3D Web-Based Shape Modelling: Building up an Adaptive Architecture

Ali Abdallah, Oleg Fryazinov, Valery Adzhiev and Alexander Pasko
The National Centre for Computer Animation,
Bournemouth University,
Poole, United Kingdom,
Email: ccengineer@gmail.com, {ofryazinov, vadzhiev, apasko}@bournemouth.ac.uk

***Abstract*—3D web-based shape modelling and rendering is becoming an increasingly important research area. Many applications have emerged, both established ones, such as collaborative design and new ones, such as heterogeneous objects modelling along with their subsequent fabrication using 3D printing. In this paper, we explore a crucial issue of the technology, which is building a proper adaptive architecture for an interactive client-server based system with a particular emphasis on rendering aspects. We identify the most probable scenarios of executing modelling and rendering in terms of server-client communications and associated decision making, and then describe a number of case-studies, which allowed us to experiment with different rendering techniques in the context of various task distribution and communications mode of the adaptive interactive client-server based system. Finally, we present and analyse the results and suggest a number of practical recommendations. The main results of the paper are concerned with consideration of rendering.**

Keywords-Adaptive architecture, 3D shape modelling, WebGL, Collaborative shape modelling, Function Representation

I. INTRODUCTION

3D web-based shape modelling and rendering is an important emerging area of recent research, especially in the context of collaborative computer aided design, distributed computer games, scientific visualization applications based on data from a number of sources, and other applications, such as increasingly popular remote fabrication and 3D printing on demand. A number of efficient software tools have been developed, which allow for a wider scope of web applications. The web browsers make use of languages, such as VRML [1], WebGL [2], etc., to describe complex 3D scenes.

Despite rapid development of hardware including specialised Graphics processing units (GPUs) and widening Internet bandwidth, truly interactive applications allowing for near real-time rendering without loss of a visual quality are yet to become a reality. Tools outputting models in the standard format of polygonal meshes have many drawbacks and limitations (especially in a collaborative mode), and inefficiency of communications through the network and rendering, 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 a proper client-server architecture for 3D web-based systems, the development of the collaborative tools for modelling and rendering requires a flexible and efficient handling of hardware and software resources of each client to achieve an efficient workflow. In this paper, we explore in depth the problems of building such an adaptive environment with a particular emphasis on rendering aspects, thus defining the most efficient way the content is transferred between the server and the clients depending on the available resources.

From the geometric point of view, the most common format to represent and store 3-dimensional objects is polygonal meshes that embody the Boundary Representation (BRep). This format is widely supported by the modern 3D APIs implemented in web browsers and quite a few methods to create and process content in this format exist. However, the polygonal meshes are by definition an approximation of the mathematically precise model and have well-known issues concerned with the loss of the shape precision and visual property definition, limited complexity, large memory consumption, problems with transferring through networks, etc [3]. These result in problems with interactivity which are so important in collaborative web-based projects. An inability to access the construction history is also an issue [4].

These problems have become increasingly critical in the context of modelling heterogeneous objects where their internal structure is to be represented and rendered. One of the solutions to this fundamental problem is using the Function Representation (FRep) instead or together with BRep, where a 3D object is represented by a continuous function of point coordinates (implicit surfaces are a particular case of FRep). It allows for dealing with objects as volumes with an internal structure, keeps the constructive tree of their modelling process and is compact by its very nature. However, the main problem with this representation is a need to convert an FRep model to existing supported representations during a rendering step which can be an expensive procedure with some difficult technical issues. In this paper, we focus on the rendering aspects of collaborative modelling systems with FRep at its core in the context of building the most efficient client-server network architecture for an adaptive environment.

In this paper, we explore different ways of building the interactive architecture for the modelling system based on FRep with different types of adaptations to the client needs with a particular consideration of rendering. The paper structure is as follows. After a survey of related works, we describe a 3D shape modelling architecture of Web-bases system considering

 ISBN: 978-1-61208-325-4

in detail first a pure client-server one and then an adaptive one. Having identified four the most probable scenarios of executing modelling and rendering in terms of server-client communications and decision making, we then consider a number of case-studies, where different rendering techniques (based on Marching cubes, Hybrid WebGL and server based rendering by using C++) are used depending on the different client abilities. Finally, we present the results of the experiments with both simple and more complex models as well as some practical recommendations reflecting the advantages and drawbacks of the tested techniques.

