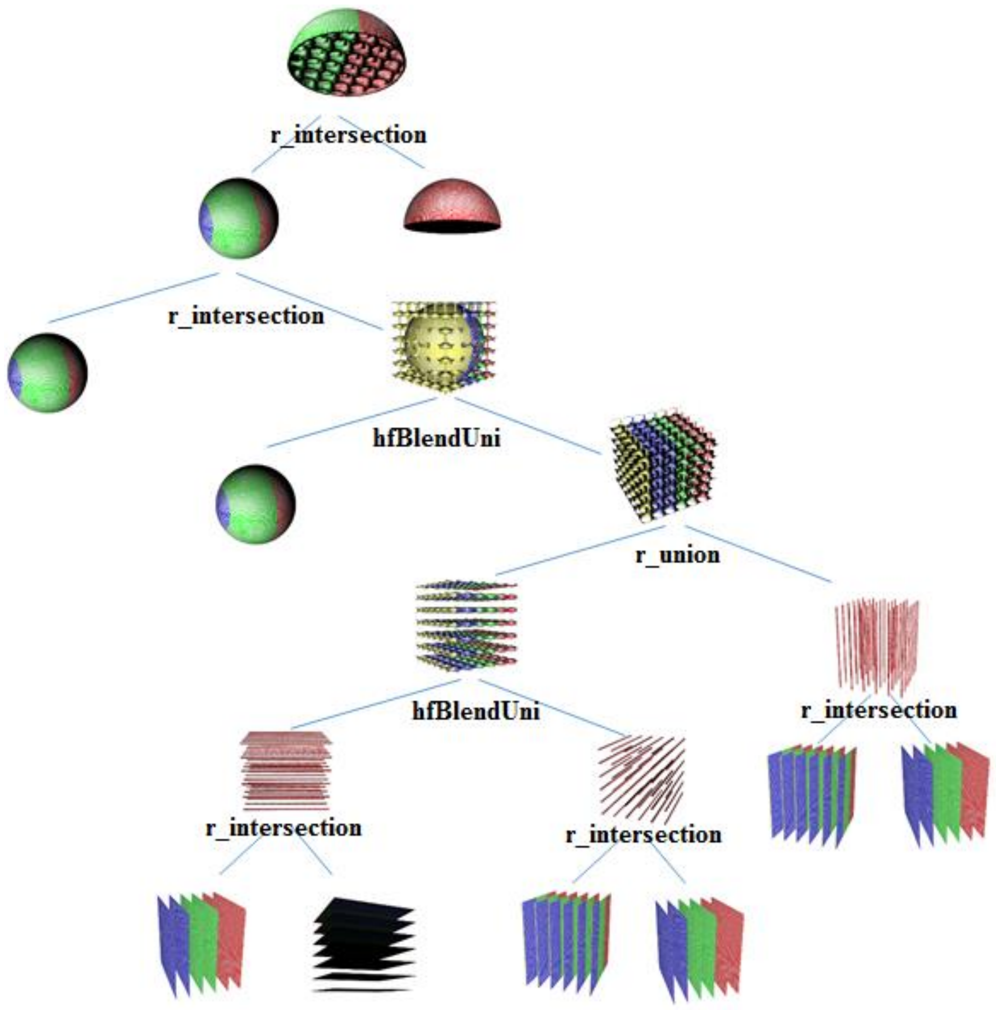


Figure 17: *A constructive tree for a complicated heterogeneous model with different HF functions as operations*

In this work, we dealt with the colour attributes in two different approaches: primitive model colouring and heterogeneous model colouring. Both approaches use (eq. 3.35) and (eq. 3.36) at the modelling phase to apply colour attributes

3.6.1 Primitive model colouring approach (Leaf colouring)

This approach is based on colouring primitive models on leaves level in the modelling constructive tree. This allows primitive multi-models colouring before combining them into one model and is basically used in simple models. Figure 18 shows primitive models coloured with different colours and then joined together to form the final model (Fig 18.a).

Several steps are applied while using primitive model or leaf colouring approach, these steps are:

Create geometry

Use MC to transform the geometry into the mesh

Create mesh material

Assign colour attribute to mesh

Display primitive model.

3.6.2 Heterogeneous volume colouring approach (Node colouring)

In this approach, we apply the colouring process at the node level of the constructive tree. The same concept of colouring is used; the major difference is by applying the colouring process at the mesh level. The colour attribute is set during the modelling phase, more than one colour can be applied to different nodes of the constructive tree, thus providing a multi-coloured complicated model, which allows applying different colours to heterogeneous volumes taking into considerations its density and its internal structure (Fig 18.b).

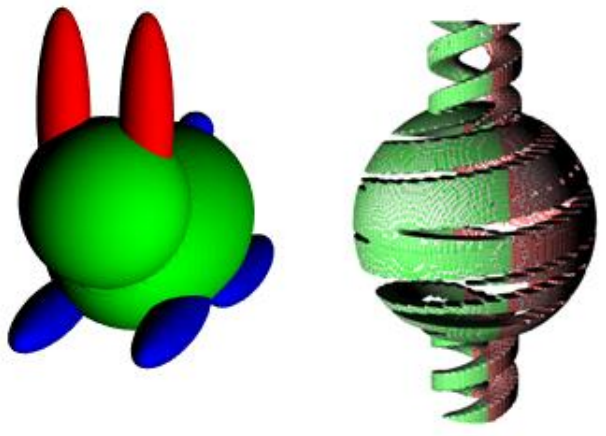


Figure 18: *Two different approaches to colouring, (a): using primitive model colouring (Leaf), (b): using complicated heterogeneous volume modelling (Node)*

3.7 Summary

In this chapter, we discussed the theory behind 3D Web-based modelling environments, and we focused on its different characteristics. We also discussed different scenarios for online rendering and implemented a special scenario optimisation framework using constraints satisfaction problem (CSP). We finally discussed the design of the proposed 3D environment and discussed some of its features. We presented the major parts of the proposed environment and discussed complexity of the rendering approaches being used. In Chapter 4, we will discuss the implementation phase of the design presented, we will focus on the different rendering techniques, and compare results. We will also discuss the characteristics of the proposed 3D environment in terms of functionality and performance, and we will present some major tools offered by the environments as well as the supporting parts such as security, data delivery, and real-time performance.

Chapter 4

Engineering, design and implementation

The Web-based real-time modelling and rendering environment (WRMR) is a client-server, Web-based and platform independent environment, it uses the browser as an interface and can gain access to clients' hardware resources. The client-side is developed using JavaScript and Visual Basic .Net technologies while the server-side uses C++. As discussed above, WRMR is an adaptive environment designed to serve different types of uses in real-time. In this Chapter we are going to discuss in details the components of WRMR. Figure 19 shows that the environment is subdivided into five inter-related parts that work together to do the job.

4.1 WRMR general characteristics

WRMR is built up from different modules integrated together to perform the modelling and rendering tasks as required. In this section, we will discuss the reasons for connecting different modules together as shown in Figure 19. Since WRMR is an online application, it is based on FRep and SDF modelling functions, users need to access it directly by writing their own functions in the form of JavaScript functions or my load them from saved files. HF function needs to be transformed into JavaScript functions to be understood by the environment. Most users may not get the knowledge of transforming HF into JavaScript functions that is why we designed an "HF to JS" converter that supports the interpreter which works as an input to WRMR. The environment is a client-server architecture, and the job is distributed among clients and servers as discussed above. The decision on what side should the modelling and rendering take place is directly related to the user computational power. WRMR encapsulate four different rendering approaches, three of which are client-side and the last one is server-side. Different rendering approaches require two types of modelling functions (FRep and SDF). So far three different modules should be integrated together to perform the job. When completing the process, it is useful to save the work (both modelling functions and rendered modelled), for that reason WRMR contains a special module to save the 3D volume.

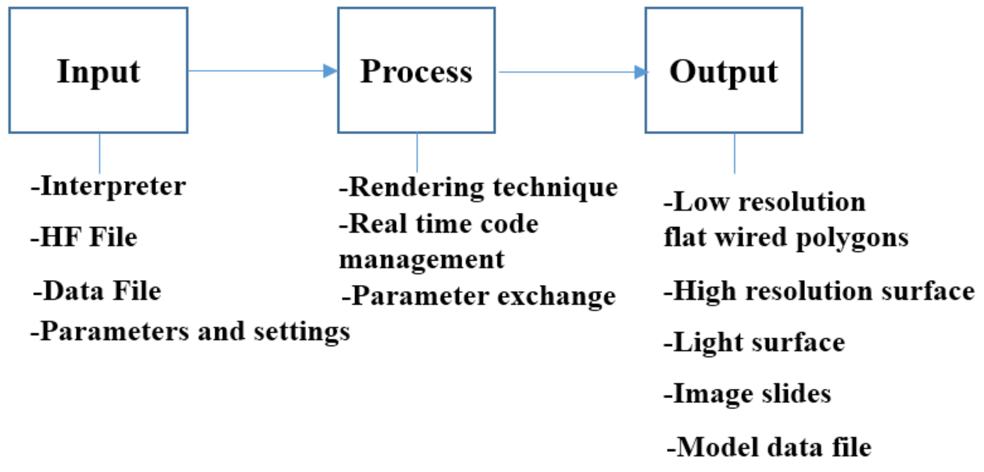


Figure 19: *WRMR conceptual parts:* input, process, *and output*

4.2 WRMR core engine

WRMR core engine is the information collection centre, through which it can analyse the client, and collect valuable information about the target platform, its operating system, and available Internet connection speed. The job of the core engine is to collect information from different inputs and deliver the outputs to the WRMR in the form of XML. The inputs and outputs are subjected to independent threads and synchronous calls. The engine performs a series of calls that take place independently, these calls are necessary to collect information about target platforms and operating systems necessary for the decision-making process that will take place. The output of the core engine will be used to decide the side of rendering (client or server) as well as what rendering approach to be used. These calls or tasks are as follows:

- Connection Speed detection.
- Machine Info detection.
- WebGL Information detection.
- Operating System (OS) Info detection.

- Screen Info detection.
- Output the results in an XML format.

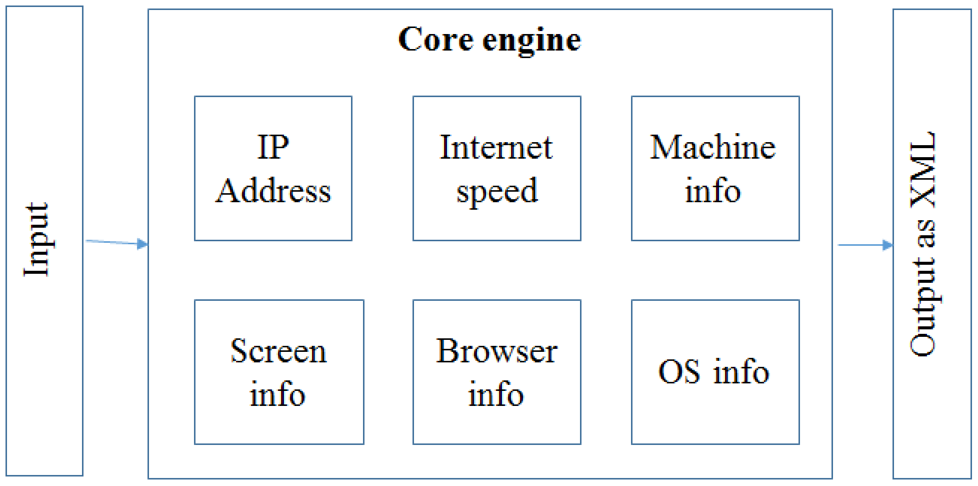


Figure 20: *Conceptual model of WRMR core engine*

These calls collect information and data about the client machine and send it to WRMR using XML messages. This process takes place periodically as long as the client is connected to WRMR and as long as it is requesting services. It is necessary to keep checking the inputs at the core engine and detect the changes that may happen. When a change is detected, the output will then change and the environment adapts to the changes and adjust its performance (Fig. 20). WRMR core engine detects the IP address of the client and determines its location, then checks the connection speed available in bits/sec to analyse the transfer rate (bandwidth) using the "Connection Speed detection" module. The necessary information to identify the type of the machine (computer, mobile, pad) is collected using the "Machine Info detection" module. The running browser is declared using "Browser Info detection" module, and the operating system can be identified using the "OS Info detection". The size of the screen can be determined using the "Screen Info detection" module.

4.2.1 Connection speed

Detecting the connection speed or the bandwidth transfer rate at the client's machine is a critical issue for WRMR. It helps in making the right decision about where, how and when the rendering process will take place. One of the efficient practices to test download speed is by downloading a file with a predefined size and calculates the necessary time for the download process to complete. Another way is to use a Web service to do the job. The problem with Web services is not always accurate. Our tests showed inconsistency when using different Web services to test the broadband speed. The Alternative solution is to measure the download by implementing a JavaScript function on our server to test the download speed at the client-side. In this way, we can measure the connection between the client and our server quickly, and respectively. A pre-defined file of a fixed size of 10 Megabytes located on the server side stared to be downloaded on the client-side as soon as the client is connected to WRMR. The duration of the download process is calculated by subtracting the end time from the start time. The transfer rate is obtained by dividing the file size in Kbytes by the duration obtained.

4.2.2 Machine information detection

Detecting and collecting information about the machine requesting the service helps WRMR decide how to deliver the service and distinguish between different devices including computers mobiles, tablets, etc... The machine info detection process determines the agent of the user and checks whether it is a pad, a smart-phone or a computer device. Some modern tablets are more efficient than old desktop machines and has more computational power. The user’s browser info could be not the most reliable way as some browsers allow giving fake id for security reasons, collecting information helps in supporting the security system of WRMR as discussed in section 4.3.3.2.

4.2.3 WebGL information task

WebGL is the core or the engine of the modelling process, WRMR needs to know whether the requesting machine is WebGL supported or not. Initialising a canvas successfully at the requesting machine indicates that the machine is WebGL enabled and informs WRMR that the client can render 3D models using a browser and GPU. WebGL is developing fast, a new version (WebGL 2.0) was released and supported with a lot of interesting features to support the rendering process.

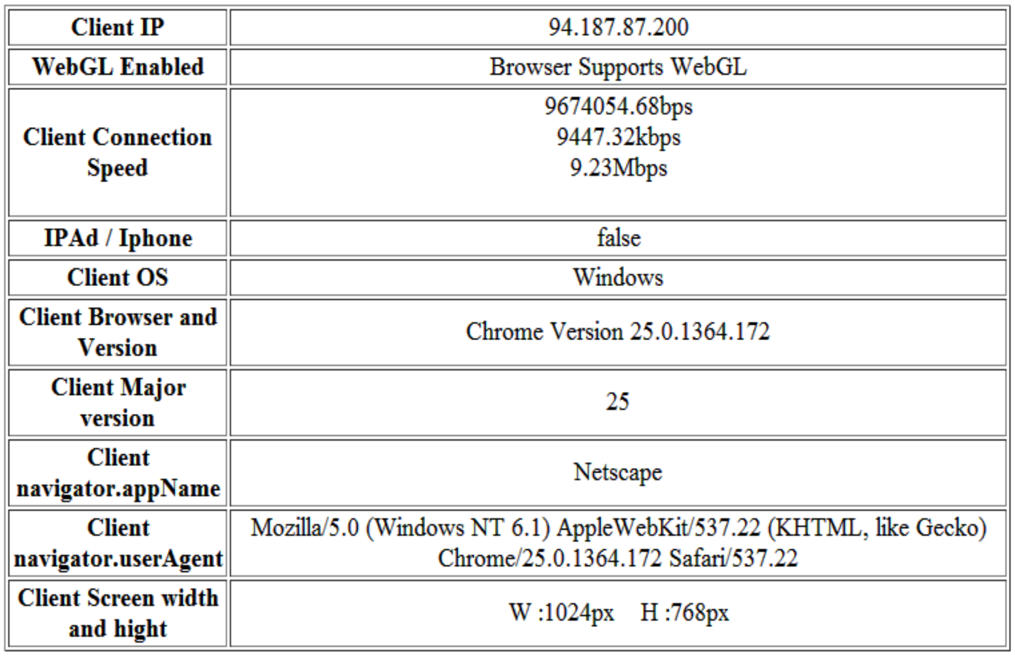
4.2.4 Operating system detection task

WRMR needs to know about the operating system being used at the client-side. The ability to determine the machine operating system gives a good idea to WRMR about types of actions and responses to be taken with the appropriate kinds of services. The types of operating system can be listed as follows: Windows, UNIX, and MacOS.

4.2.5 Screen information detection task

Calculating the screen width and height gives an idea about the machine used and whether it is a desktop, laptop, smart-phone or a tablet device. This set of parameters such as the type of the machine used, its available GPU and memory help in making the right decision in choosing the right rendering approach based on the pre-defined scenarios. The most critical information needed from the client hardware is as follows: the connection speed, the browser ability to use WebGL, and the hardware power (GPU and memory).

The diagram (Fig. 21) shows a case study in the machine info detection process. It starts by determining the agent of the user and then checks the kind of the device (portable or desktop). It then determines the functioning operating system. Checking WebGL ability is next; this can be done either from the canvas initialisation, where successful initialisation indicates that the machine supports WebGL or by analysing the Web browser information. For example, if the running browser is Internet Explorer of version 10 and earlier, it means that WebGL is not supported.



Client IP	94.187.87.200
WebGL Enabled	Browser Supports WebGL
Client Connection Speed	9674054.68bps 9447.32kbps 9.23Mbps
IPAd / Iphone	false
Client OS	Windows
Client Browser and Version	Chrome Version 25.0.1364.172
Client Major version	25
Client navigator.appName	Netscape
Client navigator.userAgent	Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.22 (KHTML, like Gecko) Chrome/25.0.1364.172 Safari/537.22
Client Screen width and hight	W :1024px H :768px

Figure 21: *Information gathered from the client machine*

4.3 WRMR conceptual design

WRMR is divided into three interrelated conceptual blocks: input, process, and output as discussed in 4.1. Figure 19 shows the blocks with their components, the input contains the interpreter, HyperFun (HF) file, mesh data file, and parameters and settings modules. The process part contains the rendering, instant code management, and parameter exchange modules. The output contains low-resolution flat wired polygons, high-resolution surface, light surface, image slides, and model data file module as shown in Figure 19. All the above-mentioned modules are designed to support the modelling and rendering process, the pipeline for the three parts will be discussed later in 4.3.2.

4.3.1 WRMR input

The first WRMR block is the input part that contains different modules including Interpreter, HF and SDF files and parameters and settings. The interpreter plays the input role to WRMR and allows users to write HF or SDF functions to be executed and rendered instantly. It also allows importing data from outside the environment mainly HF and SDF functions as well as mesh data files in the form of faces and vertices. The attributes and settings function allows to adjust the 3D heterogeneous volumes details and control some parameters related to them such as density, distances, colours, etc.

HFtoJS is a converter responsible for transforming the HF functions into JavaScript functions ready for rendering using the WRMR. WRMR depends on HF functions and attributes as inputs using the interpreter. The environment was developed using JavaScript language because it is a Web-based online application. Users familiar with HF functions find it difficult to transform them and attributes into JavaScript code suitable for the environment. That is why we developed the HFtoJS converter, which is an online instant converter that converts HF functions into JavaScript functions

4.3.2 WRMR process

The process block contains different modules including rendering, code management, and parameter exchange. The process stage is responsible to load the source code into the client's memory. The core code is divided into three parts as follows: set-up code, client interpreter code and the complementary code as shown in Figure 22. The set-up code is essential to set all the parameters necessary to launch the rendering process such as preparing the scene, setting the camera, determining the rendering approach, and initialising the canvas if needed.

