High performance 3D visualization on the Web: a biomedical case study

Jesús Jiménez, Jaime Cruz, and Juan Ruiz de Miras

Department of Computer Science, University of Jaén, Campus Las Lagunillas s/n, 23071 Jaén, Spain
{jjibanez,jctorre,demiras}@ujaen.es

Abstract. There are many desktop-based applications that offer a solution for biomedical problems. Usually, one of the most important task of these biomedical applications is the 3D graphics visualization. In last years, the development of web-based applications has taken a great importance and are defeating desktop-based applications, mainly, because the benefits this class of software has. However, until recent times, web developers were not able to directly run high performance graphics natively on a web context. But nowadays, the appearance of the Khronos WebGL standard make that limitation possible. This work summarizes the WebGL capabilities and presents a successful experience related to the inclusion of high performance graphics on the web. This is done by developing an interactive visualization of voxelized 3D models with the aim of analysing magnetic resonance images of the brain.

Keywords: 3D graphics, WebGL, Medical imaging, Box-Counting

1 Introduction

The medical image representation, both on a two- or a three-dimensional spaces, is a very important and interesting topic. For some years, several successful desktop applications have been presented to the research community [1–3]. Several of these software applications allow to read medical images, such as those obtained through magnetic resonance scanners, or to obtain a 3D virtual view of the captured element.

Nowadays, web-based applications are overcoming the classic desktop software development [4]. The software that resides on the cloud presents some advantages over local applications. Firstly, it is not necessary to install the software, the user only has to access to a webpage through his preferred web-browser. Therefore, with a web application we ensure that the user is using the latest version of the software, since the update process is done at once in the server instead of in each client machine, as happens with desktop applications. Secondly, the cross-platform character of the web-based applications could attract extra users, since they are not limited to a concrete operating system.

Web-based applications are becoming more and more powerful, and the performance distance between these ones and desktop applications is increasingly

closer. A traditional differentiator point between desktop and web applications is the capability of represent 3D graphics. During the last years, there have been different efforts on creating a development scheme for graphics representation on the web. Thus, one of the first and successful technologies was VRML (Virtual Reality Modelling Language) [5], later improved by X3D [6]. These two technologies were widely used by the scientific community and the developers for representing 3D models on the web. But VRML and X3D only offer the guidelines and sintaxis for representing the composition of a 3D scene by using plane text, in the first case, or XML documents, in the case of X3D. Therefore, the representation of the 3D scene will require the installation of especial plug-ins and 3D viewers for interpreting the text and labels, each one especially designed for a concrete web-browser. This fact avoided an easy standardization of both technologies, mainly due to the fact that the development of the plug-ins was not a browser responsibility, but it depended on third-part developments. Recently, a new technology has appeared: WebGL (Web-based Graphics Library). This new 3D API has been converted in a standard thanks to the support of the main web browser manufacturers. At the moment there are some web applications that benefit from using WebGL to deal with biomedical problems on the web, e.g. [7, 8].

The rest of the paper is organized as follows. First we describe the WebGL API, highlighting the libraries that facilitates its programming task. We then describe the biomedical case study on which we have tested the odds and capabilities of WebGL. Finally, we summarize in the conclusions section and expose some future works.

2 WebGL

WebGL is an API developed by the Khronos Group that extends the capability of the classic JavaScript programming language, allowing the generation of native 3D graphics in any compatible web browser, without needing extra plug-ins. It is not necessary to install any 3D viewer, since WebGL objects are shown on the different web browsers thanks to the new HTML5 *Canvas* element. The WebGL API [9] is based on the OpenGL ES 2.0 standard [10], so it enables a direct access to each GPU (Graphic Processing Unit) located on the client. WebGL has a cross-platform character and it is royalty-free. WebGL can easily access and interact with the HTML content through the Document Object Model (DOM) interface. Focusing on the performance of WebGL when managing 3D graphics, the results are very satisfactory, overcoming to all the others 3D Web technologies and showing frame rates close to the ones achieved with an standard OpenGL plus C++ implementation [11].

WebGL is directly interpreted and executed by each web browser, when working with any of the ones that comply with the WebGL standard (*Google Chrome*, *Mozilla Firefox*, *Apple Safari* or *Opera*). *Internet Explorer*, the Microsoft's web browser, does not offer support for this standard API.