II. Related Work

The related works include Web-based scientific visualization, collaborative shape modelling, Computer-Aided Design and Manifacturing (CAD/CAM), and Web 3D concerned with delivery and interaction with 3D geometric models on the Web.

The overview of methods allowing to handle visualization-specific representation that consists of registered and merged points, surface and volume data as well as the corresponding meta information in order to provide important features for scientific visualization was presented in [5]. The survey shows that the information is usually re-sampled onto a structured regular grid after data acquisition and before filtering the data accordingly. However, transmission of vast data arrays prevents interactive modifications and collaborative work.

Most of the tools allowing collaborative solid modelling adopt Boundary Representation as the data exchange format defining an object by a set of surface patches stitched together. Thus, Kao proposed a collaborative CAD/CAM system CO-CADCAM that includes surface modelling, tool path simulation and post-processing, and CAD geometry co-editing [6]. Another example is a NURBS modelling system that supports a multi-user environment for collaborative conceptual shape design [7]. Sharing and editing of a solid model over the web can be done by collaborative solid modelling, but requires a whole modeller installed at each client [8]. Some collaborative modelling systems restrict the editing to the limited set of operations, such as modification of specific features of the model [9].

In web-based modelling, the main question is interaction between server and the client, i.e., the information that server sends to the client to render the model. In case of volume data the amount of information transmitted between a server and a client can be significantly large and the client is often requires to install an additional software tool. Thus, in X3D format [10], 3D objects supporting point, surface and volume primitives are described, but additional plugins for the browser needs to be installed. Web-based direct volume rendering with ray-casting was discussed by Congote et al. in [11] for the purposes of medical imaging and radar meteorology. Another way to send the volume information is Bidirectional Texture function that allows the progressive transmission and interactive rendering of digitized artifacts consisting of 3D geometry and reflectance information [12]. The similar approach is used in X3DOM where a lightweight geometry is compressed and transmitted with so-called image geometries [13].

Service-Oriented Architecture (SOA) [14] composes several low-level services to more complex services with a higher level of abstraction. SOA utilizes REST, JSON and XML-based web-service protocols and helps in supporting the collaboration between different applications running on different platforms as discussed in [14]. Koller et al. transfer images to the client and include a number of active defense methods to guard against 3D reconstruction attack by providing an interesting proposal to the protection system with a remote rendering service [15].

FRep based experimental systems for interactive and collaborative modelling on the Web include HyperFun Java applet [16], EmpiricalHyperFun [17], FVRML/FX3D [18], XISL [19], Hyperfox plug-in to Firefox [4], and a BlobTree implementation with websockets [20]. Most of these works had one solution implemented for rendering, mainly the isosurface polygonizaion on the client side. Today without the need for any plug-in installation process, WebGL provides access to the native GPU layer for rendering in a browser on the client side, which provides a basis for adaptive rendering architectures.

III. Adaptive 3D Shape Modelling Architecture

As we discussed above, our main motivation in using FRep models is the reduced complexity of the models allowing to avoid well-known problem of handling large 3D data files and vast amount of computational resources. However, the major drawback of these models is difficulties of handling them inside a web-browser because of the lack of the native support of non-polygonal objects in the current standards for rendering 3D scenes. Modern browsers allow only to load 3D models defined in the form of polygonal mesh and interactively manipulate these scenes (translate, rotate, scale) inside web browser with an input device, such as mouse or keyboard. Usually, the conversion between an FRep model and a polygonal model is required with the process called polygonization. In this section we discuss different ways of implementing web-based shape modelling with FRep objects.

A. Pure Client-Server Architecture

The client/server architecture can be considered as a network environment that exchanges information between a server machine and a client machine where server is a large-capacity computer, with a large amount of data stored on it and available for sharing with different clients. The clients are smaller computers that are used to perform local computer tasks. The client/server architecture reduces multiple copying of a single file and allows an organization to have one centralized point for every computer to access the same application.