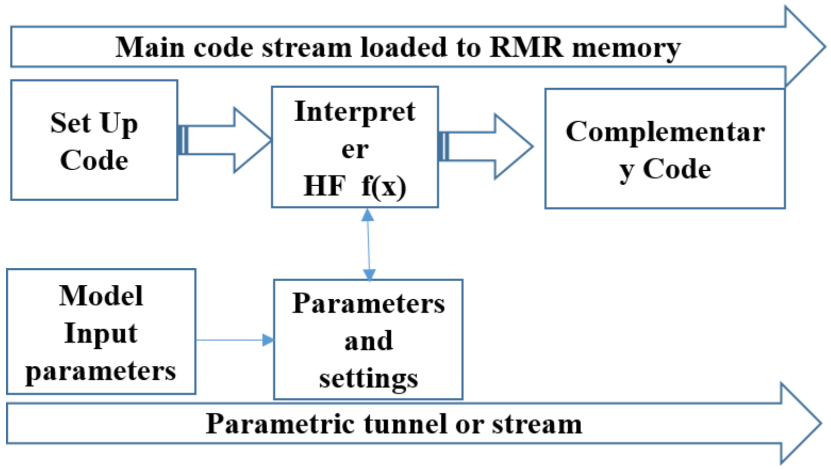


Figure 22: *WRMR process part showing source code integration in the client's memory background*

The idea of the interpreter is to evaluate the expression by calling compiled functions given the original source code. The interpreter allows users to freely add, modify and delete line-code ready to be executed in instant time. It is located in the middle between the set-up code and the complementary code. The set-up part and the complementary part include all the predefined modelling functions written in JavaScript and are essential for the modelling process. The three mentioned above parts are integrated into one main code stream loaded in the client's browser memory. Model input and changing parameters and settings are two supporting tools designed to help the client to edit the model and change some of its characteristics. These supporting tools can immediately modify the heterogeneous volume and control its behaviour, location, resolution, and the angle of vision. The parametric tools use JavaScript language and are loaded into a special tunnel or memory stream using the client's browser memory. They are also used to transfer different variables and parameters among WRMR files.

4.3.3 WRMR Output

The rendering and visualising part includes the followings: flat wired polygons, high-resolution surfaces, light surfaces, image slides, and object data files. This part is responsible for displaying the model after rendering it using one of the available rendering approaches. The low resolution flat wired polygons resulted from MC rendering, WebGL rendering produces high-resolution models and the SDF rendering generates models using light rays. Image slides are received from the server-side where the rendering takes place. 3D objects data file allows to save all the models in OBJ or X3D file formats, the user can use these models later in any other modelling environments.

4.4 WRMR major parts

The proposed environment is an online set of tools suitable for instant modelling and rendering. Some supportive tools are designed and implemented to make it easy-to-use, safe and interactive. These parts are user control, data extraction and delivery and security.

4.4.1 User command and control

The user command tool allows users to set-up attributes and parameters in addition to the ability to write and edit complicated volume functions instantly. The environment is supported by an interpreter, the interpreter is necessary and works as an input to WRMR. As discussed before, since WRMR is an online, and interactive environment, there is a need to let users write their own modelling functions (HF commands) using an open-source JavaScript language. Using the interpreter, users can write, edit, delete and save models, in addition to the available ready-made models provided by the environment. Primitive and complicated heterogeneous models are available online.

4.4.2 Data extraction

The data extraction process is directly connected to the rendering type of the process. The environment can make use of the client's GPU and memory and has the ability to build up 3D volumes using polygons generated by MC and WebGL rendering. Saving 3D volumes is a need to any 3D modelling environment, 3D volumes are represented in many ways including functions and meshes, and there is a need to save these functions and meshes for later use. Functions can be HF or JavaScript, and both represent 3D volumes and can be saved before and during the rendering process in TEXT or JS files. The polygons are constructed by extracting raw data in the form of faces and vertices on the client's machine. Extracted data are saved in 3D data files in the form of X3D or OBJ data, and can be easily imported and loaded to the environment. OBJloader is a special function provided by WebGL THREE.js library to load OBJ files. This process allows hiding the raw data (faces and vertices) from users and can protect the model from theft and piracy.

The raw data (faces and vertices) are extracted using (eq. 4.1,4.2,4.3)

$$point = \sum_{z=0}^{s} \sum_{y=0}^{s} \sum_{x=0}^{s} (x + s * y + s^2 * z) \tag{4.1}$$

Where s represents the cube size

The point set of the rendered model can be extracted using equation 4.1, where a set of nested loops representing the three-dimensional axis are applied to extract all the points constituting the model in free space. Here x, y, and z represent three axes in space and s represents the cube size..

Server-based rendering is used when the user requesting a service is using a low hardware device and is unable to go rendering using his/her GPU. The server-based rendering then uses the server GPU to generate a stream of images in the form of image slides. (Fig 23).

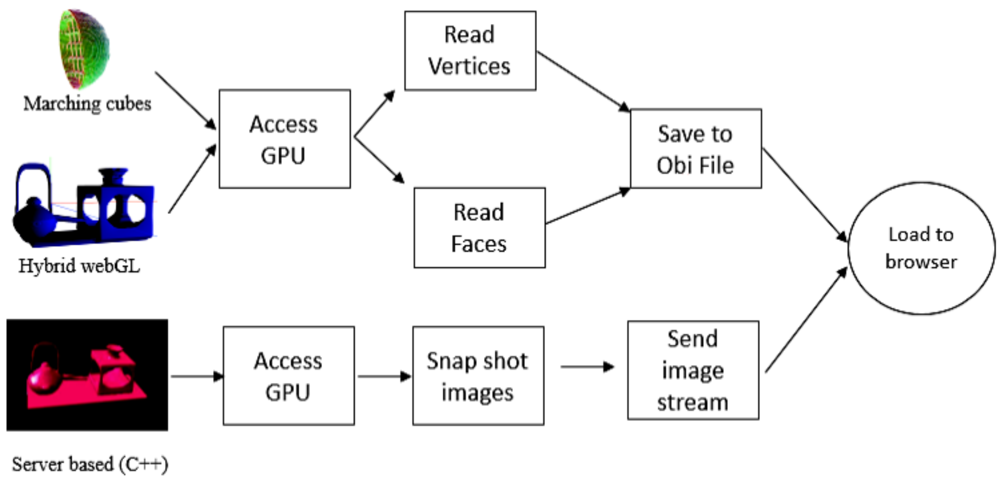
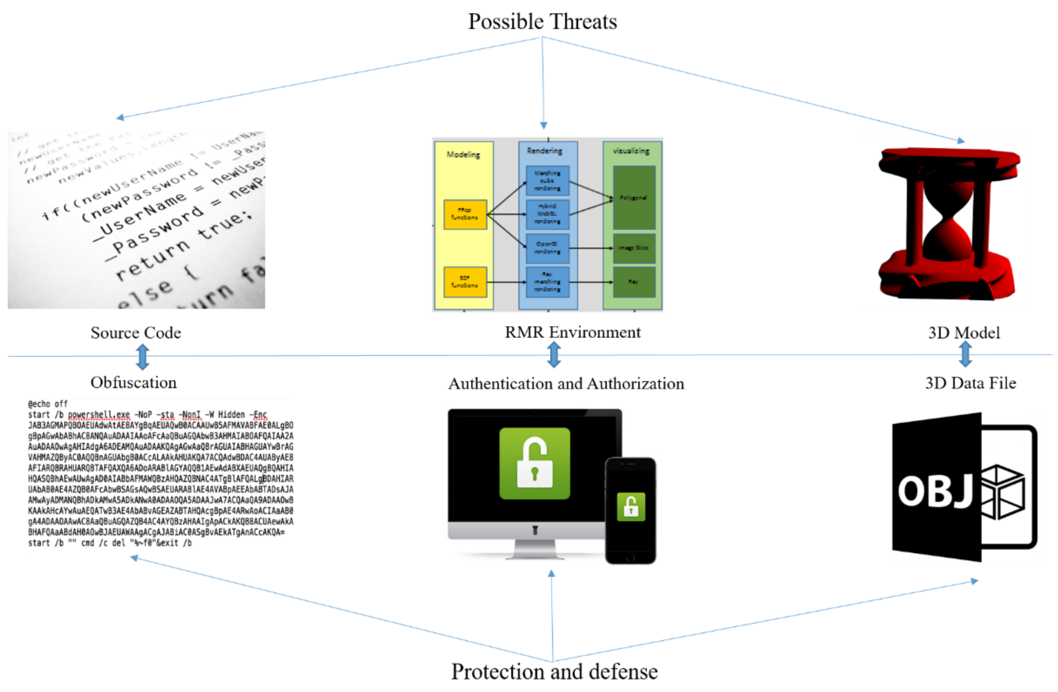


Figure 23: *The process of extracting raw data as vertices and faces can be done using different rendering approaches.*

4.4.3 Security

Theft and piracy are two big concerns, when dealing with online environments. Security tools and techniques should be taken into account to protect the environment and the models from illegal users since it is an open source online environment.

Code protection using Obfuscation, Authentication and Authorisation, and Data files extraction are three security techniques implemented as an attempt to protect the environment or at least to make it really difficult for illegal users to access it and steal the models are shown in Figure 24.



```
@echo off
start /b powershell.exe -NoP -sta -NonI -W Hidden -Enc
JAB3AGMAPQBOAEUAdwAtAE8AYgBqAEUAQwB0ACAAUwB5AFMAVABFAE0ALgBO
gBpAGwAbABhAC8ANQAuADAAIAAoAFcAaQBuAGQAbwB3AHMAIABOAFQAIAA2A
AuADAAOwAgAHIAdgA6ADEAMQAuADAAKQAgAGwAaQBrAGUAIABHAGUAYwBrAG
VAHMAZQByAC0AQQBnAGUAbgB0ACcALAAkAHUAKQA7ACQAdwBDAC4AUAByAE8
AFIARQBRAHUARQBTAFQAXQA6ADoARABlAGYAQQB1AEwAdABXAEUAQgBQAHIA
HQASQBhAEwAUwAgAD0AIABbAFMAWQBzAHQAZQBNAC4ATgBlAFQALgBDAHIAR
UAbAB0AE4AZQB0AFcAbwBSAGsAQwBSAEUARABlAE4AVABpAEEAbABTADsAJA
AMwAyADMANQBhADkAMwA5ADkANwA0ADAAOQA5ADAAJwA7ACQAaQA9ADAAOwB
KAAkAHcAYwAuAEQATwB3AE4AbABvAGEAZABTAHQAcgBpAE4ARwAoACIAaAB0
gA4ADAAOAAwAC8AaQBuAGQAZQB4AC4AYQBzAHAAIgApACkAKQB8ACUAewAkA
BHAFQAaABdAH0AOwBJAEUAWAAgACgAJABiAC0ASgBvAEkATgAnACcAKQA=
start /b "" cmd /c del "%~f0"&exit /b
```

Figure 24: *The three different security threats and their defences: Code Protection (Obfuscation), Authentication and Authorisation, and Data Extraction*

4.4.3.1 Code protection using obfuscation

Code obfuscation is a process used to encrypt source code, mainly the JavaScript code, without losing its functionality. It can still run even after encoding. Obfuscation is used to protect source code against reverse engineering. It can construct and publish encrypted programs with the same functionality of the source program using an encryption key. The main purpose of obfuscation is to make it difficult to be read and understood by non-authorised users while keeping its functionality [Rivest *et al.*, 1978]. HyperFun code is text, using the code converter which is supported by WRMR, the environment is able to convert HF functions into JavaScript functions (will be discussed later), and can encrypt those functions since they are JavaScript functions. A special algorithm (Obfuscator) was implemented to encode the source code of the environment and the modelling functions. This code is in JavaScript, it is encrypted by the encryption algorithm while keeping it functional. This makes it difficult for users to read and understand the source code of WRMR as well as the encrypted HF functions (after being converted to JavaScript functions). The algorithm starts by deleting the unnecessary spaces and notes after scanning the source code to be encrypted. All functions and variables are scanned and detected and then listed to be encrypted. An encryption key is used to encrypt the functions and variables and then to save the encrypted code to another JavaScript file. A special function is used to determine the level of encryption complexity as shown in equation 4.2:

$$l = \sum_{i=0}^{a} \sum_{j=0}^{b} i * j^{k} \tag{4.2}$$

Where a is the length of the buffer, k is the complexity level

The idea behind Obfuscation is to encrypt the source code while keeping its functionality. Equation 4.2 uses all the keywords and functions represented by char(i), reads from the source code and stores in the buffer. After selecting the word, it iterates over every single character using a loop, and encrypts it by multiplying its ASCII code by the logarithmic function Key(k).

4.4.3.2 Authentication and authorisation

Authentication techniques are used to protect the environment form illegal users to access it. Privileges for authenticated users are granted after collecting basic information to monitor their behaviour and activity as discussed in section 4.2.2. User activities are saved in a log file for later monitoring. Users can be granted different privileges and access rights in order to save, edit, delete and access the source code of the models as shown in Figure 24. Information about a user requesting a service from the environment are collected and stored. This information includes the IP address, username and password, browser info, bandwidth available, etc. Users can be offered a full mode access and the ability to edit, save and delete models. Others may be granted a model view access, and this means they can only view models with no edit or save privileges (Fig. 25).

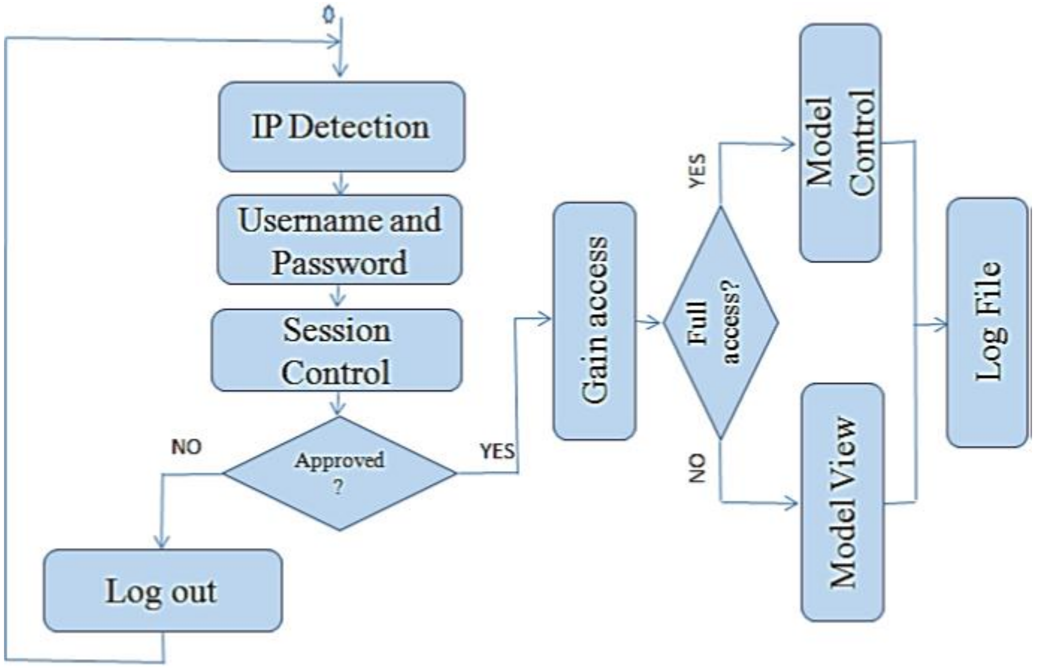


Figure 25*: Data flow diagram showing how authentication and authorisation are implemented*

4.4.3.3 Data files extraction

The process of extracting the faces and vertices from the client GPU and saving them to a 3D data file in the form of polygons allows hiding the source code of the JavaScript's functions used to construct the model. The OBJLoader functions provided by WebGL Three.js library allow displaying the model saved in the 3D data file without showing the source code or raw data.

The extraction process starts by extracting all existing vertices and faces and saving them in a buffer to be written later in the 3D data file. The point set is expressed in equation 4.1.

After the extraction process is completed, the extracted vertices and faces are saved in an OBJ or X3D file. The 3D data file can be shared and can be loaded into the Web browser using the following source code:

```
<X3D id="x3d1">
    <Scene id="scene1">
        <Viewpoint position="0.26255 0.13231 0.33884"
          orientation="- 0.25605 0.95645 0.08664 0.55979" >
</Viewpoint>
        <Transform id="Hemisphere">
            <Inline url="Hspher.x3d" solid="false"></Inline>
        </Transform>
    </Scene> </X3D>
```

4.5 WRMR modules

From the design prospective, WRMR is composed of five inter-connected parts. These parts are modelling, environment setup, rendering, visualizing (displaying), and data delivery. Some features are optional such as code obfuscation, data extraction, model extraction, etc. (Fig. 26).

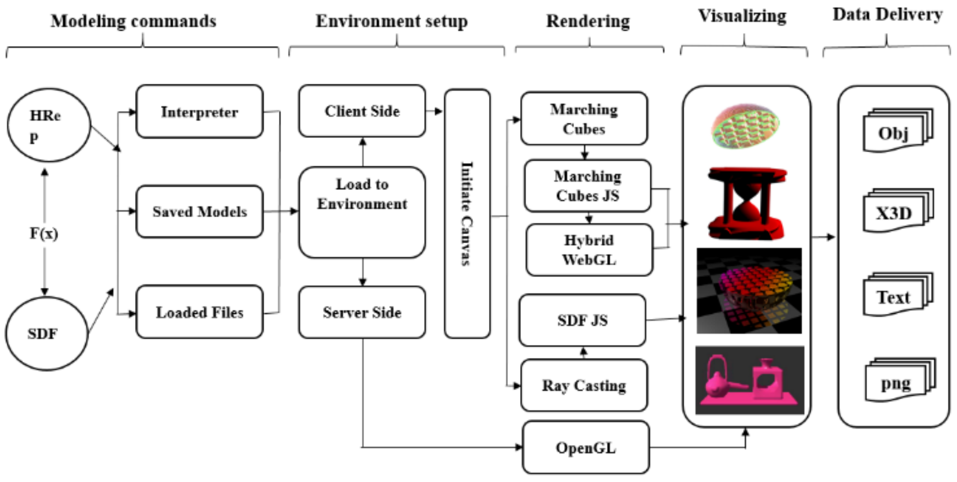


Figure 26: *WRMR parts, modules and their interconnections*

The first part is the modelling part where geometric functions are implemented and are divided into two types: FRrep and SDF functions. Those two used function representations precisely define the heterogeneous model and describe its characteristics. Editing the modelling functions can be done using the interpreter, which is a special tool designed to allow clients to write and modify their own modelling functions in instant mode. Another option is to save and load model functions from files, usually text or JavaScript files. These files contain heterogeneous models written in JavaScript language and ready to be executed as soon as they are loaded to the interpreter. The third option is to load pre-defined models supported by the environment, those simple and complicated heterogeneous models are loaded in the internal part of the environment. Users can use these models, edit them and save their own version for future use (Figure 26).

The second part is the environment set-up, where all the environmental settings take place. In this part, the environment decides whether to orient the user to use WRMR server or to use his/her local machine (client-side) depending on the scenario optimization and information collection discussed in 3.4 and 4.2. This part is responsible for collecting information about the user requesting the service and then redirecting him/her to one of the four available rendering options. The user will never worry about the rendering platform or the modelling system being used, all that he/she needs is basic modelling expertise.

The third part is rendering, where four rendering approaches will be available and ready to be used, these approaches are as follows: MC, WebGL, SDF, and OpenGL rendering. The first three approaches are client-side rendering while the fourth one is server-based rendering. The visualization part is the fourth part where 3D volumes are displayed. The precision of the 3D volume including its density depends on the applied rendering approach. The modelling, rendering, and visualisation parts are interconnected and can exchange data and commands. The last part is data extraction and delivery, where models can be saved in two ways. The first one is saving the modelling functions in a text or JavaScript file. This allows loading the functions to the interpreter for rendering or further editing. It also allows modifying the model outside the environment. The second way is to save the 3D volume data as point sets (faces and vertices) in a 3D file format such as OBJ or X3D files. This allows for loading 3D models directly to the environment without passing through the interpreter.

4.6 Summary

In this chapter we discussed the design phase of the environment. We started by presenting the parts of WRMR, then we discussed its core engine and the data collection methods. The major parts of WRMR included user command and control, data extraction and delivery. We also focused on the security part including obfuscation, user authentication, and data file extraction, and we discussed the different threats in our attempt to protect the environment and the models. We presented the conceptual design of the environment and discussed its parts and modules including the inputs, process and outputs, in additions to the interconnections between different modules.

Chapter 5

Implementation

Being a Web-oriented environment, most of the tasks for adaptation take place in the browser, meaning in the JavaScript code. Thus, at the start of the work, the server requests the information about the client by running specialised JavaScript module. The client sends the requested information (connection speed, machine info, browser info detection, OS info, and screen resolution info) to the server as XML messages as discussed earlier. The server selects one of the above-mentioned scenarios based on the information requested from the client. This process is periodically repeated during the work, while the client is connected to the server to ensure that the selected scenario is still the right one.

5.1 WRMR implementation

Four different rendering approaches are put into test: starting with MC rendering, then WebGL, followed by ray marching (SDF) and ending up with OpenGL using C++. The first three are client-based while the fourth one is a server-based rendering.