WebGL presents a powerful but also low-level library, which may cause some difficulties when developing the web applications. With the aim of solving this problem and looking for a faster and easier development, some frameworks working over the WebGL layer have recently appeared. Among them stand out *Three.JS* [12], a powerful and efficient library with a lightweight character and an stable behaviour, which is the most established and widely-used WebGL library. *Three.JS* allows to easily manage some graphics programming aspects such us the camera, lights, shadows, or data loaders, among others.

By using *Three.JS*, complex 3D scenes could be easily created and embedded on HTML5 web pages, allowing the developer to focus his efforts on create richer content for the web.

Next, a pseudo code sample is presented. This sample shows how a simple rotated cube could be included in a HTML document by using *Three.JS*. The representation of a simple scene like the one presented in the code sample, would have required a deep knowledge of the graphics visualization pipeline and tens lines of code if the direct WebGL API had been used; a fact that becomes in even a more complex problem when developing advanced and complex 3D scenes.

```
<script src="lib/Three.js"></script>
<script>
   var renderer = new THREE.WebGLRenderer(); //WebGL Handler
   renderer.setSize(window.innerWidth,  window.innerHeight );
   //Add WegGL canvas to the DOM
   document.getElementById("parent_div").
                                  appendChild(renderer.domElement);
   //Camera definition:
   var camera = new THREE.PerspectiveCamera(fov, ratio, near, far);
   camera.position.z = 400;

   var scene = new THREE.Scene(); //Parent node

   //3D mesh definition: Geometry and Material
   var geometry = new THREE.CubeGeometry(n,n,n);
   var material = new THREE.MeshNormalMaterial();
   var mesh = new THREE.Mesh( geometry, material );
   mesh.rotation.x += 0.5; //Mesh Transformations
   //Lights definition:
   var light = new THREE.DirectionalLight(hex, intensity);
   light.position.set(x,y,z);

   scene.add(mesh); //Add mesh to the scene
   scene.add(light); //Add lights to the scene
   renderer.render( scene, camera ); //Render scene
</script>
```

(Pseudocode of a sample scene definition with WebGL through the *Three.JS* API)

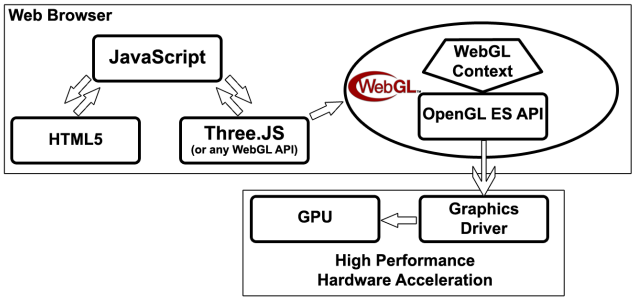

Fig. 1. WebGL system architecture.

In Figure 1, a scheme of the system architecture when using WebGL is depicted. As could be seen, JavaScript plays the intermediary role between HTML5 and the WebGL functionality. That functionality is accessed through calls to the *Three.JS* API, in our case, thus accessing to the WebGL context and consequently to the OpenGL ES API, both integrated on the web browser. Then, the browser communicates with the computer's graphic driver that finally translates the commands to the hardware, thus obtaining high performance graphics natively executed on the GPU.

3 A biomedical case study: Fractal Dimension estimation through the Box-Counting algorithm

The Fractal Dimension (FD) is a measurement of the topological complexity of an object, both on 2D or 3D. It has been demonstrated that by analysing the FD of medical images, many interesting results could be obtained [13]. Highlighting some of them, Esteban et al. present in [14] a FD analysis of the brain gray matter for an early detection of neurodegenerative diseases like Multiple Sclerosis; meanwhile, in [15] and [16] the white matter's cerebral structure and its degeneration with ageing is quantified also by using FD analysis; and a FD analysis of the complexity of the fetal cortical surface is performed in [17, 18]. An interesting review of the most relevant methods to calculate the FD and their application in the biomedical field is presented by Lopes and Betrouni in [19].

The box-counting algorithm [20] is one of the most widely used methods for estimating the FD value. As its name indicates, this algorithm consists on count how many boxes are necessary to cover a 2D or 3D object, previously discretized by using a threshold value. This process is repeated varying the edge size of the boxes (Figure 2). Thus, moving from a level to the next one implies the addition of one pixel to the edge box size. The value of the FD is calculated through a log-log linear regression in which the X axis represents the inverse of the box size, and the Y axis represents the number of counted boxes. The final value for the FD corresponds to the slope of the linear regression.

