Computing (2013) 95 (Suppl 1):S661–S673
DOI 10.1007/s00607-012-0275-z

Efficient visualization of 3D models by web browser

Bartosz Sawicki · Bartosz Chaber

Received: 13 October 2012 / Accepted: 20 December 2012 / Published online: 8 January 2013

Abstract The article presents software module designed for efficient and convenient visualization of 3D models inside the web browser environment. It is written purely in JavaScript and takes advantages of the new HTML 5 standard. The authors focus on mobile devices, so special attention is given on efficiency and low network usage. Proposed solution based on progressive mesh streaming is compared with server side rendering approach. The results of usability tests performed on real-life random users group will be discussed.

Keywords 3D visualization · Mesh streaming · HTML 5 · Mobile devices

Mathematics Subject Classification 68U05 Computing methodologies and applications; Computer graphics; computational geometry · 68U35 Computing methodologies and applications; Information systems (hypertext navigation, interfaces, decision support, etc.) · 68M20 Computer system organization; Performance evaluation; queueing; scheduling

1 Introduction

The visualization of 3D models is an essentially important tool for today's numerical simulation science. Since the eyesight is the main sense for humans, it's not surprising that researchers want to visually investigate results of calculations and easily share

This work was partially supported by Polish Ministry of Science and Higher Education (research grant no. N N510 148838).

B. Sawicki (✉) · B. Chaber
Institute of Theory of Electrical Engineering, Measurments and Information Systems,
Faculty of Electrical Engineering, Warsaw University of Technology,
ul. Koszykowa 75, 00-662 Warsaw, Poland
e-mail: Bartosz.Sawicki@ee.pw.edu.pl; sawickib@gmail.com

them with others. Mobile devices (smartphones, tables) are natural tools for this kind of presentation, but many new challenges has to be overcome, mainly related with its limited computational power and transmission channel [7,12,19].

The most of numerical simulations take advantage of computational power concentrated in server clusters. There are two groups of techniques for displaying remote models: server side rendering and client side rendering. This basic division has sometimes different names (for example image-based vs. simplification [15]), but its importance should be understood. In the first solution, servers render 3D scene, compute view and send it as a image to the client, in the second approach, mesh data are transmitted to the client which finally renders it. The main advantage of server side rendering is that mesh doesn't have to be sent over the network, which is especially important when we deal with large scale models. On the other hand client side rendering could utilise hardware acceleration, and provide more natural user navigation. Unfortunately large meshes can't be rendered on the client side, because its computational power would be exceeded. That's where progressive mesh algorithms are applied [10,11].

Both, server and client rendering modes are subject of a wide scientific research. While investigations for first solution are focused on powerful rendering infrastructure [14] and image compression algorithms [7], the second approach more often linked with progressive meshes [4] and their transmission protocols [1]. It also describes progressive way of displaying 3D models, but please mind that this progressive method is very different from the progressive meshes method defined in [4].

Thinking about mobile devices connected with mobile network, we should take care of the bandwidth consumed by the visualization system. This topic is usually studied for games or augmented reality applications [12,19]. Smooth displaying of realistic 3D worlds on thin, mobile clients requires special techniques (mesh streaming, compressed meshes, server rendering). As it was shown by Pasman [15] optimal methodology should be a combination of above algorithms.

There are many, well established technologies for very efficient handling of presentation of 3D scenes, such as OpenGL, DirectX, GPU. However, to use them user is forced to install specialised software or even upgrade his computer hardware. In this paper we present different approach. Whole interaction with 3D model is done inside standard, modern HTML browser. This way we get unusual accessibility of developed software module, since web browser are nowadays present in virtually every device. Web3D consortium long time ago introduced technologies like VRML and X3D, but new HTML 5 standard is opening new possibilities for visualizing 3D objects. Build-in WebGL and Canvas components controlled by JavaScript code running inside the browser window can vanish last differences between desktop and web applications. Presented software module is just a block which can be used for building advanced cloud computing system. Authors have successfully used it in system devoted for bioelectromagnetical simulations [17].