The system we discuss here is platform-independent from the client point of view. As the client can have very small abilities to handle geometric data, we consider the server to be responsible for performing most of the computationally intensive tasks. By computationally expensive tasks, we consider primarily the point queries, i.e., calculation of the function value for the given point in space. The point query is mostly used for rendering purposes. It can be shown that all the other operations in a shape modelling system based on FRep, such as modification of the function, adding primitives and operations to the defining function and others, are very cheap and can be done on virtually each client. Therefore, in the pure client/server architecture the user still can create objects and

 ISBN: 978-1-61208-325-4

define operations, import and export objects in the appropriate file format, or import geometry in common file formats.

The proposed three-tier architecture encompasses a client-side, a server-side and a database. The client side has a copy of the master 3D model, rendered in the web browser with one of the techniques discussed below. The web browser GUI contains a number of tools for creating primitives and performing operations on the model with using user's hardware resources (GPU and local memory). The server side contains the master copy of the model with all the references to the external files stored in the database and the kernel modelling system. The server is responsible, as mentioned above, for performing most of the computationally expensive tasks. The task of transferring data through a network medium, such as the Internet or an intranet, is performed by a communication layer between the clients and the server.

Depending on the abilities of the client, the data can be transferred from the server to the client as either polygonal mesh objects, WebGL texture objects or as pure image files. In the first two cases the picture is generated with WebGL, in the third one - by native browser's image handling. The work on the server and on the client is connected by a code written in JavaScript. In the case of WebGL rendering, the client side utilizes X3DOM, which uses JavaScript with WebGL.

We would like to stress that the resulting system should be scalable meaning that the client can be either desktop or mobile with different hardware abilities. For example, mobile clients can often process only image data and therefore the server should be able to stream images instead of 3D data.

B. Adaptive Architecture

As we mentioned before, 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: type of the client machine (desktop or mobile), CPU and GPU availability and power, amount of CPU and GPU memory and the supported software (such as WebGL support in the browser). Therefore, an adaptive architecture is the one that takes all the parameters into account to process the objects from the server to choose the best possible way to deliver model rendering to the client.

From the rendering point of view, the FRep model can be rendered in two different ways:

- Polygonization, i.e., conversion of an FRep object surface into a polygonal mesh and then rendering of the resulting mesh by traditional tools;
- Direct rendering, usually in the form of ray-casting.

In the client-server chain, the client machine potentially can be more computationally powerful than the server and in this case the adaptive architecture should take that in mind and transfer not the result of the rendering to the client but rather the model itself, such that the client performs the rest of the rendering tasks. If rendering takes part on the server, different ways to send the result to the client can be used. These include delivery of the result as

- images obtained after the direct rendering;

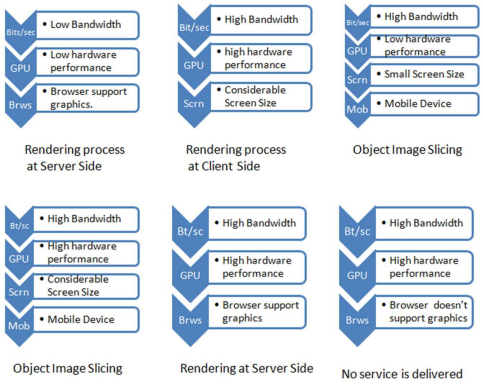

Figure 1. Different Types of Scenarios and Decision Making.

- objects delivered as image slices (voxel array);
- objects delivered as discrete data structures, for example point clouds or polygonal meshes;

Therefore, the adaptive shape modelling architecture or environment is the one that is able to adapt or react and interact with clients according to the abilities of the client, and uses different scenarios depending on the needs and the abilities of the client by choose the most efficient way.