5.1.1 Rendering using MC

The MC algorithm was published by Lorensen and Cline in 1987 [Lorensen and Clinen, 1987]. By computing iso-surfaces from discrete volume data, MC produces a triangle mesh, and thus BRep can be built [Chi Sio *et al*., 2010]. We implement the algorithm using the pure JavaScript code, JavaScript is responsible for generating the triangle mesh at the client-side, and then do the rendering at his/her GPU. We took Hemisphere as an example, which is a semi-sphere with an internal grid of rods, to show both, the external structure and the internal grid. We start by creating a mesh for the Hemisphere object and set up the x, y, and z coordinate in the 3D plane.

```
mesher = mesh['Marching Cubes'], field = cobject['Hemisphere']();
result = mesher(field.data, field.dims);
x = -1;
y =-3.9;
z = 0;
drawObjMesh(mesher, field, result, x, y, z);
```

The Hemisphere object is modelled and then rendered using the MC algorithm by setting up the x, y, and z coordinates of the object to be created (centre of the object), its radius, and a lattice. A series of modelling sequences took place to obtain the final hemisphere object to be rendered (Fig. 27); the rendered object was of sharp edges and suitable for clients with limited GPU power [Pasko *et al.*, 2010].

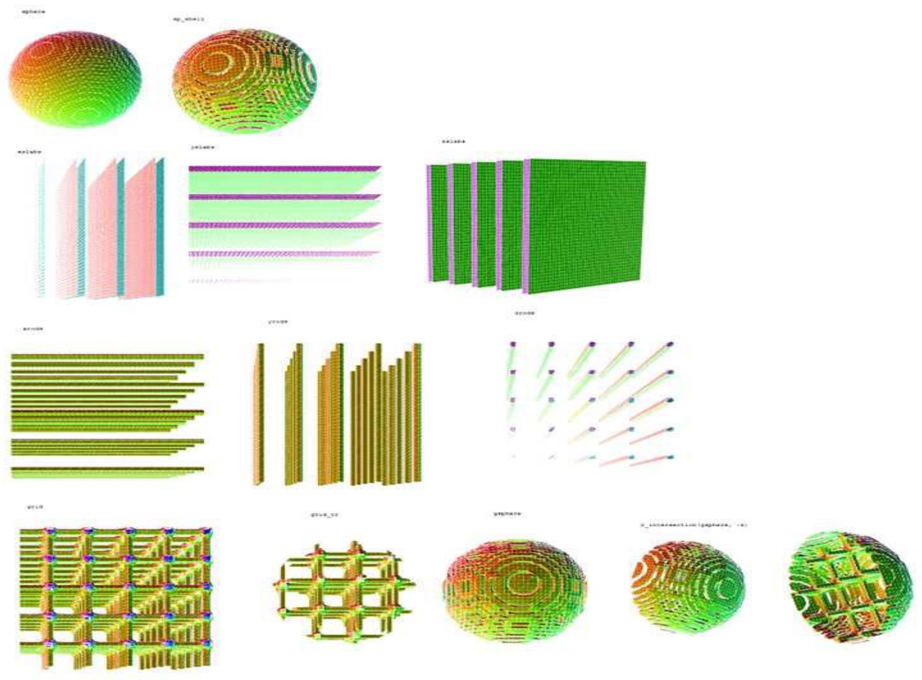


Figure 27: *Complicated hemisphere model showing its modelling phases and then rendered after applying different functions using MC*

5.1.2 WebGL rendering approach

Khronos Group developed and maintained both OpenGL and WebGL in addition to COLLADA. This allows for the creation of a fully integrated and dynamic 3D Web applications based on JavaScript [Khronos].

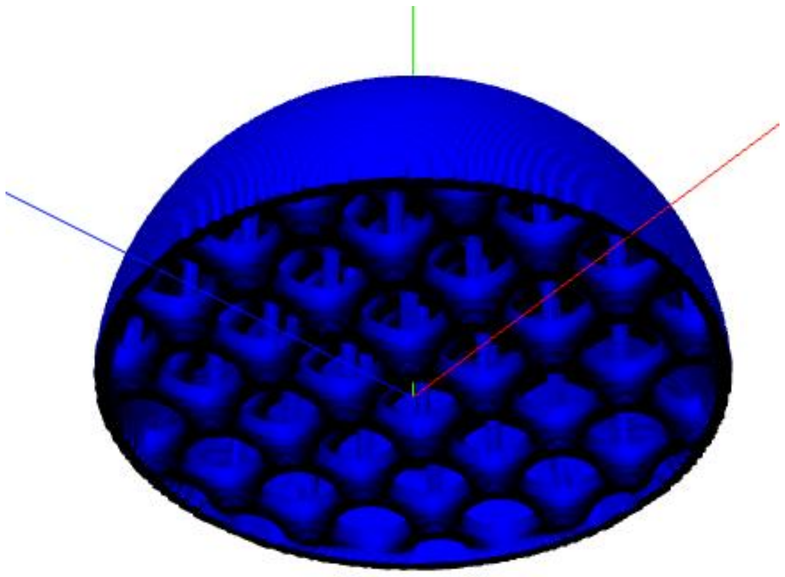


Figure 28: *Hemisphere rendered using WebGL*

In WebGL, both WebGL functions and the MC algorithm are used in the rendering process. The MC algorithm is used to create the volume parameters (the mesh), and WebGL does the rendering using predefined functions loaded from a special JavaScript engine (Three.js) to display the scene and to animate the model. The result is a high-resolution object with precise features (Fig. 28). The main difference between MC and WebGL is that MC rendering uses wire mesh materials to obtain flat wired polygons with low resolution, while WebGL uses "Lambert Material" with the ability to adjust the size of the mesh to get different resolutions. WebGL allows for rendering at the client's GPU with high power and generates heterogeneous models with considerable resolutions in such a way:

- Using MC the mesh is created on the server and sent to the client to be visualised, while in WebGL, the model is sent to the client, where the mesh is created.

- WebGL uses the Binary Space Partition Tree (BSP tree), which is used before rendering the scene to cast shadows and to remove hidden surfaces. It also uses Constructive Solid Geometry (csg.js) with Three.js JavaScript engine.

5.1.3 Ray marching (SDF) rendering approach

Ray marching using signed distance field or SDF acquired a huge advantages over mesh-based using MC and WebGL rendering approaches in terms of performance and runtime. Ray marching is the most efficient way to render signed distance fields. [Zollhofer *et al*., 2015] as shown in Figure 29.

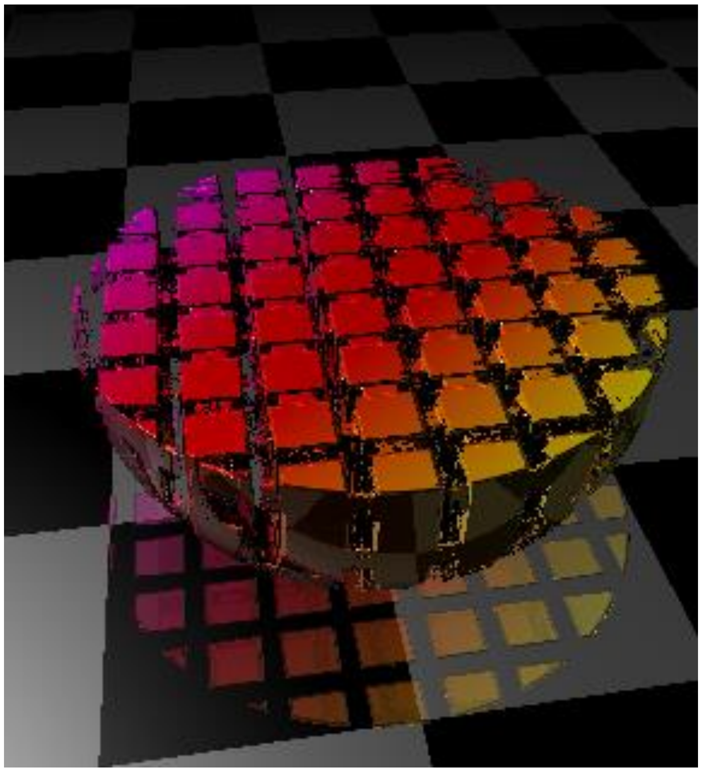


Figure 29: *Hemi-sphere rendered using ray marching of SDF*

5.1.4 OpenGL rendering approach using C++

OpenGL is a software interface built from different libraries and designed to operate and access graphics hardware or the graphics process units (GPUs). Since it is a hardware-independent interface, it can be implemented on many hardware platforms. It is a server-based rendering approach depending on C++ compilers, which uses the OpenGL commands and functions to do the rendering (Fig. 30). OpenGL depends on the MC algorithm and is designed to work on desktop applications using C language unlike WebGL which is designed to run on the Web-browser using JavaScript language.

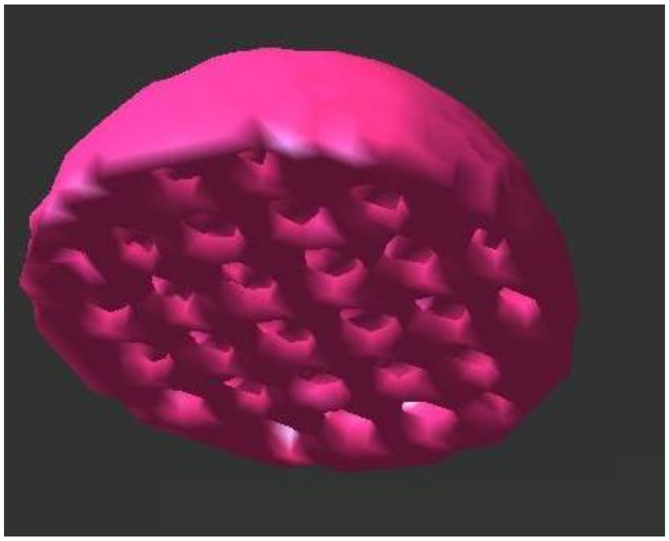


Figure 30: *Hemisphere rendered using OpenGL (C++) at the server-side*

5.1.5 Image slides

Server-based rendering is one of the rendering approaches used in delivering 3D models to low recourses clients. The server renders the model, and start to take sequential snapshots of the model from different angels. The server sends to users a stream 3D images to construct the 3D model as an image slides. The server makes a 35 by 35 snapshots of the rendered object to constitute a 3D image slider with 1225 images (Fig. 31). The image stream may reach up to 122500 KB supposing that each image is of 100 KB. The slider loads an image in the slider and stores the next one in the clients' memory, when the user requests the second image, the third one will be loaded to the buffer (memory) and so on. In that way, we reduce the memory needed to launch the slider to 200 KB only. Each image a dimension of   914 * 415 pixels, the horizontal and vertical resolution is 96 dpi and the colour information stored in the image (bit depth) used is 24.

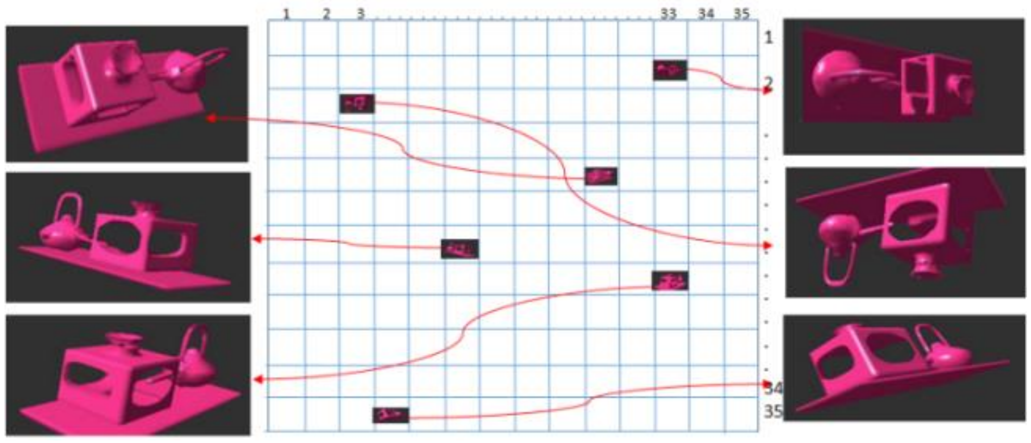
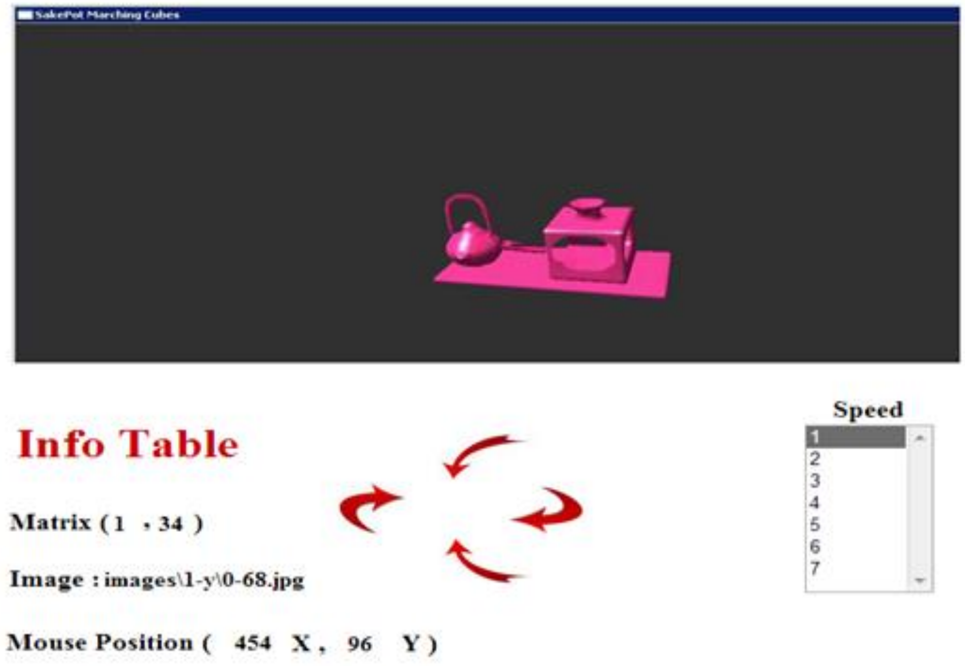


Figure 31: *Image slider matrix (35 by 35)*



Matrix (1 , 34)

Image : images\1-y\0-68.jpg

Mouse Position (454 X , 96 Y)

Figure 32: *Image slider with rotation angles and speed control*

Figure 31 describes the axis of rotation following the longitude and latitude concept. The model can rotate over the x and y-axis, each cell in the matrix refers to one of the images loaded in the memory at the client side and given an index composed of x and y, a cell of (x, y) represents an image taken from a certain angle with certain longitude and latitude. The user can view the 3D object from all angles, and can change the direction of the slider by clicking on the arrows, or by moving the mouse on the rotating model. The client can determine the rotation speed of the slider depending on the available bandwidth and the available resources (Fig. 32).

5.2 Heterogeneous volumes with attributes

Heterogeneous volumes are volumes with different densities. In our work, we focused on rendering heterogeneous volumes using different rendering approaches. Attributes were added at the modelling level where modelling functions are modified and colouring attributes were set to modelling functions as parameters. Several tests using both WebGL and MC rendering approaches were applied to generate heterogeneous volumes with colour attributes. Heterogeneous volumes were generated with different densities represented by colours. Both Leaf and Node colouring approaches, as explained in Chapter III, were used and showed no major difference in rendering time and efficiency. Leaf colouring is easy to use but does not allow multi-colouring for the same heterogeneous sub-model as shown in Fig 33 (a) and (d). Each part of the model can be coloured with one and only one colour. This method is easy to be implemented and used and is dedicated to simple models with few coloured parts. Node colouring is a multi-colouring technique; it allows colouring heterogeneous volumes (with internal structure colouring). Multi-colouring can be applied to the whole volume and even part of the volume. This approach is hard to implement but more useful as it allows generating heterogeneous volumes with real densities and colours, it also allows internal structure colouring as shown in Fig 33 (b) and (c).

Both colouring approaches were put into the test. First we generate a complicated model with leaf as shown in Fig 33, and we compared the time taken to render the coloured model with the rendering time of the same model without colouring and we came out with a conclusion that no time difference was noticed. We then applied the node colouring technique in order to colour the internal structure of the complicated model. We measured the rendering time before and after colouring, and the result was interesting as no major difference was detected.

And we came up with a conclusion that colouring complicated models with different densities do not affect the rendering time of the model.

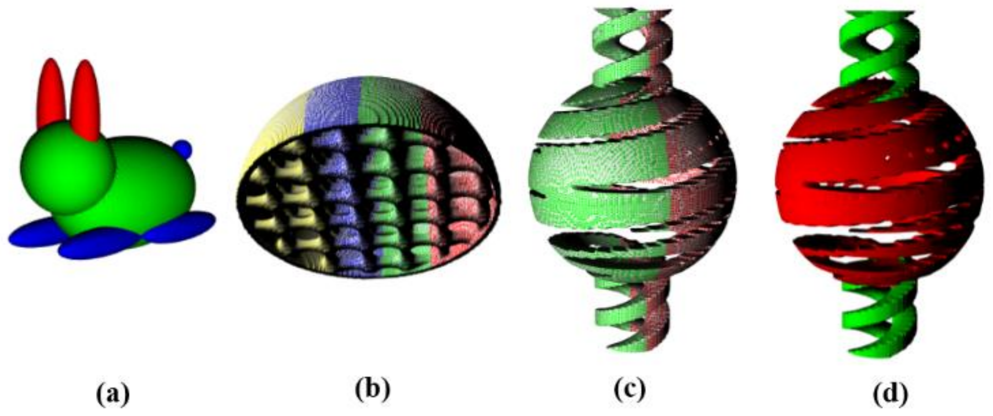


Figure 33: *Heterogeneous volumes with colour attributes generated using different rendering approaches: Leaf a, d; and Node b, c*

Node colouring is more efficient to use, especially when dealing with multi-coloured heterogeneous volumes having internal structures. We implemented a heterogeneous volume with different densities (Fig 34), and we applied the node colouring algorithm as shown in Fig. 35. Node colouring takes place at the modelling level while trying to construct the volume by applying different modelling functions. When the modelling phase is completed, we applied the rendering phase, and we measured the rendering time before and after applying the three colours, the result showed no difference in rendering time, then we made a cross-section to show the internal structure of the volume, and how node colouring can apply colours to the internal structure of the volume (Fig. 34). Different cross-sections were made along the Y and the Z-axis. The experiment and internal structure colouring was clearly shown.

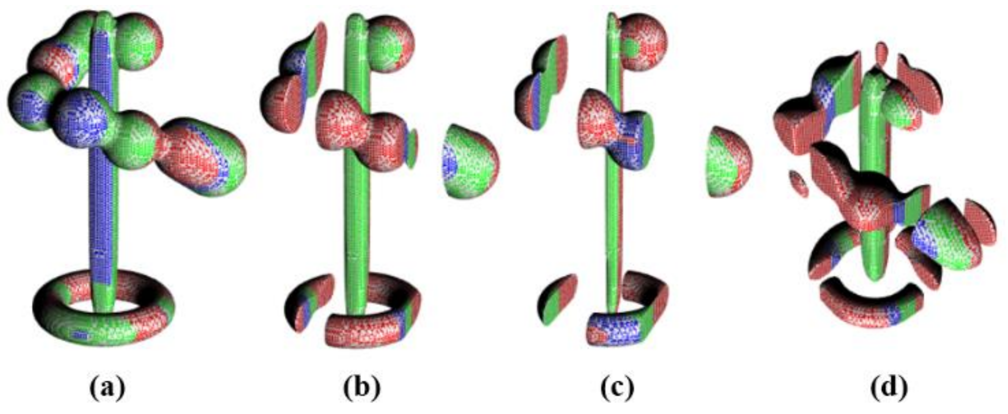


Figure 34: *A cross-section for a complicated heterogeneous volume with colour attributes showing the internal structure of the model*

```
dX =  Math.pow(px,2) ;
dY =  Math.pow(py,2) ;
dZ = Math.pow(pz,2) ;
cylz = hfCylinderZ(px,py,pz,cx,cy,cz,0.548) ;
cylx = hfCylinderX(px,py,pz,cx,cy,cz,0.316) ;
cyl =  r_union(cylz,cylx) ;
cl(1,0x00ff00);
spcyl = hfSphere(px,py,pz,cx,cy,cz,0.994) ;
spcyl =  r_union(spcyl,cyl) ;
cl(2,0x0000ff);
hole = hfCylinderY(px,py,pz,cx,cy,cz,0.916) ;
inside =  r_subtraction(spcyl,hole) ;
cl(3,0xff0000);
cube = 1.9 - dX*dX-dY*dY-dZ*dZ ;
blend = hfBlendInt(cube,inside,0.8,0.2,0.3) ;
my_model = blend ;
```

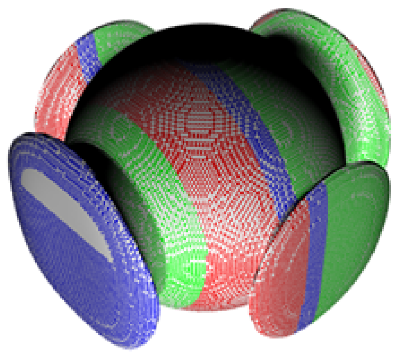


Figure 35: *Heterogeneous volume with colour attributes being modelled in JavaScript and rendered using WRMR*

The example in Figure 35 shows different colouring steps, colouring is applied during the modelling process and not at the end of it. Node colouring can be applied after each operation and different colours can be applied for the same model taking into consideration colouring its internal structure as well. Cylinders were created using "hfCylynder" function which is an HF function written in JS using the HFtoJS converter, The below example code shows that the first colouring step took place right after applying the "r_union" operation between two cylinders (node colouring). The result is a coloured cylinders after applying the union operation. The second colouring step took place right after joining the sphere with the union of the two cylinders. And the last colouring operation happened after the subtraction operation and right before the blending operation.