Authors created software module capable to interactively display 3D mesh on devices running web browsers supporting HTML 5 standard (see Fig. 1). We were concentrated to provide natural 3D experience even for unprepared user. All parameters are automatically adapted to the size of processed mesh and mobile device capabilities. Since the core of mesh streaming algorithm was discussed in previous works [16], novelty of this paper lays in application of the concept to the web browser

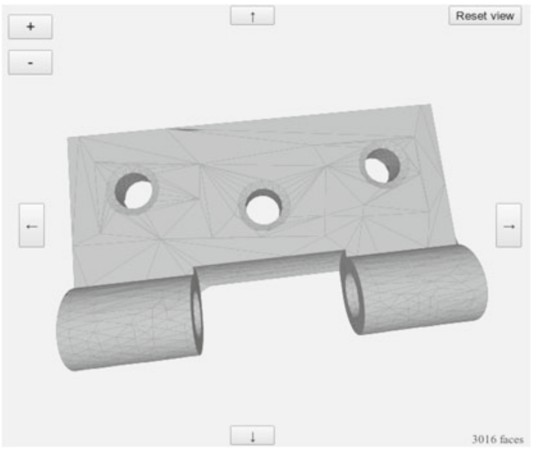


Fig. 1 Screen shot of developed web component while displaying hinge model

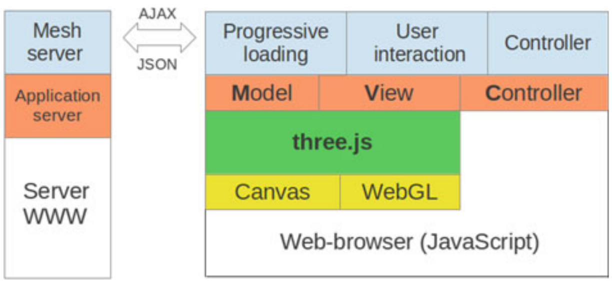


Fig. 2 Scheme of the system architecture. The MVC model is supported by Three.js [18] 3D rendering library

environment. Original contribution of the paper mainly consists of a network usage measurements analysis, the real usability tests and intuitive multi-mode interface.

Structure of this article goes as follows. The general technical overview of the developed web-component is presented in Sect. 2. Sections 3 and 4 are devoted to the most important elements (3D user interaction and progressive meshes implementation). Results of network usage tests are contained in Sect. 5, while real life validation for usability is described in Sect. 6.

2 General overview of the application

Developed system works in classic client-server regime, typical for web-applications. An AJAX transmission is used to send mesh in the JSON format. General view of system architecture is shown in Fig. 2. Model-View-Controller (MVC) programming pattern was adapted to make design simple and separate different layers of system.

Three.js [18] library was chosen as a rendering engine. It is a lightweight 3D JavaScript library which handle rendering of 3D scene into 2D view (scene, camera, lighting). Three.js provides three different renderers: canvas, svg and WebGL. Thus designed system will be flexible and portable. Three.js is the base for two MVC layers:

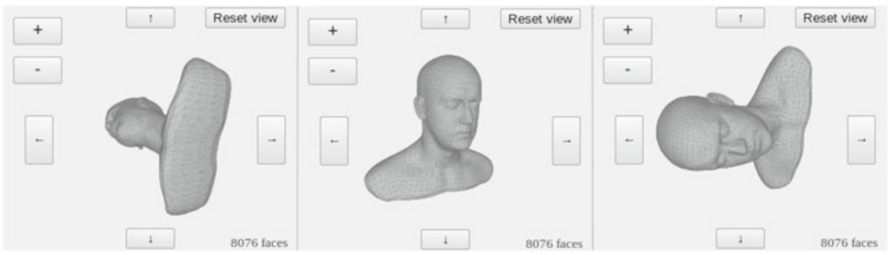


Fig. 3 Three connected components displaying the same model with different angles

a Model and a View. Model, in our case mesh, used by Three.js had to be extended by providing specialized functions for mesh streaming (described in Sect. 4). View, which is responsible for rendering view, was completed by the 3D user interaction (described in Sect. 3).