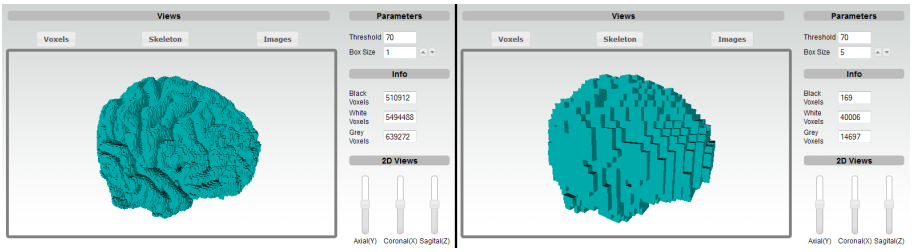

Fig. 2. Box-counting results. Two voxelized models (box size 1 pixel and 5 pixels).

In a previous work, we have developed a desktop application that allows the calculation of the 3D Fractal Dimension (3DFD) of magnetic resonance images (MRIs) [21]. In addition to the isolated 3DFD value, the software also offers an interactive 3D representation of the loaded object, and also a view of the different voxelized models obtained when applying the box-counting algorithm. By this way, the user can graphically understand how the box-counting algorithm works, and also what is the influence that the input parameters (i.e. the threshold value, or the box size) have on the considered 3D model. Next, we present how the interactive web-based version of this 3D graphic representation have been developed by using WebGL.

3.1 WebGL Implementation Details

In addition to the classic problems and difficulties associated with any 3D graphics-managing implementation, when developing with WebGL or any of its associated high-level libraries, the data which represents the 3D object (or whatever that has to be shown in the web browser) resides in a server computer, so it is necessary to perform a data transfer through the network. To minimize the performance impact of these data transfers, mainly in terms of computing time, a lightweight data-change format has to be used, for example a text format like JSON (`http://www.json.org/`), which is very compact, easy to read and write, and completely language independent.

In our implementation, once the box-counting, for a selected box size, has been calculated on the server, the result is transferred to the client web-browser as a JSON object, to visualize the 3D object through WebGL and *Three.JS*, as established along the paper. While the client is visualizing the model, and also while the user interacts with it, other set of box-counting calculation processes are launched on the server and then transferred and stored in the client using second-plane (so transparent to the user) AJAX (Asynchronous JavaScript and XML) calls. Thus, when the user wants to visualize another consecutive box size, the response is immediate since the new 3D grid has been previously calculated, received and saved.

Regarding the 3D representation of the different brain voxelizations, we applied two basic optimizations. First, the geometry is merged in only one mesh,

which reduces the number of objects and speeds their visualization. Second, when representing each voxel (cubes), the hidden faces are removed thus decreasing the complexity of the model.

Figure 2 shows two snapshots of our developed WebGL application, in which two brains at different resolutions are represented, together with the HTML controls that allow the interaction with the them.

4 Conclusions

The development of web-based applications are defeating to the desktop-based ones because the different benefits that the web software has. For this reason, nowadays JavaScript is not only a language used to perform little changes on a HTML document, it has become in a powerful tool that allows programmers to develop powerful and high performance applications.

In this work, we have presented a case study to show up how web-based biomedical applications with a high performance 3D graphics requirement can be developed thanks to WebGL. The fast popularization of WebGL is being mainly possible thanks to recently developed libraries and APIs built over WebGL, such us *Three.JS*. This facilitates the graphics programming, allowing developers to create powerful applications.

Besides WebGL, some novelties technologies are appearing in recent times for incrementing the high performance capabilities of web browsers. For example, Khronos WebCL [22] is a new standard for parallel computing on the web that warps OpenCL by using the JavaScript language. Another example is the JavaScript *Web Workers* [23], an API that allows to launch independent and parallel JavaScript scripts on a multi-core CPU. In the future, we will deal with these technologies with the aim of optimizing and improving web-based software.

Acknowledgments. This work has been partially supported by the University of Jaén, the Caja Rural de Jaén, the Andalusian Government and the European Union (via ERDF funds) through the research projects UJA2009/13/04 and PI10-TIC-5807.