5.2.1 Primitive model colouring approach (Leaf colouring)

Primitive colouring or leave colouring as mentions above is an approach to colour a whole part of a model with one colour, different parts of a model can be coloured with one colour each. This approach does not take into consideration colouring the internal structure of the volume. The code below is written in JavaScript using WebGL THREE.js library and is used to generate primitive multi-coloured models using primitive colouring approach. We used the "THREE.Geometry" function to create three different sub-volumes, each created sub-volume has its own modelling functions and is subjected to a series of modelling operations. When the modelling process finishes and just before rendering, colouring operation is applied to each sub-volume to assign one single colour to each using the "outputmesh" function as shown in the algorithm below.

```
var geometry0 = new THREE.Geometry();
var geometry1 = new THREE.Geometry();
var geometry2 = new THREE.Geometry();
geometry0 = mcalgo(values , points, geometry0);
geometry1 = mcalgo(values2 , points, geometry1);
geometry2 = mcalgo(values3 , points, geometry2);
var materialNormal = new THREE.MeshNormalMaterial()
var material = new THREE.MeshNormalMaterial( { colour: 0x00ff00,
shading: THREE.FlatShading } );
var obj1 = new THREE.Mesh( geometry0 );
outputmesh(geometry0, 0xff0000);
outputmesh(geometry1, 0x00ff00);
outputmesh(geometry2, 0x0000ff);
```

5.2.2 Heterogeneous volume colouring approach (Node colouring)

Heterogeneous volume colouring or node colouring takes place during the modelling process and not at the end of it. This approach can be applied to both simple and complicated volumes. The algorithm below is written in JavaScript code and is used to generate heterogeneous multi-coloured volume. It is clear that the colouring function "cl" is used just after the operation function "r_union" in the middle of the modelling process. Different colours can be assigned to the same volume or sub-volume, the result is a heterogeneous multi-coloured volume with coloured internal structure as shown in the algorithm below.

```
spi5 =  r_union(spi3,spi4)  ;
cl(spi5,0xff0000);
spi6 =  r_union(spi1,spi2)  ;
spi6 =  r_subtraction(spi6,sph2)  ;
cl(spi6,0x00ff00);
my_model  =   r_union(spi5,spi6)  ;
```

5.3 WRMR implementation

5.3.1 WRMR tasks and functions

The proposed environment is composed of two different major parts, the modelling part to the left and the rendering and visualisation part to the right as shown in Figure 36. Using the modelling part, the user can write, edit, save and delete HF functions encoded in JavaScript language using an interpreter, which is located in the left part of the environment. The environment also supports ready-to-use models and clients can use them instantly. Rendering and visualisation are located in the right part of the environment. Instant rendering is available by allowing direct and immediate transformation of HF functions encoded in JavaScript language to be rendered and visualised. The process of rendering and visualising starts by reading the functions written in the interpreter, integrating these functions with the core of the environment, then applying the selected rendering approach and displaying the rendered volume on the client's screen. Models can be loaded from outside the environment using .js or .txt file formats, and they can be saved back as functions or as models using .OBJ and .X3D files. This allows for the model exchange between users over the Web in the form of HF functions written in JavaScript language or in the form of a polygonal mesh.

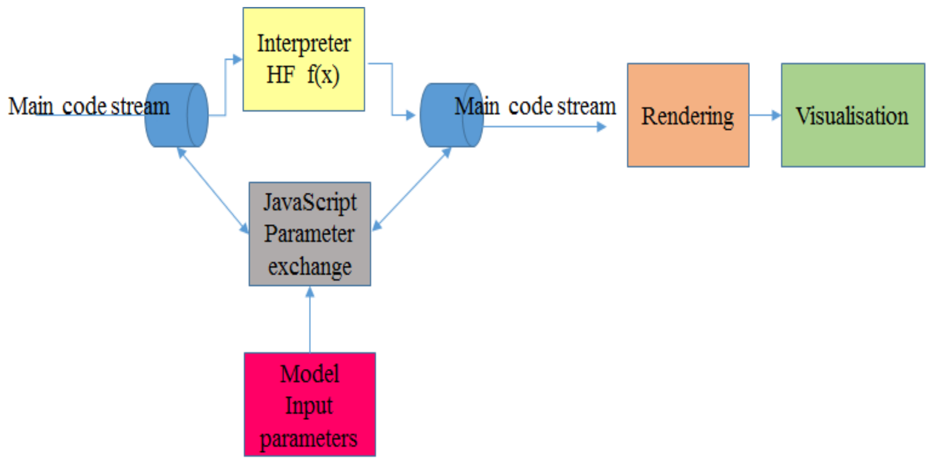


Figure 36: *Set of background tasks showing how functions and attributes being transformed and processed in WRMR background*

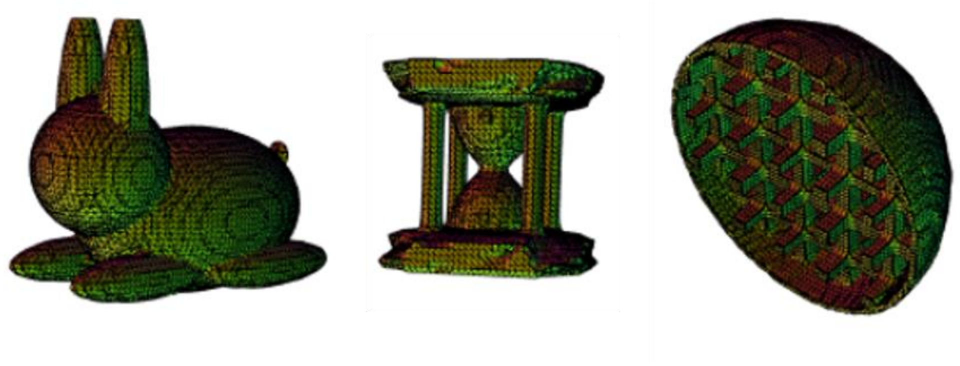
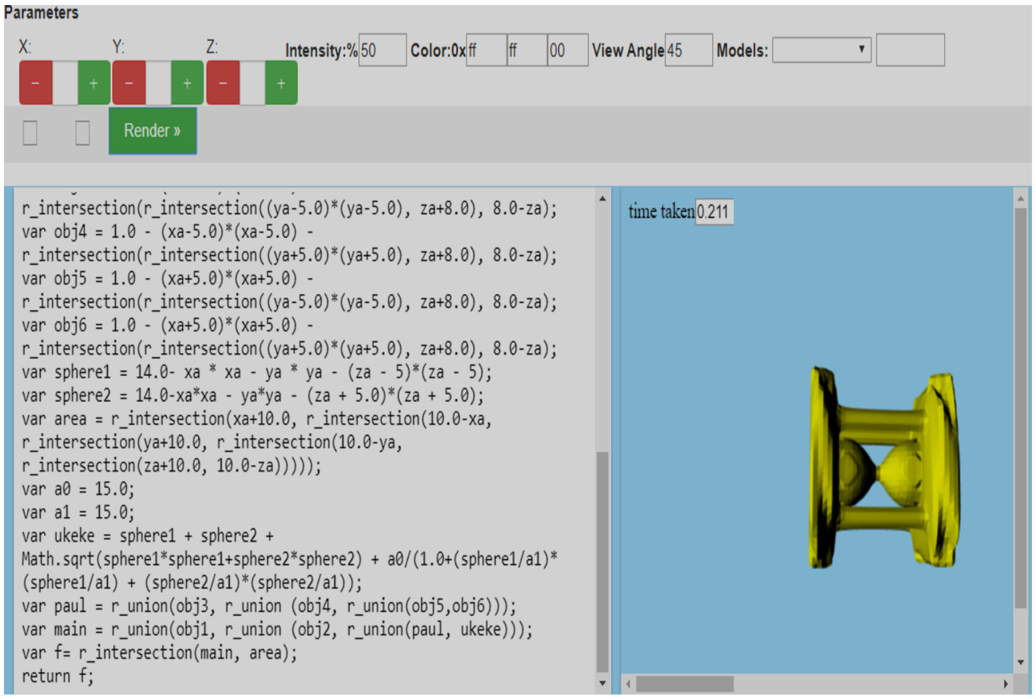


Figure 37: *Components of the proposed environment: left part (Interpreter) and the right part (rendering and visualisation)*

The WRMR interface is divided into two major parts: the interpreter and visualiser. The interpreter is the place where the user writes the JavaScript code, while the visualiser is the part where rendering and visualising take place as shown in Figure 38. A set of HF functions and attributes were re-written using JavaScript functions and take place in the WRMR background. When starting the front-end applications, a set of configuration and setting-up code takes place and is saved up at the client's memory in a stream called Main Code Stream (or

MCS). MCS is then edited by the interpreter, when HF functions are written by the client, and by the JavaScript Parameter (JSP) controller supported by the model input controller. The interpreter is responsible for sending the HF functions written in JavaScript to the MCS, while the JSP is responsible for setting up attributes and model's characteristics and sending them to MCS. In general, MCS contains the environment set-up code in addition to the code of the HF function written in JavaScript in addition to model attributes code. All are lined up and sent to the rendering module, which consists of three different client-side rendering sub-modules, and one server-side rendering sub-module. When the rendering is completed, the visualisation module takes action and displays the volume being rendered on the client's browser.



```
r_intersection(r_intersection((ya-5.0)*(ya-5.0), za+8.0), 8.0-za);
var obj4 = 1.0 - (xa-5.0)*(xa-5.0) -
r_intersection(r_intersection((ya+5.0)*(ya+5.0), za+8.0), 8.0-za);
var obj5 = 1.0 - (xa+5.0)*(xa+5.0) -
r_intersection(r_intersection((ya-5.0)*(ya-5.0), za+8.0), 8.0-za);
var obj6 = 1.0 - (xa+5.0)*(xa+5.0) -
r_intersection(r_intersection((ya+5.0)*(ya+5.0), za+8.0), 8.0-za);
var sphere1 = 14.0- xa * xa - ya * ya - (za - 5)*(za - 5);
var sphere2 = 14.0-xa*xa - ya*ya - (za + 5.0)*(za + 5.0);
var area = r_intersection(xa+10.0, r_intersection(10.0-xa,
r_intersection(ya+10.0, r_intersection(10.0-ya,
r_intersection(za+10.0, 10.0-za)))));
var a0 = 15.0;
var a1 = 15.0;
var ukeke = sphere1 + sphere2 +
Math.sqrt(sphere1*sphere1+sphere2*sphere2) + a0/(1.0+(sphere1/a1)*
(sphere1/a1) + (sphere2/a1)*(sphere2/a1));
var paul = r_union(obj3, r_union (obj4, r_union(obj5,obj6)));
var main = r_union(obj1, r_union (obj2, r_union(paul, ukeke)));
var f= r_intersection(main, area);
return f;
```

Figure 38: *Heterogeneous volumes modelled and rendered on the client-side using MC algorithm*

Heterogeneous volumes with different densities, types, and nature can be generated using the proposed environment. Different rendering approaches can generate different types of volumes. Server-based rendering generates a stream of images in the form of image slides, where client-base rendering generates interactive volumes with different densities as shown in Figures 37-41.

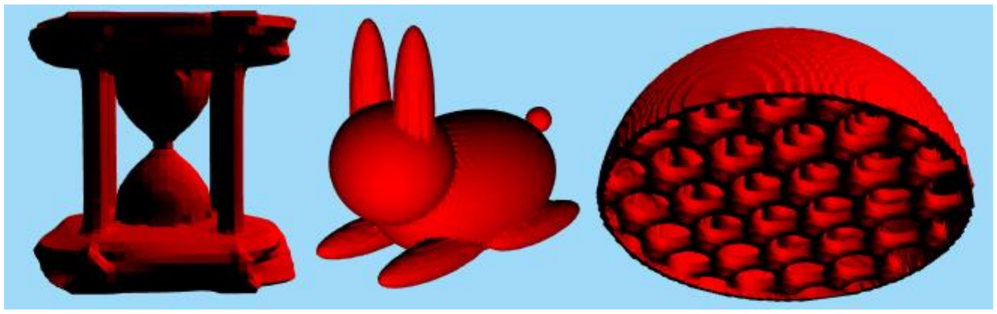


Figure 39: *Heterogeneous volumes modelled and rendered on the client-side using WebGL*

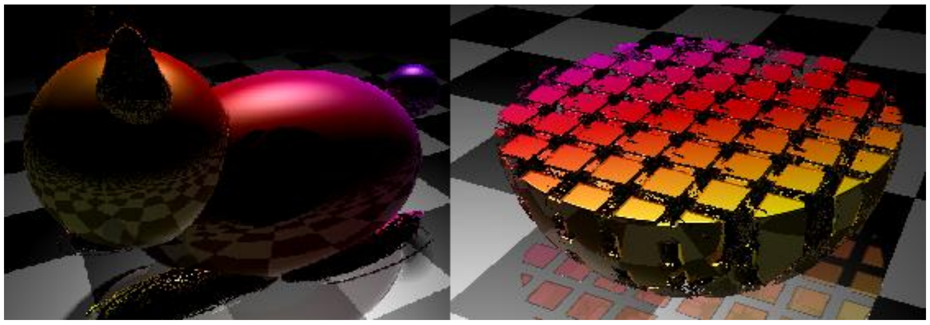


Figure 40: *Client-side ray marching rendering (SDF) for heterogeneous volumes*

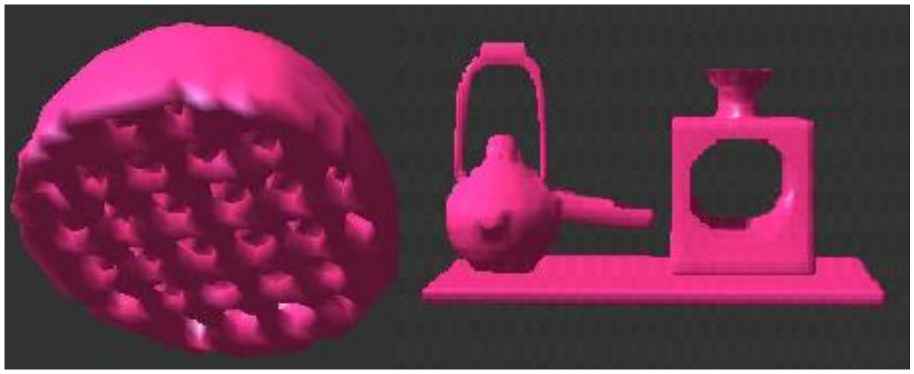


Figure 41: *Server-side rendering for heterogeneous volumes using OpenGL*

5.3.2 HyperFun to JavaScript converter (HFtoJS).

HFtoJS converter as discussed in 4.4.1 is responsible to convert HF functions to JavaScript functions. WRMR is designed to render JavaScript (JS) functions rather than HF ones. Users may not be familiar with transformation process from

HF to JS. That is why we implemented an HFtoJS converter which is composed of two parts, the left, and right parts.

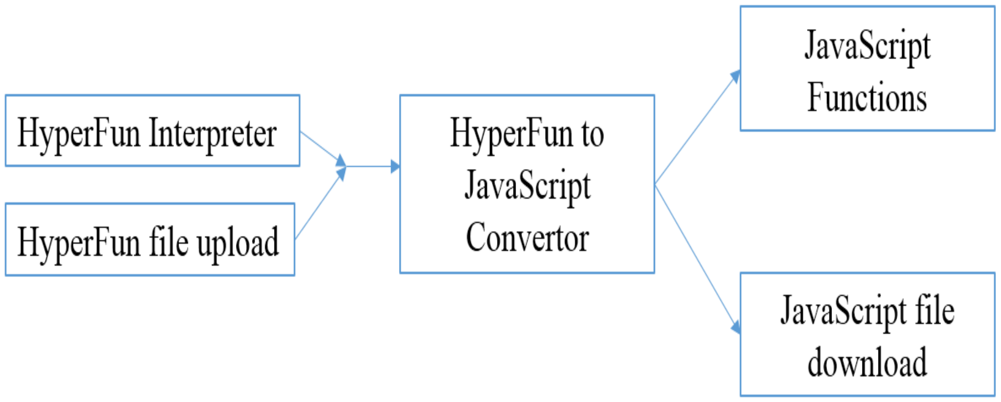
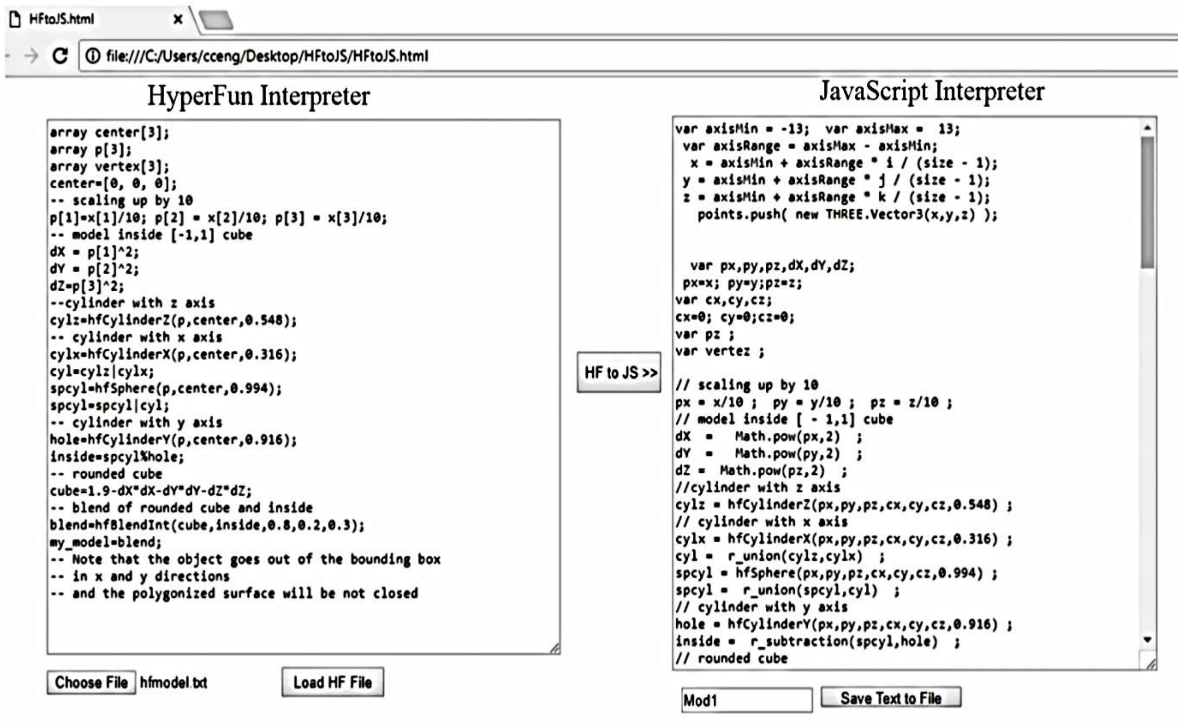


Figure 42: *HF to JavaScript converter main parts*



```
array center[3];
array p[3];
array vertex[3];
center=[0, 0, 0];
-- scaling up by 10
p[1]=x[1]/10; p[2] = x[2]/10; p[3] = x[3]/10;
-- model inside [-1,1] cube
dX = p[1]^2;
dY = p[2]^2;
dZ=p[3]^2;
--cylinder with z axis
cylz=hfCylinderZ(p,center,0.548);
-- cylinder with x axis
cylx=hfCylinderX(p,center,0.316);
cyl=cylz|cylx;
spcyl=hfSphere(p,center,0.994);
spcyl=spcyl|cyl;
-- cylinder with y axis
hole=hfCylinderY(p,center,0.916);
inside=spcyl%hole;
-- rounded cube
cube=1.9-dX*dX-dY*dY-dZ*dZ;
-- blend of rounded cube and inside
blend=hfBlendInt(cube,inside,0.8,0.2,0.3);
my_model=blend;
-- Note that the object goes out of the bounding box
-- in x and y directions
-- and the polygonized surface will be not closed
```

```
var axisMin = -13;  var axisMax =  13;
 var axisRange = axisMax - axisMin;
  x = axisMin + axisRange * i / (size - 1);
 y = axisMin + axisRange * j / (size - 1);
 z = axisMin + axisRange * k / (size - 1);
   points.push( new THREE.Vector3(x,y,z) );


  var px,py,pz,dX,dY,dZ;
 px=x; py=y;pz=z;
var cx,cy,cz;
cx=0; cy=0;cz=0;
var pz ;
var vertez ;

// scaling up by 10
px = x/10 ;  py = y/10 ;  pz = z/10 ;
// model inside [ - 1,1] cube
dX  =   Math.pow(px,2)  ;
dY  =   Math.pow(py,2)  ;
dZ =  Math.pow(pz,2)  ;
//cylinder with z axis
cylz = hfCylinderZ(px,py,pz,cx,cy,cz,0.548) ;
// cylinder with x axis
cylx = hfCylinderX(px,py,pz,cx,cy,cz,0.316) ;
cyl =  r_union(cylz,cylx)  ;
spcyl = hfSphere(px,py,pz,cx,cy,cz,0.994) ;
spcyl =  r_union(spcyl,cyl)  ;
// cylinder with y axis
hole = hfCylinderY(px,py,pz,cx,cy,cz,0.916) ;
inside =  r_subtraction(spcyl,hole)  ;
// rounded cube
```

Figure 43: *Online HyperFun to JavaScript converter*

The converter was developed using JavaScript code, and it allows HF users to write HF functions on the left side of the converter, or can simply upload HF files using the file management task located in the left part as well. The written HF functions and the loaded functions are then processed and sent to the right part of the converter, where a series of syntax and lexical analysis steps take place.

