Bartosz SAWICKI, Bartosz CHABER

Warsaw University of Technology

3D Mesh Viewer Using HTML5 Technology

Abstract. *The effective visualization of 3D meshes based on the build-in features of a web browser supporting HTML5 (Canvas, WebGL) standards is presented. The algorithms related with progressive mesh streaming over Internet network are discussed. It has been demonstrated that using described software, 3D models could be easily visualized on any modern, mobile device (internet tablets, smart-phones or netbooks).*

Streszczenie. *Artykuł prezentuje wydajną metodę wizualizacji siatek trójwymiarowych przy pomocy mechanizmów wbudowanych w przeglądarki internetowe obsługujące standard HTML5 (Canvas, WebGL). Przedstawiono algorytmy progresywnego przesyłania siatki poprzez sieć internet. Zademonstrowano sposób, który pozwala na łatwą, interaktywną pracę z obiektami 3D na współczesnych, przenośnych urządzeniach takich jak tablety, smartfony, czy netbooki. (**Wykorzystanie standardu HTML5 do wizualizacji siatek trójwymiarowych**)*

Keywords: wizualizacja trójwymiarowa, Canvas, WebGL
Słowa kluczowe: 3D visualization, Canvas, WebGL

Introduction

For several years a three dimensional reality is widely present in the computer science industry. The most popular applications are related with games placed in the virtual, realistic worlds. This paper is addressed to the visualization and inspection of scientific simulations results. With instantly growing computational power, numerical simulations is becoming the most popular scientific tool. For the same reason, visual inspection of 3D models is every day task for many researchers.

The OpenGL is well established programming standard how to efficiently describe 3D scene, and render it on computer screen. The OpenGL works as a part of classic operating system, where standalone application has direct access into the hardware. Nowadays software is more often going to be served as a service (SaaS) using Internet network. In this paper we follow this modern trends. We will discuss visualization of 3D objects inside a Internet browser window. History of the technologies designed for 3D visualization inside the WWW browsers started in 1995 when VRML (Virtual Reality Modeling Language) was introduced [1]. This text based meta-language inspired by HTML was proposed for describing 3D scenes in terms of geometry and material properties. For the rendering of the scene it was required to install a platform specific plug-in. Nowadays, VRML should be considered as an abandoned project, but X3D language could be treat as a its successor.

For few years, the Java Applets are probably the most popular solution for 3D in a web browser. An applet is a small application written in Java which is embedded inside the HTML code of a webpage. To run it, special plugin is required as well as Java Virtual Machine. The implementation of the JVM on all operating systems made Java applets ubiquitous. Java has build-in features for binding to OpenGL called JOGL which gives control on the 3D graphics hardware.

Adobe Flash and Microsoft Silverlight are another two external and popular technologies (plugins) which are designed to enrich web browsing experience. None of them has full 3D accelleration, but using vector graphic features it is possible to visualize 3D objects. Special libraries (like Five3d [4]) makes writing 3D webpages much easier in the Flash environment.

HTML5 3D viewer

The HTML5 is new version of HTML which is core language of WWW network. Even though standard is still under development (June 2011), it is already supported by majority

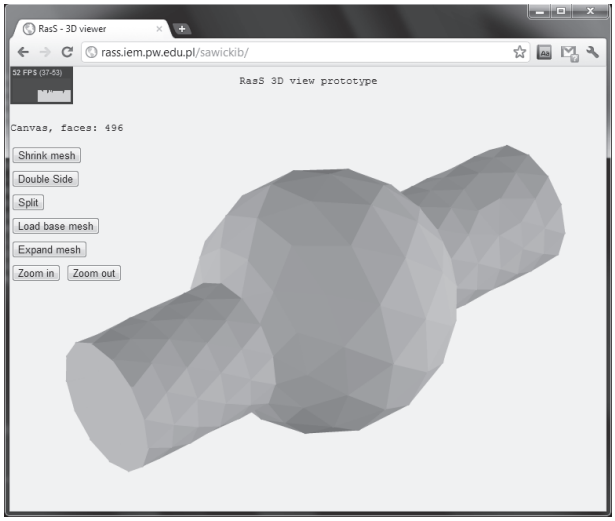

Fig. 1. Screenshot of 3D viewer prototype inside 3D Chrome browser window.

of modern browsers. HTML5 introduced several new technologies, which are designed to vanish last differences between desktop and web applications. The graphic interface has been benefited by Canvas and WebGL. These new components are closely related. Canvas is general element for 2D drawings, which is extended by 3D context by WebGL. All of features are programmable from Javascript language.