Our default renderer is based on simple Canvas element, it could be surprising for those who know that new HTML 5 technology introduced WebGL (JavaScript API to OpenGL framework). That's true, WebGL is hardware accelerated standard for 3D scene creation inside browser window. Very promising, but still very far from wide popularity. According to webglstats[1] only 2 % of the browsers on mobile devices supports this standard. We believe that the next few years will change completely this statistics, but it is currently not possible to rely on WebGL technology.

The controller (`viewController` prototype in JS code) is the main part of the system. It is designed to be standalone component of any webpage. The main HTML document namespace is not polluted, so several instances of the web3dviewer component can independently co-exists on the same page.

Components can be independent, but it is also possible to connect them and share mesh model data. By setting different default views one can build interesting three views panel (see Fig. 3), where model can be seen simultaneously from different angles. Lets take three connected components case and describe it as an example of usage:

```
<html> ...
<body onload="web3dviewer_init();">
<script type="text/javascript" src="web3dviewer.js"></script>
<div class="web3dviewer" id="main" mesh="MODEL" view="front"></div>
<div class="web3dviewer" master="main" view="top"></div>
<div class="web3dviewer" master="main" view="left"></div>
... </html>
```

Each of the components is simple `div` element with class `web3dviewer`. Elements of this class are automatically found by `web3dviewer_init()` function and initialized. The second obligatory property is `mesh`, which is set to the name of the model to be displayed. Properties `id` and `master` can be used to connect components and point where mesh data are assigned.

[1] www.webglstats.com, Florian Boesch, 2012.

Table 1 Navigation using three different interfaces: mouse pointer, touch and multi-touch

	Mouse pointer	Touch interface	Multi-touch
Pivot	Left-click and Move	Touch and move	Single-touch and move
Zoom	Roller	+/− buttons	Double-touch and distance
Pan	Right-click and move	Arrows buttons	Double-touch and move

For convenience and compactness, the whole code of the component is compiled into single file `web3dviewer.js`. JavaScript Closure compiler is used to perform this tasks. Compiled code is shorter, and usually runs faster.

3 User interaction in 3D

Basic interface for inspection of 3D models has to provide at least three operations (pivot, zoom and pan) [2]. Designing intuitive and natural user interface require to follow ”principle of least astonishment”, which says that user shouldnt be surprised by application behaviour. Such standard behaviour is well established for classic desktop computers, where mouse is used as a pointing device. However new mobile devices are usually operated by the touch or multitouch interfaces. The 3D interaction has to be redefined in this new environment [5]. Table 1 presents summary of actions implemented for different interfaces.

Presented system implements complete scene transformation which are needed to provide interaction to the user. Zoom and pan operations are simply translated into camera moves. For zoom-in camera goes closer, zoom-out camera goes back. The pan mode moves camera sideways. More complicated is implementation of pivot (rotation) of the model. It is convenient for user to automatically locate centre of rotation in the center of the model. Also one have to remember that rotation should to be done by the axis related with camera, which is natural for the user.

Multi-touch interfaces are supported by the most of modern browsers, and accessible by JavaScript events. API interface `TouchEvent` defines events `onTouchStart`, `onTouchMove`, `onTouchEnd`, which are handeled in our system. List of coordinates of touching fingers can be found in attribute array `touches`.

Another important aspect of 3D interface is so called supervised navigation [6]. Inexperienced user can easily get lost in the 3D virtual reality. To prevent such situation, interaction should be supervised by the system. Reduction of misunderstanding could be achieved by introducing limitations in navigation space or by the simple reset button. In our case ‘Reset view’ button (as seen in Fig. 1) restore default scene settings and ensure that model will be visible.

4 Mesh streaming

In order to provide user an responsive interface for 3D objects inspection regardless of presenting device, smart client-server architecture has been used. Using server for streaming 3D objects to client enables better control over data sent to client.