The output of this process is a pure JavaScript code loaded into the right side and can be saved into a JavaScript file (Fig. 43).

5.4 Case studies

In order to make use of all the above-mentioned rendering approaches, while focusing on delivering the models (rendering them) to the client according to his needs, three different case studies are being discussed and implemented. The first case to study is instant collaboration. The reason for selecting such a case to study is to explore the advantages and disadvantages of approaching instant collaboration methods and techniques and to measure the performance at the end-users. The second case to study is about parametrised models, where attributes such as colour and textures can be passed to the model at the modelling stage. The purpose is to develop an environment that is able to handle complicated models with different densities and apply different colouring attributes at different levels. The third case study to discuss is partial rendering or partial visualizing. We will try to investigate how to partially display the model in order to protect it and to reduce the rendering and visualizing time at the client-side.

5.4.1 Case study 1: Instant collaboration

The concept of collaboration does not only mean that models should be available for sharing and editing. In our case, the concept of collaboration should give us the ability to modify a model at one client and to notice the updates made on the model all over WRMR. Changing the model parameters at one screen should directly affect the model attributes at other clients, so that, the model at other clients will be updated and changed as soon as the initial one is modified in real-time and even before saving it. In that way, all other clients can see the updates simultaneously. When a client finishes editing a model, the modified model will be updated, saved and redistributed all over WRMR for all other clients. The purpose of this case study is to allow clients to share instant modelling and rendering simultaneously and instantly. Different tests were made while trying to measure the performance of instant collaboration. We measured the transmission time as well the editing time of the model over the Web at the client-side.

5.4.2 Case study 2: Parameterised modelling and rendering

So far, our tested models are delivered in a static way, the image slider in Figure 32 shows some parameters to control the rotation of the object over the slider. Volumes with attributes can be part of the WRMR system as it is designed to be a dynamic environment, where 3D models are subjected to changes and editing. Parametric features can be applied to enrich WRMR with powerful tools that help in dynamic modelling and allow for accessing the model and changing its parameters before rendering it. The change of parameters can be applied at both functions and meshes, making full use of the HRep concepts using forward-backward (BFRep-FBRep) transformation discussed in Chapter III. Heterogeneous volumes with colour attributes were introduced and two different approaches for colouring were discussed in section 5.2. Colouring attributes were approached from two different prospective, Node and leaf. Node colouring is limited to colour a whole part of the volume, while leaf colouring allows to multi-colour one single part, including its internal structure. We applied multi-colouring attributes to heterogeneous volumes and results showed that it is possible to colour the external part of the volume as well as its internal structure. Different volumes with colour attributes were put into the test to measure the performance and efficiency. Colouring complicated volumes showed no major difference in rendering time. The tests assured that colouring the internal structure of the model is possible.

5.4.3 Case study 3: Partially rendering

Rendering the whole object and displaying it on the client's screen makes it vulnerable for code stealing. As an attempt to protect the model, it will be useful to do partial rendering at the client-side, so that the client will be able to render a part of the model and not the whole one. The rendered part of the model will be the one facing the camera, and only that part of the model will be displayed on the screen. When the user navigates or moves the model, a new mesh will be sent to the client to be rendered and displayed showing the new desired part of the model. This case study helps in protecting the model from piracy, but could be very expensive regarding the required time and bandwidth and should be subjected to deep investigation and researching. One of our approaches focused on sending images rendered at the server-side to the clients. The stream of images could be large and need high internet speed and large storage as images are sent in large numbers to cover the model from all angles. To avoid this bottleneck conflict at the server-side, we displayed one image on the client screen, and load another one in the memory buffer, when the client requests to see the next image, the buffered image is displayed in his browser and a new image is loaded to the buffer and so on. Using this technique allows users with low internet and small storage to view large volumes sequentially. To test this approach, we developed a prototype, which is based on a considerable number of images generated at the server-side and sent to the client. The stream of images was controlled and monitored in a way, no more than two images can be sent to the client at the same time. One to be displayed and the other to be stored in the buffer. When the user requests to see the buffered image, it will be moved to the screen and a new image will be sent to the buffer. We measured the time taken to load the whole images to the buffer and found out that to load the whole model we need a considerable amount of memory and time. While when loading one image in the buffer at a time, the model goes slower, but with convincing loading time and a small amount of memory.

5.5 Summary

In this chapter we discussed the implementation phase of the environment. WRMR was composed of three major parts including input, process, and output. The implementation included four different rendering approaches taking into consideration the colour attributes. Three different case studies were presented and discussed. The chapter ended up with the front-end implementation, which described the implemented part of the environment and its user interface in addition to the HyperFun to JavaScript converter.

Chapter 6

Experiments, tests and results

In this chapter, we put WRMR into the test to measure the performance and to validate the novelty of this work. Hybrid representation HRep will be tested by implementing different types of models using both FRep and BRep. Hperfun to JavaScript converter will be tested and we will measure both performance and validity.

6.1 Rendering experiments

The purpose of the following set of experiments is to determine the efficiency and usability of the proposed Hybrid Representation HRep. A series of different experiments took place to test the performance of the different rendering approaches and to compare results using different scenarios. We start by experimenting the four different rendering approaches (MC, WebGL, signed distance fields and server-based rendering using C++) as mentioned in Chapter IV. Testing these approaches using a single machine first running a Chrome Web browser with a high-speed network connection to the server then we move to apply these tests on different hardware platforms. The aim of these tests is to check the rendering time of different volumes (simple and complicated) and compare different rendering approaches. The MC algorithm written in JavaScript code was applied on the client-side using polygonisation. WebGL rendering is on the client-side too. OpenGL rendering using C++ code is on the server-side, where 3D objects are transformed into still images and sent to the client. Heterogeneous volumes with attributes were also considered using both polygonisation and ray-casting. Attributes such as colours are calculated and displayed for each point. We introduced two types of heterogeneous volume colouring. The first one is the leaf or primitive colouring which allows multi-model colouring before combining them into a complicated one. The other one is the node or complex colouring, which usually takes place at the mesh level.

The still images are put together to perform a 3D image slider to enable the user to view the object from different angles. For that purpose, the server generates a series of images that constitute a bi-dimensional matrix. As the client browses the object from different angles using the mouse, the loaded images started to change on the slider, rotating around different axes, and the adjacent images start to load in the buffer as mentioned in Chapter IV.

Models with different complexities were used in our second test (see Figures 47-49). We focused on the rendering time as shown in Table 3. We used rendering with low resolution on the client machine in order to achieve a good performance rate, this caused clear visible sharp edges and lower quality (Fig 44(a), 45(a), 46(a)). In order to increase the resolution, more computational resources will be needed on the client-side (Fig 44(b), 45(b), 46(b)).

Using simple models, no major differences in timing were detected while applying different rendering approaches. However, as the complexity of the object increased, the difference in timings started to appear. In general, for the same initial conditions, the best timings were achieved in case of rendering purely on a server-side and sending the result as images to the client. Thus, the need for re-rendering the model at every change in camera position makes it hard to work with the model in an interactive manner.

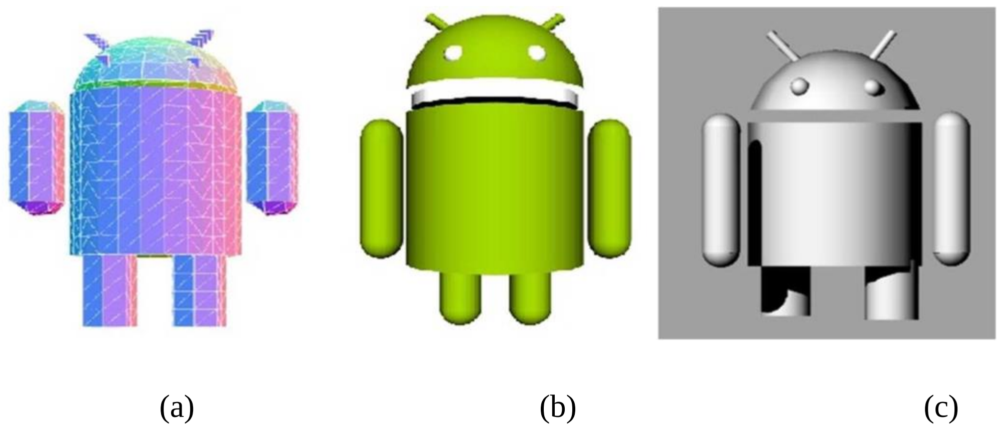
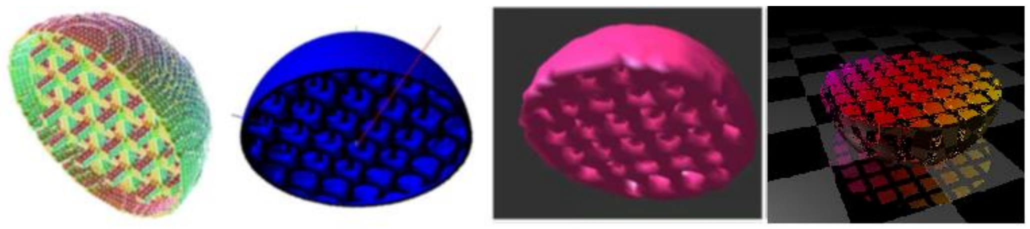
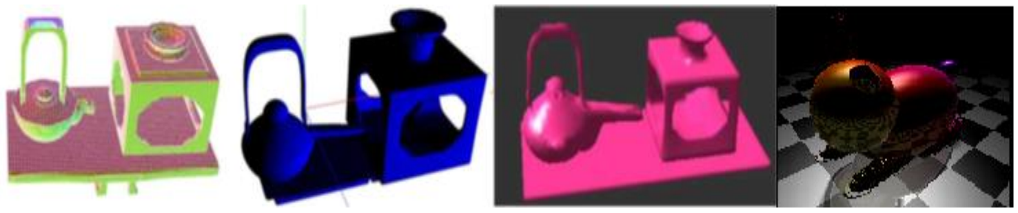


Figure 44: *Android Robots rendered using different approaches: MC (a), WebGL (b), Server-side using C++ (c)*



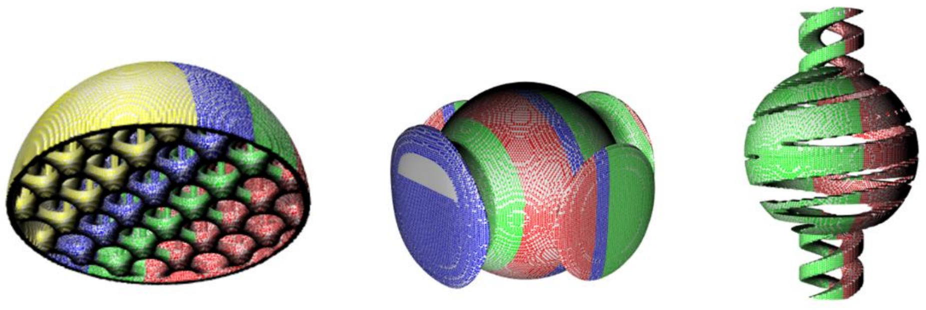
(a) (b) (c) (d)

Figure 45: *Hemisphere models: MC (a), WebGL (b), Server-side (C++) (c) and SDF (d)*



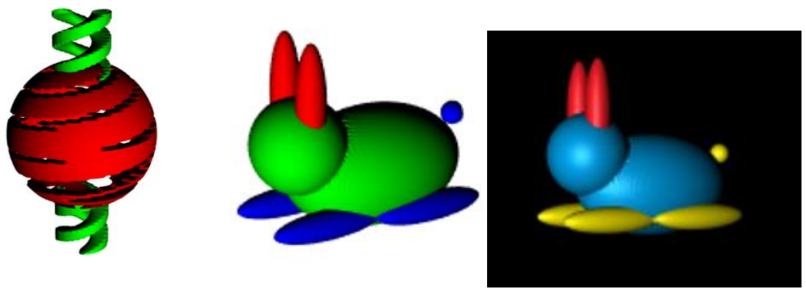
(a) (b) (c) (d)

Figure 46: *Complicated models: MC (a), WebGL (b), Server-side using C++ (c), SDF (d)*



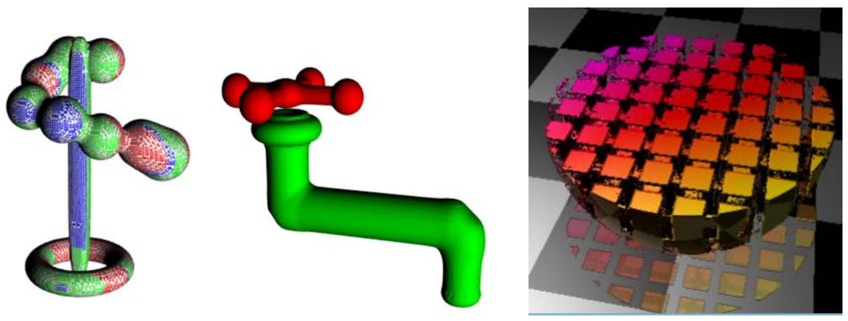
(a) (b) (c)

Figure 47: *Complicated models: Hemi-MC (a), Infinity-MC (b), Doubl helix-MC(c)*



(a) (b) (c)

Figure 48: *Complicated models: Double helix-MC (a), Rabbit-WebGL (b), Rabbit-SDF (c)*



(a) (b) (c)

Figure 49: *Complicated models: Spiral-MC (a),* Faucet- *WebGL (b), Hemi-SDF (c)*

Rendering simple objects using different approaches was really quick and took little time, the rendering time started to rise, when we rendered more complicated objects. It can be noticed that the amount of time and GPU power started to rise up, when we applied WebGL with high-resolution to complicated objects. As a result, MC can be applied on low GPU power and the time needed for rendering is less ten times less than the time needed for WebGL rendering. Clients could use WebGL, if they were supported with considerable GPU performance. Ray-marching using SDF can be applied using two different ways, the first one is by using the HF functions directly which can be very quick, and the other one is by using the mesh transferred from the server, which allows 3D data extraction for faces and vertices. In our case, both ways were used and put into test and showed very little time when rendering complicated models, but it still needs good computational resources. Table 3 shows the time needed in milliseconds for each approach in order to render 3D object.

3D Objects	MC	WebGL	SDF	Server Rendering
Rabbit	0.012	0.009	0.004	0.0009
Faucet	0.017	0.13	0.006	0.0011
Double Helix	0.143	1.539	0.0512	0.0011
Infinity	0.144	1.556	0.0529	0.0012
Spiral	0.431	3.25	0.165	0.0016
Sand	0.202	1.842	0.0653	0.0013
Android Robot	0.021	0.196	0.0152	0.178
Hemisphere	0.168	1.782	0.0535	0.0012
Sake Pot	0.507	4.597	0.2563	0.0018

Table 3: *Comparing different rendering approaches (MC, WebGL using Three.js, and SDF on the client side and server-side rendering using OpenGL)*

A quick analysis for Table 3 shows that the rabbit shows the smallest rendering time in all rendering techniques. The construction of the rabbit model is simple and no complicated functions were applied. Models with more complicated functions started to show higher rendering time, the spiral and sake pot models for example need more complicated function than other used models and thus they showed more rendering time using MC and WebGL.

6.2 Comparing rendering techniques using different parameters

Experiments were applied to the four different rendering techniques using WRMR, simple and complicated heterogeneous volumes were modelled, rendered and filed using data extraction (OBJ and X3D). Different models were examined and measured in terms of rendering time, number of vertices and faces, data file size, and model complexity.

We put into test each of the above rendering approaches, then measured different parameters related to two different types of models (simple and complicated). The parameters are data model extraction time, No. of vertices, No. of faces, file loading time and file size.

6.2.1 Marching Cubes experiments

We put into test the MC rendering approach, and we measured the above mentioned parameters on two types of models, the simple and the complicated ones as shown in Table 4.

Table 4 shows the following:

- Rendering time difference between the three objects is minor
- The number of extracted faces and vertices for the two complicated objects are approximately the same and they are as double as those for the simple objects
- The OBJ loading time for all objects is relatively acceptable and less than 0.6 sec.
- Simple volumes rendered using MC and generating polygonal meshes have small file size (less than 100 Kbytes), while complicated volumes have big size files (more than 1.5 Mbytes)
- X3D file loading time is faster than OBJ file
- Both X3D and OBJ files have approximately same size.

Table 4 shows that the number of faces and vertices started to rise with more complicated models. The loading time is also affected with the level of model complications. Faucet model, with average complication compared to the other available models, showed average rendering time, average number of faces, and vertices and average OBJ loading time. This clearly shows that the rendering time, number of faces and vertices and the OBJ loading time is directly proportional to the model level of complication

Rendering using MC						
	Hemisphere	Sake Pot	android	Double Helix	Faucet	Infinity
Data extraction time in sec	0.372	0.271	0.112	0.296	0.213	0.195
No. of Vertices	52501	55649	2827	50365	42865	35685
No. of Faces	27122	27818	1362	25135	21585	18352
Total	79623	83467	4189	75500	64450	54037
Loading .OBJ File in sec	0.540	0.601	0.114	0.521	0.425	0.384
OBJ File Size in KB	1707	1875	86	1685	1525	1369
loading X3D File	0.001	0.001	0.001	0.001	0.001	0.001
X3D File Size in KB	1976	2158	96	1895	1698	1522

Table 4: *Experiment applied to three different objects using MC rendering*

The bar chart shows the variation of extraction time between 0.112 for simple object and 0.372 as max for the hemisphere complicated object (Fig 50). This variation indicates that the extraction process could be very fast and with no major difference between simple and complicated objects when applying MC rendering. As to the number of the extracted faces and vertices, Figure 50 clearly shows that the more complicated the object is, the more vertices and faces to be extracted, this will require more extraction time.

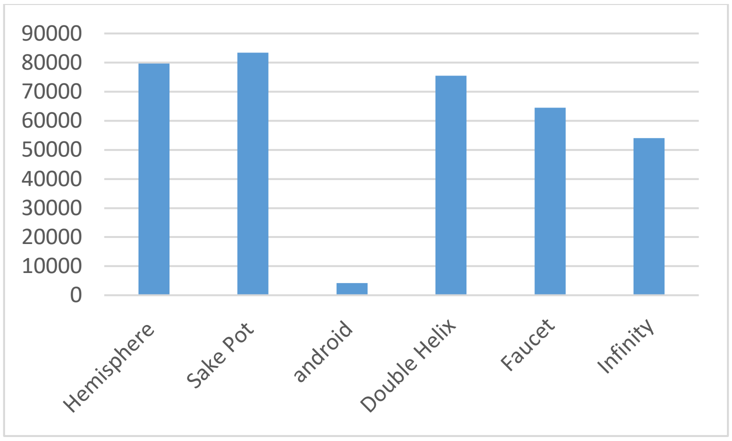


Figure 50: *Graphical representation for the extracted values, comparing different objects*

6.2.2 WebGL experiments

3D complicated volumes were tested by changing resolution parameter, six different phases were applied for six different resolutions. From the results shown in Table 5 we can conclude the followings:

1- The loading time is increasingly changing while changing the resolution of the extracted objects. When the density goes higher, more faces and vertices are needed and therefore more loading time will be needed to display the model. Suppose the normal model got a 100% density, in this experiment we tried to render the same model with different densities. Loading the hemisphere model with only 30% of its original density (distorted surface) took 0.178 sec, rising the resolution up to 80% of its original density (close enough to its normal shape) took 1.707 sec, which means ten times increase.