The WebGL standard is managed by Kronos Group [3]. The library enables direct acceleration of 3D objects using OpenGL ES. Programming OpenGL transformations could be time consuming, so plenty of middle-ware project in Javascript were established to facilitate 3D operations. Such as C3DL [7], Processingjs [8] or WebGLU.

In presented project (see prototype screenshot in Fig. 1) three.js library [5] has been deployed. The library supports WebGL rendering for GPU accelerated hardware, but also pure Canvas element rendering if such device is absent. That way we could get more flexible software component, running on majority of mobile devices.

Progressive meshes

Important advantage of the project is utilization of progressive mesh loading. As shown in the browser window on Fig. 3 initial mesh is very coarse, then simultaneously with the user interaction, mesh refinement is loading.

Progressive meshes are special format for representing triangle surface meshes. Meshes stored in such format can be visualized very efficiently. At first, mesh is coarse and inaccurate, but details emerge as more data is transmitted.

Progressive meshes were introduced by Hoppe in the paper published in 1993 [10]. He described set of operations on surface mesh which can be used to optimize its structure while introducing the lowest error. In further works [11], Hoppe has stated, that using only sequence of only one operation called `edgeCollapse` initial mesh can be reduced in such a way, that it could be reconstructed. For every `edgeCollapse` operation exists reverting operation called `vertexSplit`. After applying sequence of collapses, one can get much smaller base mesh. It could be transmitted and display much faster than original, and then refined by sending `vertexSplit` operations for reconstruction.

Deployment of this algorithms gives an opportunity to control level of details of visualized mesh. Thus, it can be useful in mobile devices using limited hardware. Moreover, we can save bandwidth, since the base mesh is far smaller than the original one and user can stop the transmission of refinements anytime.

Many implementations of progressive meshes where proposed [13]. They differ mainly by algorithms used to determine order of collapsed edges. Authors use their own implementation [12]. In our approach determination which edges will be collapsed is based on minimization of approximated change of total volume of mesh introduced by collapsing edge. Details of algorithm are omitted here, since this paper is more about the efficiency in mobile and web environment.

Before transmission over the network, original mesh has to be processed into progressive form on the server side. Because it is done off-line on a powerful server, collapsing algorithm isn't crucial part when thinking about efficiency of the system.

Vertex split algorithm

The vertex split algorithm is essential part of the progressive mesh reconstruction. Algorithm 1 and Fig. 2 presents how this operation works. Information needed for successful *edgeCollapse* inversion are packed in data set noted as a vs_data. Authors of the paper define this data as a five vertice indexes and one 3D vector. Beside vertices initially forming an edge (v_a and v_b), three other vertices are needed. Two of them are common vertices (c_0 and c_1) and one is direction vertex (n_d). Common vertices are a vertices which are connected to the v_a and v_b. Direction vertex is the vertex which determines set of other vertices to reconnect. The 3D vector is a distance (coordinates difference) between v_a and v_b.

Algorithm 1: Vertex split algorithm

aF $\leftarrow$ adjancentFaces(v_a)
faces $\leftarrow$ forReconnection(aF, v_a, vs_data)
for *face* $\in$ *faces* **do**
  $\{v_0, v_1, v_2\} \leftarrow$ faceVertices(face)
  if $v_0 = v_a$ **then**
    $v_0 \leftarrow v_b$
  if $v_1 = v_a$ **then**
    $v_1 \leftarrow v_b$
  if $v_2 = v_a$ **then**
    $v_2 \leftarrow v_b$
end
$\{v_a, v_b\} \leftarrow$ splitCoords(vs_data)
addNewVertex(v_b)
addNewFace($\{v_a, v_b, c_0\}$)
addNewFace($\{v_a, v_b, c_1\}$)

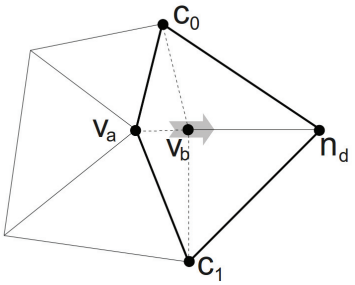

Fig. 2. Illustration for vertex split algorithm. Vertex v_a is split into v_a and v_b. Three new edges and two faces are created and reconnected with whole mesh.