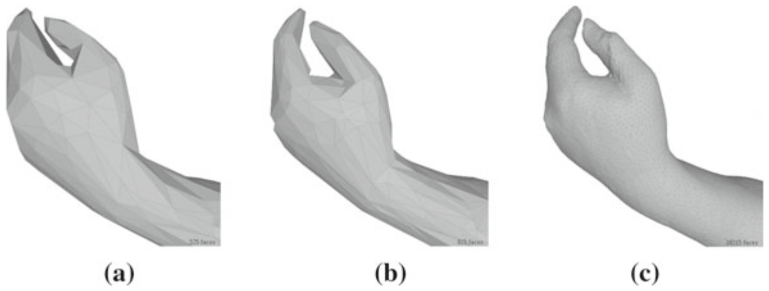
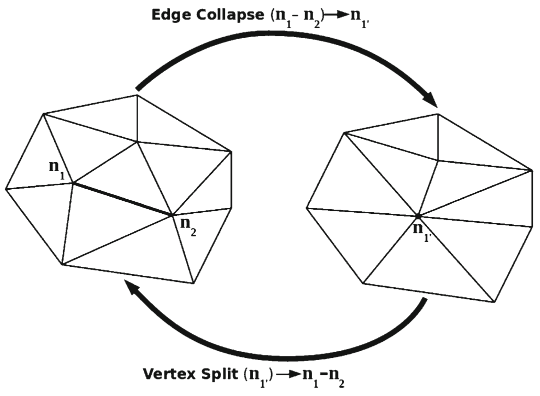


Fig. 4 Three snapshots during the progressive mesh loading of a hand: **a** base mesh with 525 faces, **b** model with 815 faces, **c** final model with 16,215 faces

Fig. 5 Illustration depicting reversibility of edge collapse of edge between $n_1 - n_2$


Edge Collapse $(n_1 - n_2) \mapsto n_{1'}$
n_1
n_2
$n_{1'}$
Vertex Split $(n_{1'}) \rightarrow n_1 - n_2$

Streaming 3D surface meshes is very similar to streaming JPEG files. At first, client receives very coarse version of original file, without neither all details nor good resolution. However, after rendering this image, user is ready to make first decision, e.g. to open another image. If image seems to have acceptable content user can wait while picture is being enhanced, and more features are visible (see Fig. 4).

Progressive meshes introduced by Hoppe in 1996 [3] define entity called base mesh. The base mesh is analogous to first, coarse version of JPEG compressed image. It is a simplified version of original mesh reflecting its main features. Beside base mesh, progressive mesh holds also sequence of operations increasing details of presented mesh. This way, after all operations are applied user is presented original mesh which has all detail. It must be stated, that user is able to inspect mesh just after base mesh is received. Then, even while mesh is rotated, translated or scaled details are appearing and whole mesh is refined.

Figure 5 shows how edge-collapse and vertex-split operations work, and complement each other. This way, by performing edge collapse operations, details are detached from base mesh, stored in intermediate way in sequence of edge collapse operations and therefore can be transmitted separately.

This paper is not focused on the details of processing 3D surface meshes into progressive format as it has been already described in [16]. Algorithm presented previous papers is enhanced version of Hoppe's method for compression and decompression of surface meshes. Here, compression process is performed on the server side as it was done previously, so details will be omitted. In this work, we have revised JavaScript implementation of vertex split operation in order to improve its performance and correctness.

One of the main important aspects of mesh simplification is choosing correct order of edges to collapse. Original Hoppe approach took advantage of global energy function which was minimized. Our algorithm is more local and efficient [16]. It takes into account change of the volume introduced by particular edge collapse. Because volume difference can be easily calculated using only points adjacent to considered edge this method is very fast and provides good results. Similar solution can be found in [9], however their method of calculating the volume change is different.

Transmission of progressive mesh to the client is complicated process than consists of four basic stages:

1. standard mesh has to be converted into progressive format (can be done off-line and only once per single mesh file),
2. base mesh is sent to the client (just like original mesh),
3. bundle of vertex split operations are iteratively sent to the client,
4. client apply vertex split operations and progressively restore original mesh quality.

Fortunately, the most heavy computations are performed on server side (in stage 1) which should be capable of handling such a big jobs. Operations 2–4 are performed on each client. This type of processing is well suited into server–client communication. What is more, process of unfolding/vertex splitting is straightforward and does not requires much of computational power. In our paper we present implementation of enhancing mesh geometry and topology implemented in plain JavaScript.