2- The number of extracted faces and vertices increased while going up with the resolution of the model and therefore the file size increased.

3- Table 5 shows that as complicated as the 3D object goes, the more extraction time is needed, and more faces and vertices to be extracted, therefore, more storage space needed to save the OBJ files.

4- Figure 51 shows a big difference in extraction time between two different complicated objects, as the resolution goes higher, the extraction time increases.

5- Figure 52 compares the OBJ file sizes, the more complicated the model is, the bigger is the OBJ file size. In our test, more than 5 Mbytes were needed to store a hemisphere model rendered using 80% of its original density (close enough to its normal shape) in an OBJ file. It needed 1.5 Mbytes to store sake pot model at the same resolution

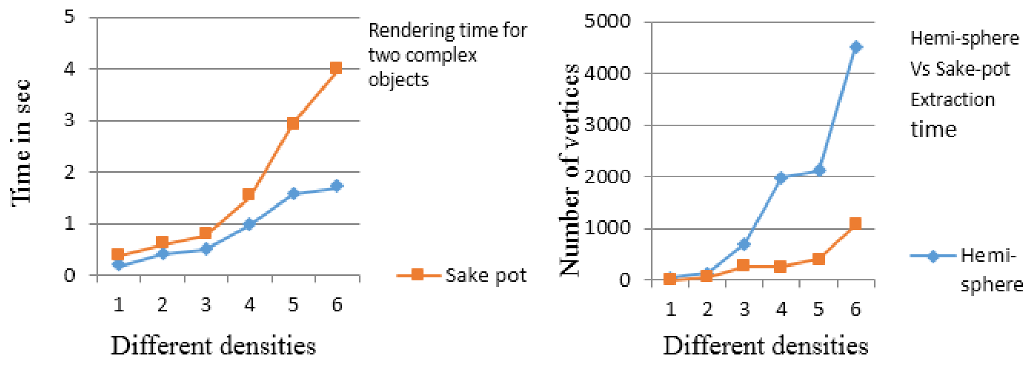


Figure 51: *Graphical representation for the extracted values, comparing two different objects*

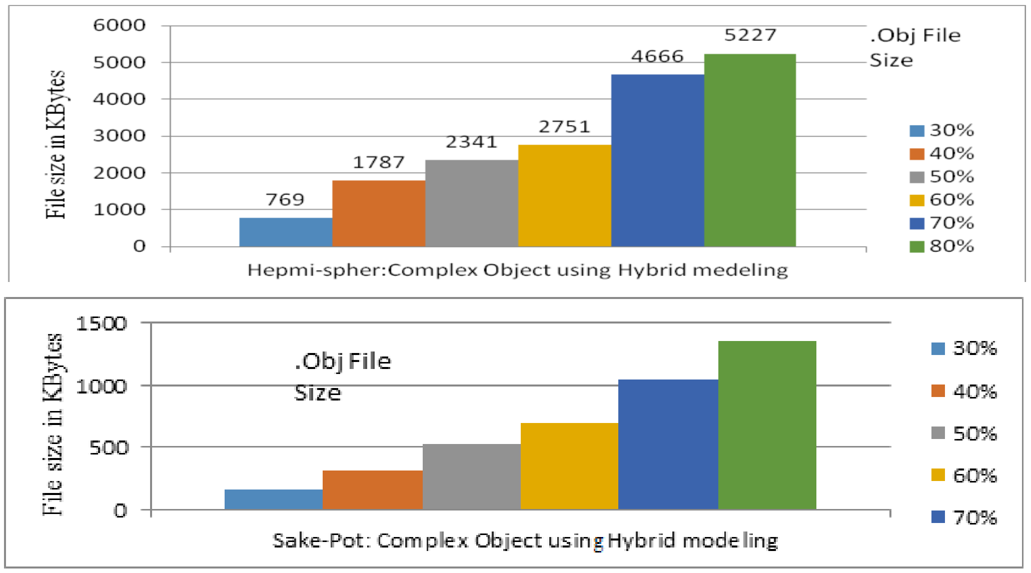


Figure 52: *Bar chart representation of the extracted values, comparing two different Objects*

Density	Load time in sec	No. of Vertices	Vertices Extraction Time	No. of Faces	Faces Extraction Time	Total extraction Time	.Obj Size in KB
Hemisphere: Complicated Object using hybrid modelling							
30%	0.178	12868	25.775	12868	17.189	42.964	769
40%	0.401	12868	60.006	12868	73.319	133.325	1787
50%	0.49	38160	419.981	38160	285.74	705.721	2341
60%	0.977	55648	1076.842	55648	918.586	1995.428	2751
70%	1.574	73568	3584.225	73568	2544.588	2128.813	4666
80%	1.707	94584	8544.011	94584	6585.005	4529.016	5227
Sake-Pot: Complicated Object using hybrid modelling							
30%	0.191	2636	5.7	2636	2.997	8.697	157
40%	0.203	7848	39.002	7848	21.513	60.515	322
50%	0.285	16460	180.2	16460	90.034	270.234	531
60%	0.541	28468	136.8	28468	121.641	258.441	699
70%	1.359	37492	124.859	37492	280.196	405.055	1053
80%	2.259	40488	1140.211	40488	1140.211	1080.422	1355

Table 5: *Experiment applied on three different objects using WebGL rendering*

MC and WEbGL are both used to demonstrate the forward conversion from FRep to BRep at the rendering level in the proposed HRep environment.

6.2.3 Ray marching using signed distance fields experiments

Experiments on rendering with SDF were applied and results in Table 6 showed the following results:

- Rendering time difference among the different objects is small.

- The number of faces and vertices for complicated volumes are much more than those for simple volumes.
- The rendering time is considered acceptable for all models as all rendering processes showed values less than 1 sec that reveals the low time required to render models with SDF.

Rendering using SDF						
	Hemisphere	Sake Pot	Android	Sand	infinity	Double Helix
Rendering time /s	0.51	0.38	0.1592	0.31	0.42	0.49
No. of Vertices	58534	57692	3128	54256	39586	58474
No. of Faces	29139	29813	1652	27158	20015	28547
Total	87673	87505	4780	81414	59601	87021

Table 6: *Experiment applied on three different objects using signed distance fields*

Table 6 shows SDF models performance, it is clearly shown that the rendering time is relatively low for all models with different complication level. The number of extracted vertices and faces is relatively close as well. SDF is used to demonstrate the backward conversion from BRep to FRep in the proposed HRep environment.

6.2.4 Server based data extraction and delivery

OpenGL is used for rendering complicated volumes at the server side to generate a stream of images taken from different angles and with fixed size. This process is dedicated to serve simple users with no GPU power. The number of generated images determines the precision of the model. We put into test eight different image-shot rates which is the time taken between two generated images, we started from two till nine, the number of generated images is maximum at the rate two and minimum at the rate nine. In Table 7, we can find that at the rate two, the image matrix contains 2025 images (45 by 45 images). When the image rate goes up to nine, the image matrix became 484 images (22 by 22 images).

	Matrix size	No. of images	Time taken	Images/sec	Image size\Kbytes	Total size
2	45*44	2025	198	10.23	100	197.76
3	42*42	1764	224	7.88	100	172.27
4	38*38	1444	274	5.27	100	141.01
5	35*35	1125	160	7.03	100	109.86
6	32*32	1024	141	6.57	100	87.26
7	28*28	784	126	6.22	100	76.56
8	25*25	625	112	5.58	100	61.03
9	22*22	484	104	4.65	100	47.27

Table 7: *Experiment applied on sake-pot object using server based rendering*

From Table 7, we can conclude that the time needed to generate a stream of images is directly proportional to the used snapshot image rate. The storage needed to save images is related to the size of the image stream. When the image matrix shrinks (snap-shot is 9), the model precision is reduced and thus low storage is needed

6.3 Comparing different scenarios

Different scenarios with three major parameters were implemented as discussed earlier, these parameters are internet bandwidth, hardware and Web-browser. Different experiments on different scenarios were put into test to examine the environment performance and adaptivity. The experiments will examine the bandwidth change, the hardware change and the Web-browser over different rendering techniques, the rendering time, which is the time taken to do the rendering of different kinds of heterogeneous models using different rendering techniques. It will be a good indicator whether the rendering technique is good to be used by the users.

6.3.1 Internet bandwidth constraint

In this experiment, we put into test the four rendering approaches while setting the Internet bandwidth as changeable parameter.

	0.5 MB	1 MB	2 MB	4 MB	8 MB	14 MB
WebGL						
Double Helix 50	1.874	1.08	0.93	1.07	2.48	1.038
Double Helix 100	1.767	1.31	1.25	1.24	4.23	1.33
Infinity 50	1.11	0.77	0.81	0.82	0.79	0.83
Infinity 100	1.75	1.22	1.22	1.31	1.34	1.26
Spiral 50	2.05	1.93	1.95	1.88	2.158	2.07
Spiral 100	2.83	2.56	2.67	2.20	2.71	2.69
MC						
Double Helix	1.31	1.19	1.08	1.05	1.17	1.08
Infinity	1.36	1.19	1.19	1.17	1.24	1.17
Spiral	0.98	0.97	0.95	0.99	0.93	0.99
SDF						
Infinity	0.96	1.13	1.04	1.06	1.22	1.15
Spiral	0.90	0.88	0.88	0.87	0.80	0.80

Table 8: *Internet bandwidth change experiment, showing the rendering time in seconds for each rendering technique on a certain machine.*

The experiment takes place on a machine running a core I5 5300U CPU with 2.3 GHz 2.9 GHz and a 4 GB of installed memory, the used GPU is an Intel HD graphics 5500 with a core speed of 300-950 MHz and 64-128 bit bus, the browser is a 64-bit Chrome version 67.0.3396.99. This experiment requires a change on Internet bandwidth (uploads and downloads). For that reason, we installed Bandwidth Shaper Net-Limiter version 4.0.36.0 to control the changes of Internet download and upload speed. Table 8 shows that three rendering techniques were put into test, and in WebGL rendering, we applied different resolutions on different heterogeneous complicated models.

The time taken for each model using different rendering methods reveals the following (Table 8):

- Maximum rendering time at 0.5 MB bandwidth is 2.83 sec when rendering spiral using WebGL, while minimum rendering time at 14 MB bandwidth is 0.476 sec when rendering spiral using SDF.

- The rendering time does not change significantly, when using a certain rendering approach to rendering different complicated models.

- SDF showed the best rendering practice, when rendering with different bandwidths.

- Rendering with SDF showed that the difference between the minimum time that is 0.77 sec at 4MB and the maximum time, which is 0.9 sec, is minor and less than 0.1 of a sec.

The results shown above (Table 8) validate the functionality of the system using HRep by applying all the available rendering techniques that suit different kinds of users successfully and within a convincing rendering time even for low bandwidth users.

6.3.2 Hardware constraint

The purpose of this experiment is to examine the performance of the environment when using different hardware resources. It was conducted after putting into test four different machines with different hardware specs and after neglecting the Internet bandwidth factor by connecting all the machines over a local area network to the same server.

The test was applied to four different hardware machines with different CPU, RAM and GPU powers, we called then M1, M2, M3 and, M4 consequently and had the following hardware specs:

- M1: Core I5 5300U CPU: 2.3 GHz 2.9 GHz Memory: 4 GB GPU: Intel HD Graphics 5500

- M2: Intel Core i7 6700 CPU @3.40GHz Memory 16 GB 64 bit Windows 7 GPU Inter® HD Graphics 530

- M3: Intel ® Core ™ 2 Duo CPU E4600 @ 2.4Ghz Memory: 3 GB 32 bit Window 7

- M4: Intel® Core ™ 2 Duo CPU E4600 @2.4 GHz Memory: 2 GB 64Bit OS windows 7 GPU: Intel® G33/G31 Express Chipset Family

The hardware constraint test using Table 9 showed the following:

- Different models with different densities got different rendering time when using WebGL rendering approach. The fastest rendering time was obtained when using the highest GPU.

- Heterogeneous volumes failed to be rendered when using WebGL or SDF rendering approaches over a low hardware device (M4 in our case).

- Heterogeneous volumes rendering using MC showed different rendering time, when rendered over different hardware platforms, the quickest rendering obtained when rendering with the highest GPU.

- The complexity of the rendered volume showed no difference when using MC rendering approach, it can be used even on low hardware recourses.

- Rendering with SDF approach over different machines showed a considerable difference in rendering speed, and it failed to load on a very low hardware platform (M4).

Rendering Approaches	Machines	M1	M2	M3	M4
WebGL	Double Helix 50	0.33 sec	0.4 sec	1.41 sec	-
	Double Helix 100	0.89 sec	0.95 sec	4.1 sec	-
	Faucet 50	0.32 sec	0.36 sec	0.76 sec	-
	Faucet 100	0.55 sec	0.56 sec	3.70 sec	-
	Spiral 50	0.39 sec	0.56 sec	1.59 sec	-
	Spiral 100	1.15 sec	1.29 sec	5.31 sec	-
MC	Double Helix	0.42 sec	0.49 sec	2.12 sec	1.3 sec
	Faucet	0.39 sec	0.64 sec	1.66sec	0.98 sec
	Spiral	0.39 sec	0.43 sec	1.72 sec	2.51 sec
SDF	Faucet	0.23 sec	0.35sec	1.82 sec	-
	Spiral	0.528 sec	0.55 sec	1.31 sec	-

Table 9: *Rendering complicated models using WebGL rendering approach running on four different machines with different hardware resources*

6.3.3 Browser constraint

Clients with browsers not supporting graphics can also access the environment and make use of its services by depending on its servers to do the rendering using OpenGL. Clients will receive a stream of images to reconstruct the model. Table 10 shows how the flow of images is affected by bandwidth change. One Megabyte of download capacity is enough to load 10 images, which is enough to see the model from a good angle of view.

In this test, we did the rendering on the server side using C++, then send a stream of images to a moderate hardware recourses client. The only variable parameter is the Internet bandwidth, which was managed and controlled by the bandwidth shaper net-limiter version 4.0.36.0. The bandwidth was segmented into six parts and the stream of images was sent to the user over each bandwidth.

Download speed	256 KB/sec	512 KB/sec	1 MB/sec	1.5 MB/sec	3 MB/sec	6 B/sec
No. of Images/s	2.56	5.12	10.24	15.3	30.72	61.44
Time Taken	13.8	6.83	3.41	2.28	1.14	0.57

Table 10: *Server-based rendering and sending images to clients in the form of image-slides*

Table 10 showed the following results:

- A minimum bandwidth of 1 MB is needed to launch a slider in a considerable time
- Users with 6 GB RAM can easily get the slider in less than 0.6 sec.
- The hardware resources are negligible with no effect in this experiment.
- Rendering time at the server side is negligible with no effect on the overall image transmission process.

6.4 WRMR testing

6.4.1 WRMR online random users.

Rendering using WRMR was put into the test by asking random users from different places to access the environment. Those users had different internet upload-download speeds and were using different hardware platforms including desktops, laptops, pads, and smartphones. They were asked to implement and run three different models as shown in Figure 49.

User	Internet speed	Hardware	Infinity (MC)	Faucet (WebGL)	Hemi (SDF)
User 1	1 MB/2 MB	Samsung Galaxy S7	6.8	-	-
User 2	2 MB/4 MB	Iphpne 6	4.2	12.6	-
User 3	4 MB/8 MB	HP Pavilion X360	0.3	0.7	0.9
User 4	4 MB/8 MB	Galaxy Book S	6.1	15.3	-
User 5	2 MB/6 MB	iPad Pro	3.8	9.4	24 .2
User 6	1 MB/2 MB	HP ProDesk 400	0.6	1.2	1.3
User 7	2 MB/4 MB	HP Elite Desktop	0.8	1.4	1.5
User 8	2 MB/4 MB	Huawei Y9 Prime	7.1	-	-
User 9	4 MB/8 MB	HP ProDesk 400	0.4	1.1	1.3
User 10	8 MB/16 MB	HP Pavilion X360	0.9	1.8	1.5

Table 11: Random users with random internet speed and hardware

The hardware specs for all users were as follows:

User 1: Samsung Galaxy S7, 5.1-inch, quad-HD display,4GB of RAM
User 2: Iphpne 6, Dual-core 1.4 GHz Typhoon (ARM v8-based),PowerVR, GX6450 (quad-core graphics),1 GB RAM
User 3: HP Pavilion X360,Core i5,16GB,Intel Integrated HD Graphics 520
User 4: Galaxy Book S, Snapdragon™ 8cx processor,8GB RAM
User 5: iPad Pro, A12X Bionic processor, Neural Engine, M12 coprocessor, 256GB
User 6:HP ProDesk 400 G5, Core™ i7 processor (i7-8700),32 GB DDR4, Integrated: Intel® UHD Graphics 630

User 7: HP Elite Desktop, Core i5 3.2GHz, 8GB RAM, AMD Radeon™ , graphics2,4

User 8: Huawei Y9 Prime, 4GB RAM, touchscreen, 16M colours.

User 9: HP ProDesk 400 G5,Core™ i5 processor, 32 GB DDR4, Integrated: Intel® UHD Graphics 630

User 10: HP Pavilion X360, Core i3,8GB,Intel Integrated HD Graphics 520

Table 11 shows that all users were able to access the environment and make use of at least one of the available rendering approaches. It clearly shows that some mobile phones failed to do rendering using WebGL and SDF, this was one of the weaknesses of the environment, future research should focus on allowing low hardware resources to access and do modelling and rendering using WRMR. The rendering time using different rendering approaches is considerably high with some mobile and PAD devices. Rendering time should be reduced to make it feasible for the user to use the environment

6.4.2 Comparing WRMR with other existing online systems

WRMR is an online environment for 3D modelling and rendering, one of the important characteristics is combining two well know representations into a hybrid one making use of both characteristics. This makes it so different from other existing online 3D systems. Tinkercad for example (https://www.tinkercad.com/dashboard) is an online modelling system, it is limited to primitive or predefined objects only, while WRMR can develop any complicated volume from scratch. Tinkercad allows users to drag and drop primitive objects and combine them together, while WRMR allows writing modelling function, and save those functions for later use. ShapeJS is another online modelling system (http://shapejs.shapeways.com/ide), it is equipped with an interpreter for writing JS code and supported with instant rendering and visualization. WRMR supports all the above-mentioned characteristics, with one major difference, which is allowing HF users to convert their desktop models into JS models using an HFtoJS converter. This tool is considered a huge advantage over ShapeJS. Uformit (http://uform.co/) uses grid-based computation for low hardware resources, WRMR uses different rendering approaches to achieve a better goal and allow different users with different hardware resources to do online modelling.

6.5 Hyperfun to JavaScript converter (HFtoJS).

HFtoJS is dedicated to HF users who are not familiar with scripting languages especially JavaScript. The purpose is to allow those users to take their desktop models into the Web without doing additional efforts. Here comes the novelty of developing such a converter to make it easy for HF users to do online modelling without paying attention to the scripting language. In order to demonstrate the novelty of this work, we tested the converter by allowing different users to write their functions using HyperFun language, and then run them on WRMR using HFtoJS converter.

Model	Pass/ Fail	Comments
Model 1	Pass	No modification needed
Model 2	Pass	Model needed modification
Model 3	Fail	-
Model 4	Pass	No modification needed
Model 5	Pass	-
Model 6	Fail	No modification needed
Model 7	Pass	Model needed modification
Model 8	Pass	Model needed modification
Model 9	Pass	Model needed modification
Model 10	Pass	Model needed modification

Table 12: Testing HyperFun to JavaScript converter.

The process of conversion from HF to JS was quite good. Ten different users were asked to do ten different HF models and try to run them using WRMR with the HFtoJS converter, the results were shown in Table 12. Only two out of ten models failed to convert their models and the other eight were passed. Five out of ten needed modification or adjustments to suit the conversion process and three out of ten needed no further modifications at all. From the result we can conclude that HFtoJS converter works fine but with certain limitations including the way the user is presenting his HF code, the types of declared variables, the lines, breaks and spaces used, and other special symbols. HFtoJS converter needs more investigation, modification, and implementations to suit different ways of HF codes, functions, and variables.