To make a decision regarding the appropriate scenario, the adaptive architecture gets the information about the client machine while communicating in the background to retrieve the data about the characteristics of the hardware and the browser and to calculate the bandwidth download rate. The retrieved information helps the environment to analyze each client or user and decide which scenario it should use for this particular client. It can be seen that lots of various combinations of the types of rendering, machine for rendering and others can lead to large number of different combinations (Fig. 1). However, in practice we identified four most common scenarios. They are the following:

1) The request from the server identifies a client with low bandwidth and low computational resources, the server renders the model and sends the result as a low-resolution static image.
2) The client has sufficient bandwidth, but the browser is not capable to render 3D object, the server renders the model from different points of view and sends the result as set of mid-resolution static images to allow transformation of the object on the client.
3) The client has sufficient bandwidth and is capable to render 3D objects, however, it has low computational resources comparing with the server. The server performs polygonization of the model and sends the resulting polygonal mesh to the client. The client renders the polygonal model using its own resources.
4) The server identifies the high performance machine on the client side, it sends the complete model to the client, such that the client can render the model using local resources. In this scenario, the server performs only the model data transfer, because no rendering takes place on its hardware resources.

In the following section, we present implementation and

 ISBN: 978-1-61208-325-4

experiments with some of the identified scenarios of FRep models rendering on adaptive architectures.

IV. Implementation and Results

Leaving questions of collaborative editing of FRep objects within a shape modelling system beyond the scope of this paper, we want to focus more on the rendering aspects of such a system. The core of this system is the adaptive architecture we described above. For our experiments, we implemented a prototype of this adaptive environment that allows to work with pre-defined FRep models of different complexity. Below we discuss some aspects of the prototype and the experimental results.

A. Adaptive architecture implementation

Being a web-oriented system, most of the tasks for adaptation take part 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, screen resolution info) to the server as XML messages. Of course, there is no need to re-send information, such, as OS info and IP address, that is not going to change. 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 client is connected to the server to ensure that the selected scenario needs to be continued.

As it can be seen from the scenarios we discussed in the previous section, most critical information that server gets from the client is the connection speed and the browser ability to use WebGL. The connection speed detection begins when the client starts downloading a certain file with predefined size in kilobytes from the server in the background; a timer starts as soon as the downloading process starts, the duration is obtained as soon as the download process stops, the transfer bit rate is then calculated by dividing the size of the file by the duration. The duration is calculated by subtracting the end time from the start tame and dividing the result by one thousand (to convert from milliseconds to seconds), the number of bits loaded is calculated by multiplying the downloaded file size by eight, the speed of download is obtained by dividing the number of bits loaded by the duration obtained. The diagram (Fig. 2) describes the machine info detection process starting by determining the agent of the user and check whether it is a portable device or not. Determining other mobile devices can be performed by the operating system detection process. Detection of WebGL abilities can be done either from the canvas initialisation, meaning that successful initialisation indicates that the machine supports WebGL, or by analysing web browser information. For example, information that the browser is Internet Explorer of version 10 and early means that WebGL is not supported.

B. Rendering techniques comparison

For our tests, we implemented three rendering techniques corresponding to three different scenarios. All the tests were done on the same machine running Chrome web browser

Figure 2. 3D simple objects created using marching cubes, with low quality (resolution) and sharp edges.

Figure 3. 3D simple objects rendered using WebGL library (Three.js) for rendering, objects are with smooth edges and higher resolution.

Figure 4. 3D simple objects rendered at the server side (using C compiler) and sent to the client as images.

with high-speed network connection to the server. The first technique is rendering on a server (Fig. 5) by using C++ code and sending still images to a client. The second one is polygonization on the server and rendering the resulting mesh on a client by using WebGL (three.js library) (Fig. 4). The third one is performing polygonization on a client by applying marching cubes technique implemented as JavaScript code (Fig. 3). Note that in the second scenario the polygonization is done with C++ code on a server while in the third scenario the polygon mesh is being created on a client side implemented in JavaScript.

Models with different complexity were used in our tests (Figs. 5,7,8). We show the timings of the rendering by using different methods in the Table 1. To achieve similar performance rate, low resolution was used for rendering on a client side that resulted in visible edges and lower quality of the result (Fig. 3). The resolution can be increased, but in this case more computational resources from the client are required.

It can be seen that for simple models different rendering techniques showed no major difference in timings. However, as the complexity of the object increased, the difference in timings becomes more clear. 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 an image to the client. At the same time it can be seen that the ability to work with model in interactive way is limited, as we have to re-render the model for each new camera position.