The main aspect of splitting vertex into an edge is reconnection of some of faces adjacent to splitting vertex to new vertex which will form a new edge. In fact, to be able to reconstruct such connection one has to know 8 parameters: two indexes of vertices forming edge $a - b$, two indexes of vertices originally connected to collapsed edge $(c0, c1)$, one direction vertex d determining which faces have to be reconnected and three components of displacement vector. Displacement vector is a vector representing translation between original position of vertex and it's position after edge collapse. It is worth noting that every edge is collapsed to one vertex which is located in the geometric center of the edge so single vector is sufficient for calculating position of both edge's vertices. Figure 6 presents the idea of first five parameters of vertex split operation. More details regarding vertex splitting algorithm are presented in [16].

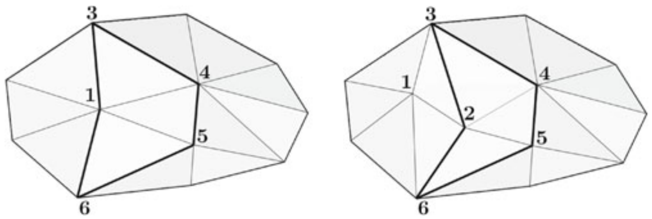


Fig. 6 Vertex split operation of vertex $a = 1$ into edge $a - b$ $(b = 2)$. Common vertices $c0 = 3, c1 = 6$, and direction node $d = 4$

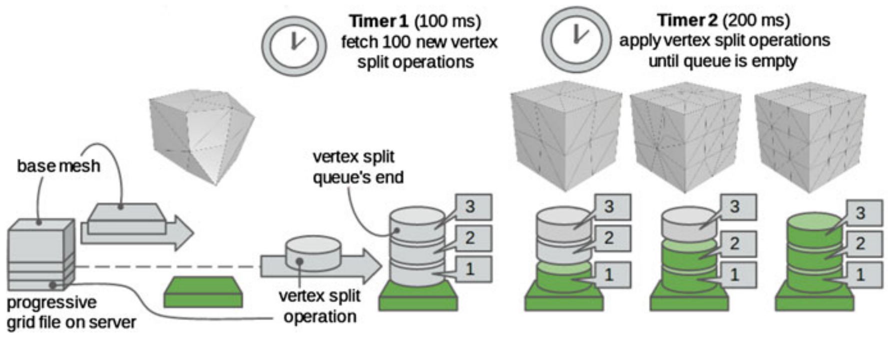

Timer 1 (100 ms)
fetch 100 new vertex split operations
Timer 2 (200 ms)
apply vertex split operations until queue is empty
base mesh
vertex split queue's end
3
2
1
progressive grid file on server
vertex split operation

Fig. 7 Internal mechanism of handling refinements utilizes two JavaScript timers and the simple queue of vertex split operations to prevent concurrent mesh modification

All communication between server and JavaScript client is done using JSON format. JSON is text based universal container capable for transmission of all kind of data. In case of our algorithm, single vertex split operation is sent as:

```
{
  a: 3, b: 5,
  c0: 4, c1: 6, d: 7,
  x: 1.3, y: 1.2, z: 0.2
}
```

The main improvement of the JavaScript implementation of vertex split is its adaptation for working in asynchronous environment. In order to do this, two JavaScript timers periodically executing appropriate actions were used. Since JavaScript doesn't provide threads and we don't want to freeze user interface when mesh is progressively loading, some aspects of implementations should be clarified. As presented in Fig. 7 implementation is based on the queue and two JavaScript Timers. One timer (Timer 1) is firing an AJAX call to web server asking for 100 new vertex split operations, which are pushed to the queue after server responds. The second timer (Timer 2) is sequentially applying vertex splits until the queue is empty.

Existence of those two timers and the queue is very important because of asynchronous nature of AJAX calls. We are not able to predict how much time it will take for the device to apply vertex split or how long the refinements will be downloaded from the web server. To prevent concurrent modification of the mesh or disturbance of the vertex splits order, the queue is crucial JavaScript component.