System architecture

Designed system should be classified as a classical rich web-application. As could be seen in Fig. 3 system is divided into two parts. One is running inside the browser window (as a Javascript), and the second one is started by the Web server. Communication between browser and server is based on JSON [6] containers transmitted over http protocol.

The server part of the application (see Fig. 3, on the right) is operating on mesh stored in special progressive format (`mesh.pgrid`). In consists of the base tetrahedron and a vertex split dataset. And there are consequently two interfaces to receive progressive mesh: `base_mesh` and `refine_mesh`. The first is designed to get initial tetrahedron, then `refine_mesh` is iteratively called to get successive parts of vertex split vector.

Client side (see left in Fig. 3) is responsible for real object displaying and user interaction. As stated before all of this is controlled by Javascript supported by three.js library. In the background client side script is communicating with server, and loads vertex split operations.

Adaptive resolution

The algorithm presented in previous section provide a methodology to serve mesh in different levels of complexity. Now we are going to discuss strategies used to determinate maximum mesh complexity which could be managed by the browser running on a specific device. To control the process of refinement we need indicator of current hardware utilization. It should be universal and easy to calculated in Javascript environment. The most natural choice would be CPU load. Unfortunately, for security reasons Javascript code is separated in the sandbox without access to hardware measures. To find a solution we need to investigate animation technology provided by modern Internet browsers.

Majority of the animations on webpages are done using third party plugins, like Adobe Flash, but it is also well established technique to display frames of the animation by the use of a `setTimeout()` or a `setInterval()` functions. Apart from small differences, both of them works in similar way. They call another function or evaluates an expression at specified repetition (in milliseconds). This method is simple but also ineffective since rendering of complicated frame could cost a lot of numerical power. In 2010 developers of Mozilla Firefox proposed new API where browser can automatically control frequency of animation. In 2011 Google Chromium Team joined this project, and now we have official specification of a function named `RequestAnimationFrame()` [9]. In this new interface function is called as often as required to produce smooth animation. Currently it is adviced to make an animations using `RequestAnimationFrame()`, because this way browser could prevent situation when overload anima-

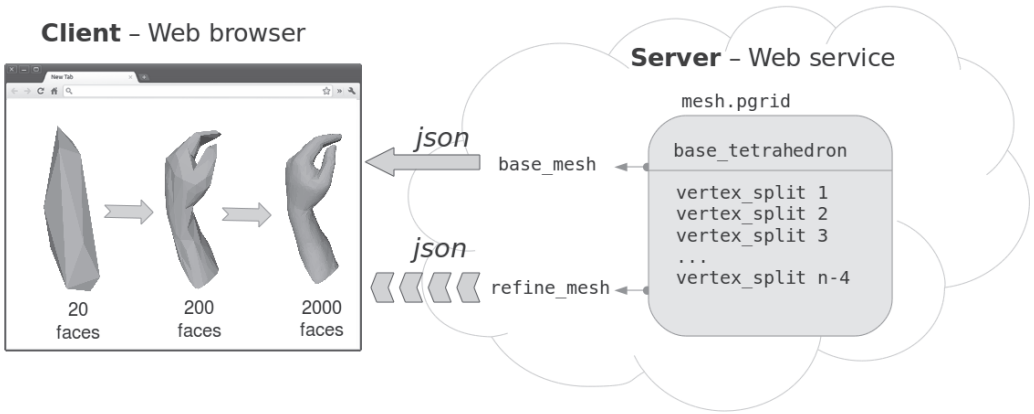

Fig. 3. Architecture of the system. On the left side, web browser running Javascript and communicating with application server (on the right) to obtain and display progressive mesh.

tion would destabilize whole browser application. An intelligent browser could also slow down animation when possible, and this way reduce energy consumption, which is especially important for portable devices.

To measure current CPU load we count how many frames are rendered in one second (FPS) by `RequestAnimationFrame()`. This is indirect indicator of hardware utilization. The problem is that not all browsers properly control speed of animation. We observed that sometimes browser isn't slowing animation down, even if CPU is overloaded. This bug will be surely eliminated in the next versions of browsers.