The chart in Fig. 9 shows the performance for the three rendering techniques with respect to shape models of different complexity. The green curve represents the 3D objects rendered using C++ at the server side. It shows that complex

 ISBN: 978-1-61208-325-4

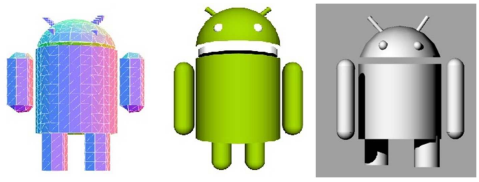

Figure 5. Android Robots redered using different techniques, the robot to the left was created using marching cubes, the robot in the middle was rendered using WebGL library (Three.js) and the robot to the right was rendered at the server side using rhinoceros and sent as image to the client.

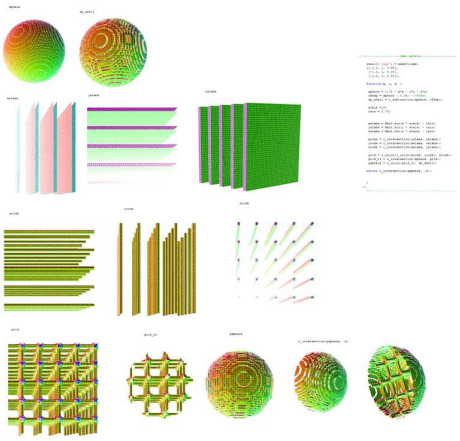

Figure 6. Image slicing showing the different rendering phases the complex Hemi-sphere model took after applying different functions using marching cubes.

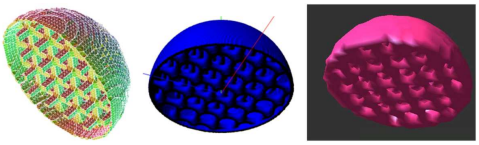

Figure 7. Hemi-Sphere models (Complex Models) were rendered using different techniques, the complex model to the left was created using marching cubes, the one in the middle was rendered using WebGL library (Three.js) and the one to the right was rendered at the server side using C Language (C++ in our case) and sent as image to the client.

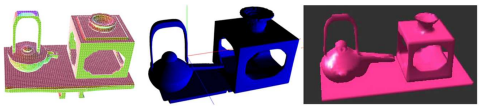

Figure 8. Sake pot models (Complex Models) were created using different techniques, the complex model to the left was created using marching cubes, the one in the middle was rendered using WebGL library (Three.js)and the one to the right was rendered at the server side using C++ and sent as image to the client.

objects (Hemi-sphere and Sake Pot models) were rendered in a fraction of a second. The blue curve represents objects rendered using polygonization with the Marching Cubes. The rendering time stayed almost constant when rendering simple objects, and rose when rendering complex objects. The rendering time for the hemi-sphere and the sake pot objects was less than half a second. The red curve representing rendering with hybrid WebGL started to rise up sharply when rendering complex objects, this indicates the amount of time and the

TABLE I. COMPARING DIFFERENT RENDERING TECHNIQUES(MARCHING CUBES, WEBGL USING THREE.JS, AND SERVER SIDE RENDERING), THE TABLE BELOW SHOWS THE TIME NEEDED IN MILLISECONDS FOR EACH TECHNIQUE IN ORDER TO CREATE A SIMPLE 3D OBJECT

3D Object	Marching Cubes	WebGLThree.js	Server Rendering
Sphere	0.006	0.003	0.0013
Semi-Sphere	0.005	0.03	0.0013
Cylinder	0.003	0.02	0.0012
Closed-Cylinder	0.007	0.025	0.0013
Android Robot	0.021	0.196	0.178
Hemi-Sphere	0.168	1.782	0.0012
Sake Pot	0.507	4.597	0.0018