The queue on the client side is implemented using plain JavaScript list object. Timer 2 runs `refine` function. It pulls the last element from the queue, and forwards it to the `splitNode` function. Refinements retrieved from web server are appended to the beginning of the queue, so the most recently acquired packets are at the end of the list.

5 Network usage tests

In introduction section it was mentioned that there are two main concepts of remote 3D models visualization: server side rendering, and client rendering. Since our approach

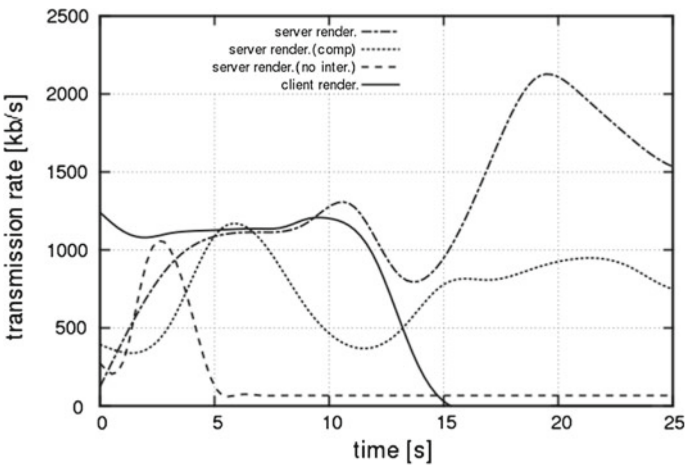


Fig. 8 Smoothed transmission rate during 25 s of interaction. Note that the client renderer transfer vanishes after 15 s (when mesh is fully loaded), while in case of the server renders transmission will be continued till user ends interacting with 3D object

is representing second idea, we decided to perform comparison for both solutions. Our criterion is based on network usage efficiency, which isn't usually issue in local networks (LAN), but definitely should be considered in case of mobile devices using mobile network.

During research we have compared our implementation of progressive meshes with the same problem solution based on server rendering by the use VNC protocol [21]. VNC is popular software for remote control over server environment. It addresses problems of accessing and interacting with applications and resources on remote server using thin client. We choose VNC, because it is a state-of-the-art product, where a lot of attention was paid to the efficiency of communication. Furthermore it was already successfully used in rendering clusters [14].

The first factor of comparison was general sensation of interaction with model and its responsibility. After short tests we stated that impressions using progressive mesh format with JavaScript inside web-browser are similar to the VNC server rendering. Both solution provide natural and smooth interaction for the user.

To precisely analyse network communication whole traffic was captures. The plot in Fig. 8 clearly show difference between the methods. There is one solid line for client renderer and three lines for server side rendering. As we can see after approximately 15 s, progressive mesh algorithm has loaded the mesh, so after that time no transmission is visible. In the case of VNC with no interaction, only one peak at the beginning of the test can be realised (the first, full image has to be loaded). After that, only the minimal traffic is observable. Normal, active session using server side VNC system shows that transmission rate never will be zero, and depends on size of the viewing window and type of interaction. It is shown that compression can significantly reduce bandwidth utilization.

The main purpose of this test was to compare how much data is sent between client and server during different algorithms. Since it is natural that amount of transferred

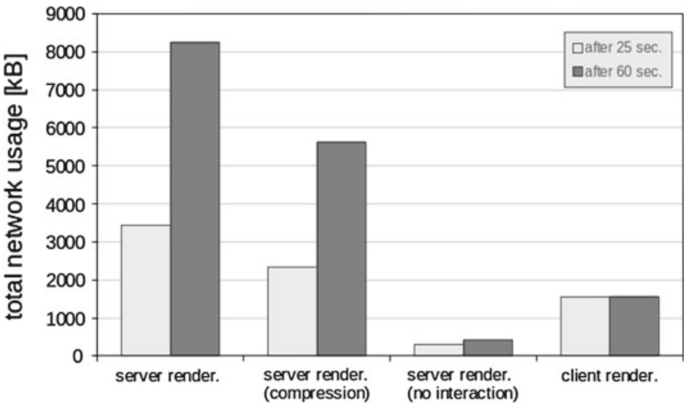


Fig. 9 Total network transmission after 25 s inspection of 9,000 facets model. Comparison for server side rendering based on VNC protocol, and client side rendering implemented in web3dviewer

data strictly depends on the size of the model and the time of measurements, we decided to present values for 25 and 60 s. Investigations had been conducted using 3D surface model of the hinge consisting of over 9,000 faces (base mesh for progressive model had only 76 elements).