Performance tests

Developed software has been tested on three portable devices, each representing different class of products:

- smart phone (Samsung Galaxy S),
- tablet (Apple iPad 1),
- netbook (CPU: Intel Core Duo, 1.87GHz, RAM: 1GB).

The smart phone and the tablet run default internet browser application supporting HTML5, but without WebGL extention. On the netbook operated by Microsoft Windows 7, two different browsers were verified: Mozilla Firefox 5 and Google Chrome 10. To keep comparison fair, for all of the cases only universal Canvas renderer was choosen.

Performance tests was based on small mesh with 496 faces (as seen in Fig. 1). Starting from the base tetrahedron was slowly refined. After each set of vertex splits, performance as a number of frames per second was calculated and stored.

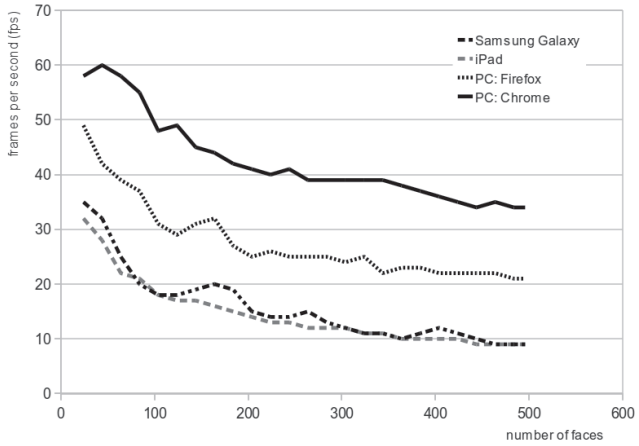

Fig. 4. FPS performance charts for different browsers/platforms

Results are presented in Fig. 4. One can see that, as expected, performance deteriorates while size of the mesh is enlarging. The smart phone and tabled showed very similar visualization capabilities, hence netbook results are much better. The clear winer is Google Chrome browser, which is known for efficient Javascript engine.

Conclusions

Described system proved to be useful for visualization of 3D objects on mobile devices. Progressive streaming technique allows to match number of displayed details to the capabilities of the hardware. Experimets showed that today 200 faces mesh is reasonable compromise for browsers supporting HTML5 Canvas standard.

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

BIBLIOGRAPHY

[1] E. Pinto, G. Amador, A. Gomes: A graphics library for delivering 3D contents on Web browsers Digital Content, Multimedia Technology and its Applications (IDC), 2010 6th International Conference on Seoul, 16-18 Aug. 2010
[2] C. Leung, and A. Salga: Enabling WebGL, Proceedings of the 19th international conference on World wide web, Raleigh, North Carolina, USA 2010
[3] Khronos Group, Webgl - opengl es 2.0 for the web. 2009, http://www.khronos.org/webgl/
[4] Mathieu Badimon, Five3D - Flash 3D library, http://five3d.mathieu-badimon.com/
[5] Mr. Doob, three.js, Javascript 3D Engine, https://github.com/mrdoob/three.js, 2010-2011
[6] Crockford, D., JSON (JavaScript Object Notation). http://www.json.org/.
[7] C3DL - The Canvas 3D JS Library, 2011, http://www.c3dl.org
[8] John Resig, processing.js, http://processingjs.org
[9] W3C Working Draft: Timing control for script-based animations, 2 June 2011, http://www.w3.org/TR/animation-timing/
[10] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, W. Stuetzle, Mesh optimization, Proceedings of the 20th annual conference on Computer graphics and interactive techniques, 1993
[11] H. Hoppe, Progressive meshes, Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, 1996
[12] B. Chaber, Zastosowanie siatek progresywnych do efektywnej wizualizacji wyników symulacji MES, master's thesis, Insitute of Theory of Electrical Engineering, Measurement and Information Systems, Warsaw University of Technology, 2011
[13] H. Hoppe, Efficient implementation of progressive meshes, Computers & Graphics, Volume 22, Issue 1, 1998

Authors: Bartosz Sawicki, Bartosz Chaber, Institute of Theory of Electrical Engineering, Measurement and Information Systems, Faculty of Electrical Engineering, Warsaw University of Technology, ul. Koszykowa 75, 00-662 Warszawa, Poland, email: sawickib@iem.pw.edu.pl