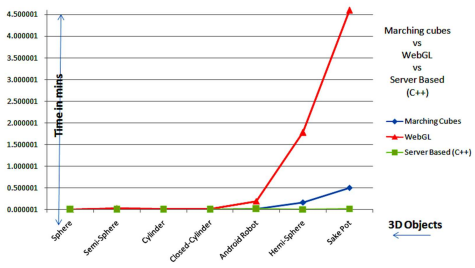

Figure 9. Chart diagram shows the time taken by each 3D model when applying different rendering techniques.

power of GPU needed to perform such rendering,

As a result, the objects rendered with Marching Cubes are ten times lesser in rendering time and GPU power, and it is perfect for users with low GPU performance, and/or low Internet bandwidth, while clients with high GPU performance can rely on hybrid WebGL. Users with low GPU performance and high Internet bandwidth can use server side rendering where objects with high resolution can be rendered (Fig. 6).

The bar chart in Fig. 10 compares the rendering time taken to obtain the same object using the three rendering techniques, it is clear that the hybrid WebGL rendering method came in the first place in terms of GPU high performance, and high rendering time, the Marching Cubes came second with considerable performance on the GPU and rendering time, and last came the server side rendering method using C++, which is very efficient in both GPU power and timing.

V. CONCLUSIONS AND DISCUSSION

In this paper, we have considered the features of a Client-Server adaptive architecture to establish the most convenient and efficient way of 3D Web-based modelling and rendering with a particular emphasis on the latter. We started from outlining some advantages of using the Function Representation over polygon meshes for modelling system; then we described the specifics of an interactive client-server architecture for modelling and rendering, and identified four most common scenarios of executing those processes along with the necessary communication and decision making using that adaptive architecture.

The main results of the paper are concerned with consideration of rendering. To explore the specifics of the proposed architecture with varying client and server abilities in the context of the outlined scenarios we have implemented three

 ISBN: 978-1-61208-325-4

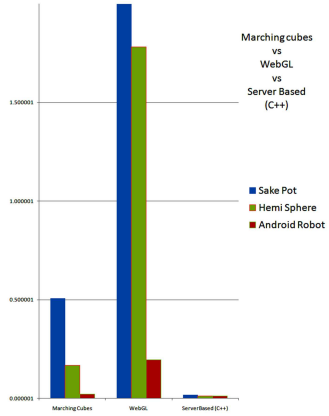

Figure 10. Bar chart diagram comparing the rendering time taken by each complex model when rendered using three different techniques.

different rendering techniques, namely: Server-based rendering using C++ code (with sending still images to the client); polygonization on the server and rendering the resulting mesh on a client using WebGL; Marching Cubes based polygonization implemented as JavaScript code on the client side. First, we showed as a prove of concept that rather simple objects can successfully be rendered using different techniques. Then, two more complex models (Hemi-sphere and Sake pot) were rendered using the three identified methods. The detailed comparative numerical and visual results have allowed us to make the following conclusions:

1) Gathering information about a client requesting the service is a key factor in determining what kind of service (i.e., rendering technique) to deliver
2) Different scenarios should be considered, the proposed architecture allows for reliable and efficient rendering process
3) Different rendering techniques can be applied over objects of various degree of complexity, although the required GPU power and the necessary rendering time can vary in a significant extent for simple and complex models.
4) 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
5) Rendering using Marching Cubes is very efficient in both rendering time and GPU power as it is done at the client side; however, this method properly works only for low resolution objects.
6) Rendering using hybrid WebGL and Marching Cubes techniques allows for high resolution objects and proved to be the optimal solution on machines with high GPU power.
7) Building up an adaptive environment, which is capable to interact and deal with different kinds of users is a challenge, and still needs further research.

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

- More experiments on more complex objects should be executed to further analyze the behavior of these objects in terms of their modelling and rendering within the proposed environment.
- Other rendering techniques, such as ray casting and volume rendering should be investigated.
- Implementation of an intelligent engine in the core of the adaptive environment promises more functionality for making decisions and thus for providing better rendering services.

Future work will include exploration of collaborative web-based modelling and rendering of heterogeneous objects with a complex internal structure in the context of a flexible interactive adaptive architecture including testing different scenarios of modelling and rendering with taking into consideration different multiple platform configurations.