The bar graph presented in Fig. 9 visualizes total accumulated network usage. Since complete loading of progressive mesh takes 15 s, client renderer bars are equal for 25 and 60 s. Far less preferred situation can be seen on server renderer bars, where network usage is nearly linear function of time.

It is not possible to identify the clear winner in this comparison. If navigation was short and lazy, server side renderer would need less data. But once it takes more time and will be more intensive, client renderer will show its superiority. With increase of interaction with the object, cost of image-based transmission becomes far more significant then when using progressive mesh transmission. It has to be noted, that we have compared VNC technology which is binary protocol with compression features, with simple plain-text JSON-based progressive mesh transmission implemented in web3dviewer. Further works should consider using techniques of compressed progressive meshes [8,13], as one the possible ways to cut down the size of transmission.

6 Conference experiment

The system was publicly presented on conference ESCO 2012, which took place in Pilzen, Czech Republic on June 25–29, 2012. During the poster session participants were attracted by a large QR-Code (see Fig. 10) to take out their mobile devices and scan it. Then they were redirected to the internet address, where web3dviewer was demonstrated.

Users were encouraged to verify usability of presented software, test performance of their devices, and share result with others. This way authors were able to collect results for 15 different mobile devices (see Table 2).

Fig. 10 Real life usability tests in the front of the poster (on the *left* QR-Code visible)

Table 2 Results captured during ESCO 2012 conference poster session

Participants were encourage to test performance of their mobile devices. The result is a number of facets in which frames-per-second ratio drops below 20

Class	Device	Browser	Result
Smartphone	Motorola Defy	Android	96
	HTC Sensation	Android	218
	iPhone 4	Safari	136
	HTC Desire	Opera	fail
	HTC Wildfire	Dolphin	96
	Nokia 5800XM	Opera	76
	Samsung Nexus	Android	156
	HTC One V	Android	136
	HTC Desire	Maxthon	136
Netbook/Tablet	Samsung Galaxy Tablet	Android	296
	MSI Wind270	Firefox	600
	Apple iPad 1	Safari	136
Laptop	HP Ellitebook	Firefox	2,500
	ThinkPad X201	Opera	3,256
	Dell Vostro V131	Chromium	3,576

The experiment was constructed in such way that basic mesh loaded at the beginning was very simple (76 facets). Then model where iteratively refined until device can't handle animation with 20 frames per second. Final number of loaded facets were recorded as a result.

The main aim of the experiment was to verify developed software on real life situation, with random variety of devices and different user skills. Majority of the group has never used their mobile devices to visualize 3D models, but using web3dviewer were immediately able to inspect provided mesh. This proves that developed software is accessible and usable even for incidental, unprepared users.

We observe only one case where Opera browser failed after loading our component. It could be related with errors in particular device configuration, since Opera browsers performed correctly in other cases.

Results presented in Table 2 shows that mobile devices can be grouped into three classes: smartphones, tablets and laptops. Smartphones where able to render less than 200 facets, laptops more than 2,000, results for netbooks/tablets are between those values. That way, using randomly selected devices, we confirm predictions formulated in previous works [16], that 200 facets is reasonable size of base mesh, which could be acceptably displayed by virtually any device.

7 Summary

The article presented web component based on modern HTML 5 technologies. The main objective was to create solution for efficient visualization of 3D models with high usability and portability. It was achieved by developing open-source JavaScript application called web3dviewer [20]. An advantage of progressive meshes streaming and modern multi-touch interfaces have been taken. System automatically fits complexity of the model with capabilities of displaying device.