6.6 Summary

In this chapter, we demonstrated the novelty of this work by testing the usefulness and functionality of the system. We put into test different rendering approaches, and we compared the results using different inputs. These rendering approaches were as follows: Marching cubes, WebGl, and Signed distance fields. Different scenarios were tested and a deep comparison was made using the following constraints: internet bandwidth, hardware, and browser. Heterogeneous modelling with colour attributes was tested and showed promising results in terms of colouring the interior structure of the model as well as its exterior. We also tested the proposed Hybrid Representation HRep and demonstrate its usefulness. We also showed the importance of the HyperFun to JavaScript converter (HFtoJS) by testing it and determining its strength and weakness.

Chapter 7

Conclusion

7.1 Summary

Internet is developing rapidly, and the demand on the Internet resources is getting bigger. On the other hand, Web-browsers started to be suitable environments for 3D modelling, and new rendering tools and techniques appear. All these facts reflect a need, which is taking shape modelling into online interactive level using the rapid progress of Internet and Web-browsers. Such an interactive environment should be adaptive and flexible, should contain all possible rendering tools and modelling techniques, all together to produce solid and easy to use applications based on what is called an adaptive and interactive 3D shape modelling environment (WRMR).

In this work, we focused on three major points and tried to confirm and validate their importance to the proposed system. We introduced the concept of hybrid representation HRep, by combining two different representations, Function and Boundary, in a new Hybrid one. We also introduced, discussed, and implemented Heterogeneous volumes with attributes based on the proposed hybrid representation and we proposed two different colouring approaches for that purpose. We also implemented a Hyperfun to JavaScript converter (HFtoJS) in-order to take HF desktop models over the Web using a converter responsible to convert HF code into JS code suitable for online modelling.

We have proposed a 3D Web-based modelling and rendering environment, we described its architecture as a client-server environment, and we proposed different scenarios by tring to solve the constraint satisfaction problem and we dealt with the creation of heterogeneous volumes with attributes. We implemented the proposed environment after discussing the design and all the parts and modules necessary to make it functional. Security concepts were taken into consideration, and some security modules were discussed and implements to provide a certain level of protection. We put the environment into tests by experimenting with the different rendering approaches and by applying different scenarios. This document discussed the basic concepts of such an environment and tried to focus on three essential elements, namely: Networking, Modelling and Rendering. It came up with different scenarios to predict the client asking for the service and to select the best way of the service. The document also presented a core engine (in JavaScript) responsible for information gathering about the clients to help in decision-making. We outlined some advantages and disadvantages of both FRep and BRep and we discussed a Hybrid Representation behaviour and integration to improve modelling results. The document ended up with real examples of the concept regarding the rendering techniques, where four different rendering techniques were implemented (MC, WebGL, SDF and server based C++); three of them were on the client side, and one on the server side. Simple 3D objects were created using these rendering techniques and some basic recognition were taken regarding these techniques. Three different case studies were proposed to be researched, discussed and implemented.

Starting with networking, we have considered the features of a client-server WRMR to establish the most convenient and efficient way of 3D Web-based modelling and rendering with the particular emphasis on the latter. We described the specifics of interactive client-server architecture for modelling and rendering and identified four of the most common scenarios for executing those processes along with necessary communication and decision-making using WRMR.

Four different rendering techniques were implemented in order to explore the characteristics of the proposed WRMR. These techniques include server-based rendering based on C++ code, where still images are sent to the client. Client side rendering is using WebGL. Client side polygonisation using MC is implemented as JavaScript code and client side rendering using SDF as well. First, we showed that rather simple objects can be successfully rendered using different techniques as a proof of concept. Then complicated models were rendered using the four different methods.

We spotted three different threats and tried to implement security solutions to overcome them. After introducing the adaptive environment, we discussed its scenarios and different rendering approaches available for complicated heterogeneous volume objects. Since the environment is Web-based, it was necessary to implement a specialised algorithm to secure the code and protect it. A special obfuscation algorithm was introduced to make it really hard for others to break the code. Authentication and authorisation modules were implemented to protect the environment from illegal access and prevent non-authorised users to have full access to it. Two different access rights were granted to two different kinds of users. Finally, we tried to protect the models themselves by hiding the functions and raw data (the mesh) from non-authorised users. We implemented special modules to extract the mesh (vertices and faces) and saved them in an OBJ file. Ordinary users can load the file and access the model without having any access to the functions and data. The model became accessible but well protected against piracy. Some of the conclusions that can be noted are:

1) A hybrid representation can be achieved by mixing two available representations (Function and Boundary), and make use of both characteristics.

2) Gathering information about the client requesting the service is a key factor in determining what kind of service (i.e., rendering technique) to deliver. The collected information helps WRMR in decision making, a core engine running in the background in a continuous manner detects any critical changes that may happen at the client's machine.

3) Different scenarios should be considered, the document introduced four different and essential ones. More scenarios can be considered, enabling WRMR to improve decision-making. The proposed environment allows for reliable and efficient rendering process referring to the four discussed scenarios.

4) Different rendering approaches can be applied to both simple (no attributes) and heterogeneous volume objects. The main issue regarding

both objects is the GPU power consumption and the time needed to do the rendering. Simple objects proved to use low GPU power in a short time, while heterogeneous volume models (complicated objects) showed high GPU power consumption and considerable rendering time.

5) Rendering at the server side using C++ implementation can be very efficient in terms of processing power and rendering time. However, the problem is that objects are delivered to clients as still images. Image slides could be a good solution, where a series of continuous snap-shot images are delivered to the client to constitute a 3D sliding object that can be seen from different angles. The client needs a moderate Internet bandwidth to load the slider.

6) Rendering using MC is very efficient in both rendering time and GPU power consumption. It takes place at the client side. Both simple and complicated objects can be rendered in a short time. This method best suits clients with low GPU power and it delivers low-resolution objects only.

7) Rendering using WebGL is suitable for high-resolution objects and proved to be the optimal solution on machines with high GPU power. Rendering takes place at the client-side and it consumes the client’s GPU power to its limit. In general, this process requires a considerable rendering time that varies according to the object resolution.

8) Rendering using SDF can be used for heterogeneous volumes and proved to be optimal solution with high GPU power on the client-side. This approach proved to so quick with high resolution.

9) Heterogeneous volumes with attributes proved to be a good solution. They were approached from two different prospective, and showed good results using WRMR.

10) Building up WRMR, which is capable to interact with different kinds of clients, is a challenge and needs further research. More investigations should be done, different scenarios can be added, more rendering

techniques could be introduced, and more complicated objects can be implemented and compared.

11) Obfuscation with a special encryption algorithm can be applied to make it hard for code breaker to break it and change its content, for that reason a special algorithm was applied to generate obfuscated code.

12) Granting access for users can prevent un-authorized users to access critical parts of the environment. Different users can be granted different access rights, which helps in controlling users and monitoring their behaviour.

13) Extracting raw data from the GPU buffer can be done. A special algorithm was implemented to extract vertices and faces and save them to Wavefront OBJ and X3D file formats.

14) Using X3D and OBJ file formats after data extraction can hide the model's functions and data from users. The displayed models may lose some of their characteristics, but it can protect them against piracy.

15) HyperFun models can be converted to JavaScript models suitable for the Web using a special converter.

7.2 Future work

Based on the accomplished studies and experiments the future work will focus on implementing a platform independent environment, supported with a smart core engine enhanced with a scenario optimisation for decision making. For that purpose, all mentioned and new rendering techniques will be integrated and aligned to fit different clients' needs and demands. In order to satisfy these emerging needs the proposed environment requires full adaptation and integration between different rendering techniques acquainted by a solid modelling system, which is supported by a framework for scenario optimisation and protected by a special security and immunity system.

The following recommendations reflecting the advantages and drawbacks of the tested rendering techniques provided by the WRMR can be stated:

- Satisfy the necessary cloud computing infrastructure requirements in order to develop a cloud based WRMR that allows instant information collection and has direct access to heterogeneous volume objects saved in data files for modelling and rendering purposes.

- Implement a scenario optimisation framework that supports WRMR with the best available solution based on predefined scenarios. This framework will be able to combine two or more solutions to provide clients with optimal rendering for better service.

- More experiments on complicated objects should be executed to further analyse the behaviour of these objects in terms of their modelling and rendering within the proposed WRMR.

- Other rendering techniques such as ray casting and volume rendering should be investigated, implemented and compared with existing techniques.

- Implementation of an intelligent engine in the core of WRMR promises more functionality to support decision-making and thus for providing better rendering services.

- Deep investigation, technical implementation, and testing will take place to support the proposed case studies discussed in Section 4: real time collaboration, parameterised modelling and rendering, and partial rendering in order to prove their efficiency and adaptability regarding time and resources.

- Securing WRMR should be investigated to protect the online models and to prevent hackers from changing or reusing the models or accessing the source code.

- More experiments and tests should be done in real cases to ensure security success.

- More types of privileges and access rights should be granted to allow wide variety of users to access the environment.

- 3D models extraction process should be improved to cover different types of 3D file formats.

Future work will include exploration of collaborative Web-based modelling and rendering of heterogeneous objects with a complicated internal structure in the context of a flexible interactive WRMR. It will also include scenario optimisation solutions, which will be able to determine the best solution to be delivered based on the defined scenarios. Different case studies will be researched and developed. Securing both WRMR and the 3D objects will be investigated and implemented to ensure better protection and availability. Implementing a more complicated obfuscation algorithm and granting more access rights to different kinds of users with different security levels. More 3D file formats will be available and the extraction process will be more precise and accurate.

Appendix A

List of publications

Abdallah, A., 2018. Real-Time Heterogeneous Volume Modelling And Rendering Environment. *International Arab Conference On Information Technology (ACIT'2018),* 19, 129-136. DIO: https://doi.org/10.1109/ACIT.2018.8672700

Abdallah, A., Fryazinov, O., Adzhiev, V., Pasko, A., 2014. 3D Web-Based Shape Modelling: Building up an Adaptive Architecture. *ACHI 2014: The Seventh International Conference on Advances in Computer-Human Interactions*, Barcelona, Spain. 96-102. DIO: https://www.thinkmind.org/index.php?view=article&articleid=achi_2014_4_40_20229

Abdallah, A., 2017. 3D Web-Based Shape Modelling: Data Extraction and Delivery. *FASSI 2017, the Third International Conference on Fundamentals and Advances in Software Systems Integration*, Rome, Italy, 2017. Conference paper 5-11. DIO: https://www.thinkmind.org/index.php?view=article&articleid=fassi_2017_1_20_80016.

Abdallah, A., 2017, Securing Online 3D Web-Based Models. *ICT and Societal Challenges*, LAU Beirut / New York, April 2017. DIO: https://www.researchgate.net/publication/316988424_SECURING_ONLINE_3D_WEB-BASED_MODELS_Research_in_Progress

Abdallah, A., Fryazinov, O., Adzhiev, V., Pasko, A., 2019, *Heterogeneous 3D Volumes on the Net: Real-Time Modelling and Rendering, (in progress), journal ready for submission.*

Implementation of the Web-based Real-time Modelling and Rendering Environment (WRMR): http:// 212.98.139.67:82/lmd_online/interactive/online/3denv.html

Appendix B

Complexity Analysis

1) MC generic algorithm in terms of loops and statements

```
Loop3()
          Loop2 ()
                    Loop1 () {
                       <<list of sequence statements>>
                 }
```

The Big-O notation for MC algorithm is expressed as follows:

$$f(n) = f_1 * f_2 * f_3 \tag{3.37}$$

Loop 3 is expressed as follows:

$$f_1(n) = (n-1) + c * (n-1) \tag{3.38}$$

$$f_1(n) = (c+1)(n-1) \rightarrow O(n) \tag{3.39}$$

$$f_1(n) = (c+1)n - (c+1) \rightarrow O(n) \tag{3.40}$$

Loop 2 is expressed as follows:

$$f_2(n) = (n-1) \rightarrow O(n) \tag{3.41}$$

Loop 3 is expressed as follows:

$$f_3(n) = (n-1) \rightarrow O(n) \tag{3.42}$$

The complexity analysis for the MC algorithm is $O(n^3)$:

$$f(n) = \big((c+1)n - (c+1)\big) * \ (n-1) * \ (n-1) \rightarrow O(n^3) \tag{3.43}$$

WebGL generic algorithm in terms of loops and statements

```
Loop3()
```

```
        Loop2()
                Loop1 (…){
                        < list of sequence statements>>
                }
Loop6 (…)
        Loop5 (…)
                Loop4 (…){
                        <list of sequence statements>>
                                While loop{
                                        <<list of sequence statements>>
                                }}
```

2) The Big-O notation for WebGL algorithm is expressed as follows:

$$f(n) = (f_1 * f_2 * f_3) + (f_4 * f_5 * f_6 * f_7) \qquad (3.44)$$

Where:

Loop 1 is expressed as follows:

$$f_1(n) = (n-1) + c * (n-1) \qquad (3.45)$$

$$f_1(n) = (c+1)(n-1) \rightarrow O(n) \qquad (3.46)$$

$$f_1(n) = (c+1)n - (c+1) \rightarrow O(n) \qquad (3.47)$$

Loop 2 is expressed as follows:

$$f_2(n) = (n-1) \rightarrow O(n) \qquad (3.48)$$

Loop 3 is expressed as follows:

$$f_3(n) = (n-1) \rightarrow O(n) \qquad (3.49)$$

Loop 4 is expressed as follows:

$$f_4(n) = (n-1) + c * (n-1) \tag{3.50}$$

$$f_4(n) = (c+1)(n-1) \rightarrow O(n) \tag{3.51}$$

$$f_4(n) = (c+1)n - (c+1) \rightarrow \; O(n) \tag{3.52}$$

Loop 5 is expressed as follows:

$$f_5(n) = (n-1) \rightarrow O(n) \tag{3.53}$$

Loop 6 is expressed as follows:

$$f_6(n) = (n-1) \rightarrow O(n) \tag{3.54}$$

Loop 7 is expressed as follows:

$$f_7(n) = (n-1) + c * (n-1) \tag{3.55}$$

$$f_7(n) = (c+1)(n-1) \rightarrow O(n) \tag{3.56}$$

$$f_7(n) = (c+1)n - (c+1) \rightarrow \; O(n) \tag{3.57}$$

$$f(n) = \Big(\big((c+1)n-(c+1)\big)\Big) * (n-1) \; * \; (n-1)) + (\big((c+1)n - (c+1)\big) * \; (n-1) \; * (n-1) \; ((c+1)n-(c+1))) \tag{3.58}$$

The complexity analysis for WebGL algorithm is $O(n^4)$:

$$F(n) = O(n^3) + \; O(n^4) \rightarrow O(n^4) \tag{3.59}$$

3) SDF generic algorithm in terms of loops and statements

```
Loop3(){
    Loop1(){
        <SDF list of sequence statements>>}
    Loop2(){
        <list of sequence statements>>}
}
Loop6(){
        Loop4(){
                <list of sequence statements>>}
        Loop5(){
                <list of sequence statements>>}
}
```

The Big-O notation for SDF algorithm is expressed as follows:

$$f(n) = (f_1 + f_2) * f_3 + (f_4 + f_5) * f_6 \tag{3.60}$$

Where:

Loop 1 is expressed as follows:

$$f_1(n) = (c_1 + 1)n - (c_1 + 1) \rightarrow O(n) \tag{3.61}$$

Loop 2 is expressed as follows:

$$f_2(n) = (c_2 + 1)n - (c_2 + 1) \rightarrow O(n) \tag{3.62}$$

Loop 3 is expressed as follows:

$$f_3(n) = (n - 1) \rightarrow O(n) \tag{3.63}$$

Loop 4 is expressed as follows:

$$f_4(n) = (c_4 + 1)n - (c_4 + 1) \rightarrow O(n) \tag{3.64}$$

Loop 5 is expressed as follows:

$$f_5(n) = (c_5 + 1)n - (c_5 + 1) \rightarrow O(n) \tag{3.65}$$

Loop 6 is expressed as follows:

$$f_6(n) = (n - 1) \rightarrow O(n) \tag{3.66}$$

The complexity analysis for SDF algorithm is $O(n^2)$:

$$f(n) = \left[\left(\left((c_1+1)n-(c_1+1)\right)+\left((c_2+1)n-(c_2+1)\right)\right)*(n-1)\right]+ \left[\left(\left((c_4+1)n-(c_4+1)\right)+\left((c_5+1)n-(c_5+1)\right)\right)*(n-1)\right] \quad (3.67)$$

$$f(n) = O(n^2)+O(n^2) \rightarrow O(n^2) \quad (3.68)$$

BIBLIOGRAPHY

Abrahamson, S., Wallace, D., Senin, N., Sferro, P., 2000. Integrated Design In A Servicemarket Place. J. *Computer-Aided Design*, 32, 97-107.

Adzhiev, V., Kazakov, M., Pasko, A., and Savchenko, V., 2000. Hybrid System Architecture for Volume Modelling. *Computers & Graphics*, 24(1), 67–78.

Applied Shaped Limited. *Applied Shapes*. Available from: http://appliedshapes.com

Asghari, M., 2013. An Overview on Boundary Representation Data Structures for 3D Models Representation. *Proceedings of the 3rd International Symposium and Exhibition on Geoinformation*. 14.

Barak, B., Garg, S., KAlai, T. Y., Paneth, O., S, Amit., 2014. Protecting Obfuscation against Algebraic Attacks. *In: Nguyen P.Q., Oswald E. (eds) Advances in Cryptology – EUROCRYPT 2014. EUROCRYPT 2014. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg,* 221-238.

Behr, J., Jung, Y., Keil, J., Drevensek, T., Zoellner, M., Eschler, P., and Fellner, D., 2010. A scalable Architecture for the Html5/X3d Integration Model X3dom. *In proceedings of the 15th International Conference on Web 3D Technology, ser. Web3D '10,* New York, NY, USA: ACM, 185–194.

Berndt, R., Fellner, D. and Havemann. S., 2005. Generative3d models: A Key to More Information within Less Bandwidth at Higher Quality. *In Web3D '05: Proceedings of the Tenth International Conference on 3D Web Technology, ACM Press,* New York, NY, USA, 111–121.

3DTin, *3D Modelling for Everyone*. Available from: https://3dtin.wordpress.com/

Branco, R., and Leitao, A., 2017. Integrated Algorithm Design: A Single-Script Approach for Multiple Design Tasks, *Ecaade 35, Design Tool-Theory,* 1, 729-738.

Bürger, R. and Hauser, H., 2007. Visualisation of Multi-Variate Scientific Data. *In Eurographics, State of the Art Reports (STARs)*, 117–134.

Cahyawiajya, S., and Supriana, I., 2015. Fast and Low Cost Automated Human 3D Modeling and Skeleton Generation. *International Journal of Computer Sciences*, 12(3), 176-181.

Caliskan, A., and Cevik, U., 2017. Three-Dimensional Modeling in Medical Image Processing by Using Fractal Geometry, *Journal of Computer*, 12(5), 479-485.

Canelhas, D., Schaffernicht, E., Stoyanov, T., and Lilienthal, A., 2016. An EigenShapes Approach to Compressed Signed Distance Fields and Their Utility in Robot Mapping, Robotics (Cs. RO), 6(3), 1-13.

Canetti, R., Goldreich, O., and Halevi, S., 2004. The Random Oracle Methodology, Revisited. *Journal of the ACM (JACM)*, 51(4), 557-594.

Cartwright, R., Adzhiev, V., Pasko, A., Goto, Y., and Kunii, T. L., 2005. Web-Based Shape Modelling with HyperFun, *IEEE Computer Graphics and Applications*, 25(2), 60–69.

Chen, D., Ouhyoung, M., Tian, X., and Shen. Y., 2003. On Visual Similarity Based 3D Model Retrieval. Computer Graphics Forum, 223–232.

Chen, J., Li, J., and Li, M., 2016. Progressive Visualisation of Complex 3d Models over the Internet, *Trasaction in GIS, 20(6), 887-902*.

Cho, S., Baek, D., Baek, S., Lee, K., and Bang, H., 2014. 3D Volume Drawing on a Potter's Wheel. *IEEE Computer Graphics and Application Special Issues: Beyond the Screen*, 34(3), 50-58.

Chou, D., Jhou C.-Y., and Chu, S.-C., 2009. Reversible Watermark For 3D Vertices Based On Data Hiding in Mesh Models. *International Journal of Innovative Computing, Information and Control*, 5(7), 1893-1901.