REFERENCES

[1] R. Carey and G. Bell, The annotated VRML 2.0 reference manual. Addison-Wesley Longman Ltd., 1997.

[2] “OpenGL ES 2.0 for the web,” http://www.khronos.org/webgl/.

[3] R. Balan and G. Taubin, “3d geometry compression and progressive transmission,” Computer-Aided Design, no. 32, 2000, pp. 825–846.

[4] T. Vilbrandt, O. Fryazinov, C. Stamm, and A. Pasko, “A web oriented function-based volume modeling framework,” Computer Graphics & Geometry, vol. 12, 2010, pp. 41–51.

[5] R. Bürger and H. Hauser, “Visualization of multi-variate scientific data,” in EuroGraphics 2007 State of the Art Reports (STARs), 2007, pp. 117–134.

[6] Y.-C. Kao and G. C. Lin, “Development of a collaborative cad/cam system,” Robotics and Computer-Integrated Manufacturing, vol. 14, no. 1, 1998, pp. 55 – 68.

[7] C. V. Foster, Y. Shapirstein, C. D. Cera, and W. C. Regli, “Multi-user modeling of nurbs-based objects,” in ASME Design Engineering Technical Conferences, Computers and Information in Engineering Conference (DETC 2001/CIE-21256), ASME, 2001.

[8] S. Chan, M. Wong, and V. Ng, “Collaborative solid modeling on the www,” in Proceedings of the 1999 ACM Symposium on Applied Computing, ser. SAC ’99. New York, NY, USA: ACM, 1999, pp. 598–602.

[9] R. Bidarra, E. Berg, and W. F. Bronsvoort, “Interactive facilities for collaborative feature modeling on the web,” in Proc. of the Tenth Portuguese Conference on Computer Graphics, 2001, pp. 43–52.

[10] “X3D - extensible 3D, New-Generation Open Web3D Standard,” http://www.web3d.org/x3d/.

[11] J. Congote, A. Segura, L. Kabongo, A. Moreno, J. Posada, and O. Ruiz, “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, 2011, pp. 137–146.

[12] C. Schwartz, R. Ruiters, M. Weinmann, and R. Klein, “Webgl-based streaming and presentation framework for bidirectional texture functions,” in The 12th International Symposium on Virtual Reality, Archeology and Cultural Heritage VAST 2011, Eurographics Association. Eurographics Association, Oct. 2011, pp. 113–120, best Paper Award.

[13] J. Behr, Y. Jung, J. Keil, T. Drevensek, M. Zoellner, P. Eschler, and D. Fellner, “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, 2010, pp. 185–194.

[14] T. Erl, Service-Oriented Architecture: Concepts, Technology, and Design. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2005.

[15] D. Koller, M. Turitzin, M. Levoy, M. Tarini, G. Croccia, P. Cignoni, and R. Scopigno, “Protected interactive 3d graphics via remote rendering,” ACM Trans. Graph., vol. 23, no. 3, Aug. 2004, pp. 695–703.

 ISBN: 978-1-61208-325-4

[16] P.-A. Fayolle, B. Schmitt, Y. Goto, and A. Pasko, “Web-based constructive shape modeling using real distance functions,” IEICE - Trans. Inf. Syst., vol. E88-D, no. 5, May 2005, pp. 828–835.

[17] R. Cartwright, V. Adzhiev, A. A. Pasko, Y. Goto, and T. L. Kunii, “Web-based shape modeling with hyperfun,” IEEE Computer Graphics and Applications, vol. 25, no. 2, 2005, pp. 60–69.

[18] Q. Liu and A. Sourin, “Function-defined shape metamorphoses in visual cyberworlds,” Vis. Comput., vol. 22, no. 12, Nov. 2006, pp. 977–990.

[19] J. Parulek, P. Novotny, and M. Sramek, “Xisla development tool for construction of implicit surfaces,” in In SCCG 06: Proceedings of the 22nd spring conference on Computer graphics, Comenius University, Bratislava, 2006, pp. 128–135.

[20] H. Grasberger, P. Shirazian, B. Wyvill, and S. Greenberg, “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, 2013, pp. 29–37.

 ISBN: 978-1-61208-325-4