Usability of the system and its performance in real life situation was positively verified by participants of ESCO 2012 conference. Results obtained from randomly chosen mobile devices allowed to calibrate optimal size of the base mesh for the progressive mesh algorithm. In addition, it was shown that the network usage of proposed architecture is significantly lower than systems based on server side rendering. System proved to be efficient and convenient way for visualization of 3D models in the web browser environment. The main disadvantage is related with additional computational effort which has to be taken on the server side to prepare mesh before displaying. One can understand it as shifting complexity to the server, to save client power and network usage.

Future work should cover expanding features of the system by introducing different types of the 3D models, like scalar and vector fields or stream lines. The popularity of WebGL technology should be attentively observed, since it would require to update system settings.

References

1. Cheng W, Ooi WT, Mondet S, Grigoras R, Morin G (2011) Modeling progressive mesh streaming: does data dependency matter? ACM Trans Multimed Comput Commun Appl 7(2):1–24
2. Chris Hand (1997) A survey of 3D interaction techniques. Comput Graph Forum 16(5):269–281
3. Hoppe H (1996) Progressive meshes. ACM SIGGRAPH 1996 Proceedings, pp 99–108
4. Hoppe H (1998) Efficient implementation of progressive meshes. Comput Graph 22(1):27–36
5. Fiorella D, Sanna A, Lamberti F (2010) Multi-touch user interface evaluation for 3D object manipulation on mobile devices. J Multimodal User Interfaces 4:310
6. Fitzmaurice G, Matejka J, Mordatch I, Khan A, Kurtenbach G (2008) Safe 3D navigation. I3D '08, Proceedings of the 2008 symposium on Interactive 3D graphics and games
7. Lamberti F, Sanna A (2007) A streaming-based solution for remote visualization of 3D graphics on mobile devices. IEEE Trans Vis Comput Graph 13(2):247–260
8. Lavou G, Dupont F (2012) Plateforme web pour le streaming de donnes 3D compresses. Proceedings of VISU 2012, Paris
9. Xinguo L, Hujun B, Qunsheng P, PhengAnn H, TienTsin W, Hanqiu S (2000) Progressive geometry compression for meshes. Proceedings of the 8th Pacific Conference on Computer Graphics and Applications, pp 408

10. Ling Y, Kong D, Jiang Z (2010) An improved progressive mesh model for mobile. Computer science and information technology (ICCSIT), 2010 3rd IEEE International Conference on Devices, vol 6, pp 69–73
11. Ma J, Chen B, Chen Q, Gong Y (2011) A strategy for generating progressive meshes and its transmission. J Inf Comput Sci 8(2):370–376
12. Nurminen A (2007) Mobile, hardware-accelerated urban 3D maps in 3G networks. Proceedings of the 12th international conference on 3D web technology, pp 7–16
13. Renato Pajarola, Jarek Rossignac (2000) Compressed progressive meshes. IEEE Trans Vis Comput Graph 6(1):79–93
14. Paul B, Ahern S, Wes BE, Brugger E, Cook R, Daniel J, Lewis K, Owen J, Southard D (2008) Chromium renderserver: scalable and open remote rendering infrastructure. IEEE Trans Vis Comput Graph 14(3):627–639
15. Pasman W, Jansen FW (2003) Comparing simplification and image-based techniques for 3D client-server rendering systems. IEEE Trans Vis Comput Graph 9(2):226–240
16. Sawicki B, Chaber B (2012) 3D mesh viewer using HTML5 technology. Przeglad Elektrotechniczny (Electrical Review), ISSN 0033–2097, R. 88 NR 5a/2012, pp 155–158
17. Sawicki Bartosz, Chaber Bartosz, Starzynski Jacek, Szmurlo Robert (2012) Internet application concept to trivialize EMF biomedical computing. COMPEL: Int J Comput Math Electr Electron Eng 31(4):1190–1197
18. Ricardo C (2012) (mrdoob), three.js 3D JavaScript library. http://github.com/mrdoob/three.js
19. Xun W, Bai-Lin Y (2012) 3D models simplification algorithm for mobile devices. J Softw 7(11)
20. web3dviewer (2012) 3D viewer using HTML5 technology. http://code.google.com/p/web3dviewer/
21. Richardson T, Stafford-Fraser Q, Wood KR, Hopper A (1998) Virtual network computing. IEEE Internet Comput 2(1):33–38