Chourio, X., Luengo, F., and Pirela-Morillo, G., 2011, Creating Multiusere Web3D Applications Embedded in Web Pages. *IJCSI International Journal of Computer Science Issues*, 8(1), 67-73.

Chi Sio, C., Ngan, M., Yi, J., Chen, X., 2010. Volume Rendering with Marching Cube Algorithm. University of Southern California.

Cirne, M and Pedrini, H., 2013. Marching cubes Technique for Volumetric Visualization Accelerated with Graphics Processing Units, *Journal of the Brazilian Computer Society*, 223–233.

Collberg, C., and Thomborson, C., 2000. Watermarking, Tamper- Proofing, And Obfuscation - Tools for Software Protection. *Technical Report TR00-03*, The Department of Computer Science, University of Arizona.

Congote, J., Kabongo, L., Moreno, A., Segura, A., Beristain A, Posada J, Ruiz, O., 2012. Volume Ray Casting in WebGL. *Computer Graphics*. *InTech*. Rijeka, 157–178 .

Congote, J., Segura, A., Kabongo, L., Moreno, A., Posada, J., and Ruiz, O., 2011. Interactive Visualization of Volumetric Data with WebGL in Real-Time. *In Proceedings of the 16th International Conference on 3D Web Technology, ser. Web3D '11. ACM*, 137–146.

Corney, J., Rea, H., Clark, D., 2002. Pritchard, J., Breaks, M., and Macleod. R., Coarse Filters For Shape Matching. *IEEE Computer Graphics & Applications*, 22(3), 65–74.

Dietrich, A., Gobbetti, E., and Yoon, S., 2007. Massive-Model Rendering Techniques. *IEEE computer graphics and applications*, 27(6), 20-34.

Dong, X., Chen, Z., Siadati, H., Tople, S., Saxena, P., Liang, Z., 2013. Protecting Sensitive Web Content from Client-Side Vulnerabilities with CRYPTONs. *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 1311-1324.

Erl, T., 2005. *Service-Oriented Architecture: Concepts, Technology, and Design*. 1st edition, Upper Saddle River, NJ, USA: Prentice Hall PTR.

Evans, A., Romeo, M., Bahrehmand, A., Agenjo, J., and Blat, Josep, 2014. 3D Graphics on the Web: a Survey. *Interactive Technology Group*. University Pompeu Fabra, Barcelona, Spain, 41, 43-61.

Fayolle, P.-A., Schmitt, B., Goto, Y., and Pasko, A., 2005. Web-Based Constructive Shape Modelling Using Real Distance Functions. *IEICE - Trans. Inf. Syst*, 88(5), 828–835.

Fayolle, P., and Pasko., A, 2016. Surface Discretization of Multi-material Heterogeneous Volume Objects. *ICME Workshop,*

Fisseler, D., Muller, G., and Weichert, F., 2017. Web-Based Scientific Exploration and Analysis of 3D Scanned Cuneiform Datasets for Collaborative Research. *Informatics.* 4(4), 1-16.

Fryazinov, O., Pasko, A., and Adzhiev, V., 2008. An Exact Representation Of Polygonal Objects By C1-Continuous Scalar Fields Based On Binary Space Partitioning. *IEEE Computer Society, the National Centre for Computer Animation, Bournemouth University, UK*. 132-139.

Calbriath, G., MacMurchy, P., and Wyvill, B., 2004. BlobTree Trees. *Proceedings of Computer International Conference, CGI, Crete, Greece.* 78-85.

Gerth, B. Berndt, R., Havemann, S., and Fellner. D., 2005. 3D Modelling For Non-Expert Users with the Castle Construction Kit V0.5. *The 6th International Symposium on Virtual Reality (VAST 2005), Archaeology and Intelligent Cultural Heritage, Pisa, Italy*. 1-9.

Gisi, M. A., and Sacchi, C., 1994. Co-CAD, a Multi-User Collaborative Mechanical Cad System, *Presence Teleoperators & Virtual Environments.* 3(4), 341-350.

Trimble Inc. *Sketchup, Where Great Ideas Get to work, Design it. Make it. Enjoy the process*. Available from: http://sketchup.google.com.

Grasberger, H., Shirazian, P., Wyvill, B., and Greenberg, S., 2013. A Data Efficient Collaborative Modelling Method Using Websockets and the Blob tree for Over-The Air Networks. *In proceedings of the 18th International Conference on 3D Web Technology, ser. Web3D '13*. New York, NY, USA: ACM. 2–37.

Grillie, E., Menna, F. and Remondino, F., 2017. A Review of Point Clouds Segmentation and Classification Algorithms. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences XLII-2/W3,* 339-334.

Gupta, V., Kasana, K.S., Tandon, P., 2010. Computer Aided Design Modeling for Heterogeneous Objects. *International Journal of Computer Science, IJCSI*. 7(2), 31-38.

Havemann, S., 2005. *Generative Mesh Modelling.* PhD thesis, Braunschweig Technical University, Germany.

Hoppe, H., 1966. Progressive Meshes. *In proceedings to the Twenty-Third Annual Conference on Computer Graphics and Interactive Techniques,* Louisiana. 99-108.

Jung, Y., Behr, J,. Drevensek, T., Wagner, S., 2012. Declarative 3D Approaches for Distributed Web-Based Scientific Visualization Services, *Proceedings of the 1st International Workshop on Declarative 3D for the Web Architecture*. 869, 7.

Kang, S. and Lee, J., 2017. Developing a Title-Based Rendering Method to Improve Rendering Speed of 3D Geospatial with HTML and WebGL, *Hindawi, Journal 0f Sensors*. 2017, 11.

Klein, M., 1991. Supporting Conflict Resolution in Cooperative Design, *IEEE Transactions on Systems, J. Man and Cybernetics, Special issue on Distributed Artificial Intelligence*. 21(6), 1379-1390.

Kobbelt, L., Bischoff, S., Botschm M., Kahler, K., Rossl, C., Shneider, R., and Vorsatz, J., 2000. Geometric Modeling Based on Polygonal Meshes. *Eurographic*. 1-48.

Koller, D., Turitzin, M., Levoy, M., Tarini, M., Croccia, G., Cignoni, P., and Scopigno, R., 2004. Protected Interactive 3D Graphics Via Remote Rendering. *ACM Trans. Graph,* 23(3), 695–703.

Khronos Group, Khronos royalty-free open standards for 3D graphics, Virtual and Augmented Reality, Parallel Computing, Neural Networks, and Vision Processing. USA. Available from: https://www.khronos.org/.

Krottmaier, H., Kurth, F., Steenweg, T., H.-J, A., and Fellner. D., 2007. PROBADO - A Generic Repository Integration Framework. *In Proceedings of the 11th European Conference on Digital Libraries, LNCS 4675 (3),* 518–521.

Kumar, V., Rajagopalan, S., Cutkosky, M., and Dutta, D., 1998. Representation and Processing of Heterogeneous Objects for Solid Freeform Fabrication. *IFIF WG5.2 Geometric modelling workshop.* 7-9.

Laine, S., 2013. A Topological Approach to Voxelization, *EGSR '13 Proceedings of the Eurographics Symposium on Rendering*. 77-86

Lavoue, G., Larabi, M., and Vasa, L., 2016. On The Efficiency Of Image Metrics For Evaluating The Visual Quality Of 3D Models, *IEEE Transaction On Visualization And Computer Graphics*. 22(8), 1987-1999

Lee, H., Lavoue, G., and Dupont, F, 2012. Rate-Distortion Optimization For Progressive Compression Of 3D Mesh With Colour Attributes. *The Visual Computer,* 137-153.

Leitao, A. and Lopes, J., 2011. Portable Generative Design for CAD Application. Integration through Computation. *Proceedings of the 31st Annual Conference of the Association for Computer-Aided Design in Architecture (ACADIA)*. 196-203.

Li, S., Xiao, Y, and Wang, X., 2013. Three-Dimensional Information Security Combined Fringe Projection With Double Random Phase Encoding Optics Communication. *ScienceDirect Journal, Optics Communications.* 296, 35-40.

Limper, M., Wanger, S., Stein, C., Jung, Y., Strok, A., IGD, F, Darmstadt, T., 2013. Fast delivery of 3D Web content: A case study. *Web3D '13 Proceedings of the 18th International Conference on 3D Web Technology*, 11-17.

Lindborg, T., Gifford, P., and Fryazinov, O., 2017. Interactive Parameterized Heterogeneous 3D Modelling With Signed Distance Fields. *SIGGRAPH '17 ACM SIGGRAPH 2017*. Posters Article. 6, 1.

Lin, Y., and Wu, J., 2012. Unseen Visible Watermarking For Colour Plus Depth Map Images. *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP),* 1801-1804.

Liu, Q. and Sourin, A., 2006. Function-defined Shape Metamorphoses in Visual Cyber worlds. *Visual Comput*., 22(12), 977–990.

Lui, Y., Prabhakaran, B., Guo, X., 2012. Spectral Watermarking for Parameterized Surfaces. *IEEE Transaction on Information Forensics and Security*. 7(5), 1459-1471.

Lorensen, W., and Cline, H., 1987. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *Proceeding SIGGRAPH '87 Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, NY USA. 21(4), 163-169.

Maglo, A., Courbet, C., Alliez, P., Hudelot, C., 2012. Progressive Compression of Manifold Polygon Meshes. *Computers and Graphics, Elsevier, Shape Modeling International (SMI) Conference 2013,* 36 (5), 49-359.

Mars, J., and Hundt, R., 2009. Scenario Based Optimization: A Framework for Statically Enabling Online Optimizations, *Code Generation and Optimization. International Symposium, Seattle, WA, USA*. 169-170.

McHenry, K. and Bajcsy, P., 2008. *An Overview of 3D Data Content, File Formats and Viewers*. *National Center for Supercomputing Application*, 1205 W Clark, Urbana.

Oleynikova , H., Millane, A., Taylor, Z., Galceran, E., Nieto , J., and Siegwart , R., 2016. Signed Distance Fields: A Natural Representation for Both Mapping and Planning. *Geometry and Beyond Representations, Physics, and Scene Understanding for Robotics, Ann Arbor, MI, USA. 6.*

Parmar, B., and Bhatt, T., 2016. Volume Visualization Using Marching Cubes Algorithms: Survey & Analysis, *IJIRT*, 2(11), 21-25.

Parulek, J., Novotny, P., and Sramek, M., 2006. Xisla Development Tool for Construction of Implicit Surfaces. *In SCCG 06: Proceedings Of The 22nd Spring Conference On Computer Graphics, Comenius University, Bratislava*, 128–135.

Pasko A., and Adzhiev V., 2004. Function-Based Shape Modelling: Mathematical Framework and Specialized Language. *In: Winkler F. (eds) Automated Deduction in Geometry. ADG 2002. Lecture Notes in Computer Science*, 2930,132-160.

Pasko, A., Adzhiev V., Sourin, A. and Savchenko, V., 1995. Function Representation in Geometric Modelling: Concepts, Implementation and Applications. *The Visual Computer*, 11, 429–446.

Pasko, A., Adzhiev, V., 2001. Schmitt, B., and Schlick, C., Constructive Hypervolume Modeling. *Graphical Models*, ScienceDirect Journal. 63(6), 413-442.

Pasko, G., Pasko A., and Kunii., T., 2005. Bounded Blending for Function-based Shape Modelling, IEEE CG&A magazine, 25, 36-45.

Pasko, A., Vilbrandt, T., Fryazinov, O., Adzhiev V., 2010. Procedural Function-Based Spatial Microstructures. *Shape Modelling International Conference*, *Aix in Provence, France*. 47-56.

Pantaleoni, J., 2011**.** VoxelPipe: A Programmable Pipeline for 3D Voxelization, *HPG '11 Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics, Vancouver, British Columbia, Canada.* 99-106

Pan, J., Zheng, J., and Zhao, G., 2013. Blind Watermarking of NURBS Curves and Surfaces. *Computer Aided Design*, 45(2), 144-153.

Peng, J., Kim, C.,-S., and Jay Kuo, C. C., 2005. Technologies for 3D Mesh Compression: A survey. *J. Comum. Image Represent*, 688-733.

Popvski, F., Nedelkovski, I., and Mijakovska, S., 2014. Generating 3D Model In Virtual Reality And Analyzing Its Performance. *International Journal of Computer Science & Information Technology (IJCSIT)*, 6, 6.

Qin, S., Wright, D., 2004. Incremental Simulation Modelling for Internet Collaborative Design, *Proc. of ImechE, Part B, Journal of Engineering Manufacture*, 218, 1009-1015.

Ramani, K., Agrawat, A., and Babu, M., 2003. CADDAC: Multi-Client Collaborative Shape Design System with Server-based Geometry Kernel. *In Journal of Computing and information Science in Engineering*. 3(2), 170-173.

Rezayat, M., 2000. The enterprise-Web Portal for Life-Cycle Support. *J. Computer-Aided Design*, 32, 85-96.

Requicha, A., 1980. Representations for Rigid Solids: Theory, Methods, and Systems. *ACM Computing Surveys*, 12(4), 437-464.

Rivest, R., Adleman, Len., and Dertouzos, L. C., 1978. On Data Banks and Privacy Homomorphisms. *In Foundations Of Secure Computation, Work- shop, Georgia Inst. Tech., Atlanta, Ga., 197*, Academic, New York. 169-179.

Rodrigues, M., Kormann, M., and Davison, L., 2011. A Case Study Of 3D Technologies In Higher Education: Scanning The Metalwork Collection of Museums Sheffield and its Implications to Teaching And Learning, International Conference On Information Technology Based Higher. IEEE, 1-6.

Rodrigues, R. And Robinson, A., 2009, *Developing Interactive 3D Models for E-Learning applications*. Book chapter in M.T. de Mello, Hipermidias Interfaces Digitais em EAD, Editora Laborciencia Ltd, Sao Paulo, C.Z. Carvalho Neto, and F.J. Spanhol (Eds), 155-175.

Sanchez, M., Fryazinov, O., Pasko, A., 2012. Efficient evaluation of continuous signed distance to a polygonal mesh. *12 Proceedings of the 28th Spring Conference on Computer Graphics*, NY, USA. 101-108.

Sanchez, M., Fryazinov, O., Fayolle, P., and Pasko, A., 2015. Convolution Filtering Of Continuous Signed Distance Fields For Polygonal Meshes. *Computer Graphics Forum*, 00 (0), 1-12.

Santos, P., Strok, A., 2004. SmartSketches: *A Multimodal Approach to Improve Usability in the Early States of Product Design*, D 18b SketchAR Technical Report, Information Society Technologies (ITS).

Satoshi, H., 2000. Zero-knowledge and code obfuscation. In T. Okamoto, Editor, Advances in Cryptology - ASIACRYPT 2000, Lecture Notes in Computer Science, Kyoto, Japan, *International Association for Cryptologic Research*, Springer-Verlag, Berlin Germany, 443-457

Schmitt, B., Pasko, a., Adzhiev, V., and Schlick, V., 2001. Constructive Texturing Based on Hypervolume Modeling. *The Journal of Visualization and Computer Animation*, 12(5), 297-310

Schimitt, B., Pasko, a., and Schlick, C., 2004. Constructive Sculpting of Heterogeneous Volumetric Objects Using Trivariate B-splines. *The Visual Computer, International Journal of Computer Graphics*, 20 (2–3), 130–148.

Schmitt, B., Pasko, A., Adzhiev, V., Paskp, G., and Schlick, C., 2008. Modelling Function-Based Mixed-Dimensional Objects with Attributes, *Part of the Lecture Notes in Computer Science book series (LNCS)*, 4889, 90-117.

Schwartz, C., Ruiters, R., Weinmann, M., and Klein, R., 2013. WebGL-based Streaming and Presentation of Objects with Bidirectional Texture Functions, *ACM Journal on Computing and Cultural Heritage*, 6(3),11.

Shapesmith, *Parametric Open Source 3D Modelling*. Available from: http://shapesmith.net.

Shapeways, *Write Code, Generate Products*. Available from: http://shapejs.shapeways.com

Shyamsundar, N., Gadh., R., 2001. Internet-Based Collaborative Product Design with Assembly Features and Virtual Design Spaces, *Computer-Aided Design*, 33, 637–651.

Slavcheva, M., Kehl, W., Navab, N., and Ilic, S., 2006. SDF-2-SDF: Highly Accurate 3D Object *Reconstructions, Computer Vision – ECCV 2016, 680-696*.

Soltani, A., Huang, H., Wu, Jiajun, Kulkarni, T. and Tenenbaum, J., 2017. Synthesizing 3D Shapes Via Modeling Multi-View Depth Maps and Silhouettes With Deep Generative Networks, *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, USA. 2511-2519.

Sons, K., Klein, F., Rubinstein, D., Byelozyorov, S., Slusallek, P., 2010. XML3D- Interactive 3D Graphics for the Web. *Proceedings of the 15th International Conference on Web 3D Technology,* 175-184.

Sutherland, I., 2003. Sketchpad: A Man-Machine Graphical Communication System, University of Cambridge, UK.

Autodesk Tinkercad. *From mind to design in minutes*. Available from: http://tinkercad.com.

Tsai, F., Chang, H. and Lin, E.-W., 2017. Combining 3D Volume and Mesh Models for Representing Complicated Heritage Buildings. *The International Archives of the Photogrammetry*, *Remote Sensing and Spatial Information Sciences*. XLII-2/w5, 673-677.

Uformit, *Online Marketplace and Platform for Personalized Design.* Available from: www.uformit.com.

Vanhoey, K., Sauvage, B., Kraemer, P., and Lavoue, G., 2017. Visual Quality Assessment of 3D Models: On the Influence of Light-Material Interaction. *ACM Transaction on Applied Perception.* 15(1.), 1-18.

Vilbrandt, C., Pasko, G., Pasko, A., Fayolle, P., Vilbrandt, T., Goodwin, J., and Goodwin, J., 2004. Cultural Heritage Preservation Using Constructive Shape Modelling. 23 (1), 25–41.

Vilbrandt, T., Fryazinov, O., Stamm, C. and Pasko, A., 2010. A Web-Oriented Function-Based Volume Modelling Framework. *Computer Graphics & Geometry*. 12, 41–51.

Vranic. D., 2003. An Improvement Of Rotation Invariant 3D Shape Descriptor Based On Functions On Concentric Spheres. *In IEEE International Conference On Image Processing (ICIP 2003).* 3, 757–760.

Wang, R., Yu, B., Marco, J., Hu, T., Gutierrez, D., Bao, H., 2016. Real-time Rendering on a Power Budget, *ACM Transaction on Graphics*. Anaheim, CA. 35(4)

Wang, W., Liu, Y., Wang, C., Lui, L., and Lui X., 2017. Support-Free Hallowing, *IEEE Transaction on Visualization and Computer Graphics*. 24(10), 2787-2798.

While, L., Himgston, P., Barone, L., and Hunand, S., 2006. A Faster Algorithm for Calculating Hyper volume. *IEEE Transaction on Evolutionary Computation*, 10 (1), 29-38

Wu, D., Rosen, D.W., Wang, L., and Schaefer, D., 2014. Cloud-Based Manufacturing: Old Wine in New Bottles?, Variety Management in Manufacturing. *Proceedings of the 47th CIRP Conference on Manufacturing Systems*. 17, 94-99.

Wu, D., Terpenny, J., and Schaefer, D., 2016. Digital Design and Manufacturing on the Cloud: A Review of Software and Services, Artificial Intelligence for Engineering Design, Analysis and Manufacturing, Cambridge University. 31(1), 104-118

Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., and Xiao, J., 2015. 3D ShapeNets: A Deep Representation for Volumetric Shapes. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA*. 9.

Xiao, D. And Shih, F.Y, 2010. A Reversible Image Authentication Scheme Based On Chaotic Fragile Watermark. *International Journal Of Innovative Commuting, Information And Control*, 6(20), 4731-4742.

Yu, H., Zhang, J., Wang, L., and Barksdale, J., 2003. A secure Web Application: 3D Visualization and Collaboration, *Proceedings of the 2nd IASTED International Conference, Communication, Internet, and Information Technology*, Scottsdale. AZ, USA. 13-18

Zeki, A., Abu Bakar, A., 2013. 3D digital watermarking: Issues and Challenges. *Proceeding of the International conference on Artificial Intelligence in Computer Science and ICT*, Langkawi, Malaysia. 334-342.

Zollhofer, M., Dai, A., Innmann, M., Wu, C., Stamminger, M. , Theobalt, C. and Nießner, M., 2015. Shading-Based Refinement on Volumetric Signed Distance Functions. *ACM Transactions on Graphics (TOG)*,34(4), 96:1-96:14.