References

1. Fedorov, A., Beichel, R., Kalpathy-Cramer, J., Finet, J., Fillion-Robin, J., Pujol, S., Bauer, C., Jennings, D., Fennessy, F., Sonka, M., Buatti, J., Aylward, S., Miller, J.V., Pieper, S., Kikinis, R.: 3D Slicer as an image computing platform for the Quantitative Imaging Network. Magnetic resonance imaging, DOI:10.1016/j.mri.2012.05.001. (2012)
2. Friese, K., Blanke, P. Wolter, F.: YaDiV - An open platform for 3D visualization and 3D segmentation of medical data. Visual Computer, 27(2) 129–139. (2011)
3. Fischl, B.: FreeSurfer. NeuroImage 62(2) 774–781. (2012)
4. Taivalsaari, A., Mikkonen, T., Anttonen, M., Salminen, A.: The death of binary software: End user software moves to the web. Proceedings - 9th International Conference on Creating, Connecting and Collaborating through Computing.(2011)

5. Ames, A.L., Nadeau, D.R., Moreland, J.L.: The VRML Sourcebook. (1996)
6. Brutzman, D., Daly, L.: X3D: extensible 3D graphics for Web authors. Morgan Kaufmann. (2007)
7. Cantor-Rivera, D., Bartha, R., Peters, T.: Efficient 3D rendering for Web-based Medical Imaging Software: a proof of concept. Progress in Biomedical Optics and Imaging - Proceedings of SPIE. (2011)
8. Jacinto, H., Kéchichian, R., Desvignes, M., Prost, R., Valette, S.: A web interface for 3D visualization and interactive segmentation of medical images. Proceedings, Web3D 2012 - 17th International Conference on 3D Web Technology, 51–58. (2012)
9. WebGL Specification 1.0. Khronos Group. `https://www.khronos.org/registry/webgl/specs/1.0/` (2011)
10. OpenGL ES. Khronos Group. http://www.khronos.org/opengles/ (2012)
11. Hoetzlein, R.C.: Graphics performance in rich internet applications. IEEE Computer Graphics and Applications 32(5), 98–104. (2012)
12. Cabello, R.: Three.JS library: download and documentation. `http://mrdoob.github.com/three.js/` (2012)
13. Fernández, E., Jelinek, H.F.: Use of fractal theory in neuroscience: methods, advantages, and potential problems. Methods 24(4), 309–321. (2001)
14. Esteban, F.J., Sepulcre, J., Ruiz de Miras, J., Navas, J., de Mendizbal, N.V., Goñi, J., Quesada, J.M., Bejarano, B., Villoslada, P.: Fractal dimension analysis of grey matter in multiple sclerosis. Journal of Neurological Sciences 282, 67-71. (2009)
15. Liu, J.Z., Zhang, L.D., Yue, G.H.: Fractal dimension in human cerebellum measured by magnetic resonance imaging. Biophysical Journal 85, 4041–4046.(2003)
16. Zhang, L., Dean, D., Liu, J.Z., Sahgal, V., Wang, X., Yue, G.H.: Quantifying degeneration of white matter in normal aging using fractal dimension. Neurobiology of Aging 28, 1543–1555. (2007)
17. Wu, Y.T., Shyu, K.K., Chen, T.R., Guo, W.Y.: Using three-dimensional fractal dimension to analyze the complexity of fetal cortical surface from magnetic resonance images. Nonlinear Dynamics 58, 745–752. (2009)
18. Shyu, K.K., Wu, Y.T., Chen, T.R., Chen, H.Y., Hu, H.H., Guo, W.Y.: Analysis of fetal cortical complexity from MR images using 3D entropy based information fractal dimension. Nonlinear Dynamics 61, 363–372. (2010)
19. Lopes, R., Betrouni, N.: Fractal and multifractal analysis: A review. Medical Image Analysis 13, 634–649. (2009)
20. Russel, D., Hanson, J., Ott, E.: Dimension of strange attractors. Physical Review Letters 45, 1175–1178. (1980)
21. Ruiz de Miras, J., Villoslada, P., Navas, J., Esteban, F.J.: UJA-3DFD: A program to compute the 3D fractal dimension from MRI data. Computer Methods and Programs in Biomedicine 104, 452–460. (2011)
22. Aarnio, T., Bourges-Sevenier, M.: WebCL Working Draft. Khronos WebCL Working Group. `https://cvs.khronos.org/svn/repos/registry/trunk/public/webcl/spec/latest/index.html`. 2012.
23. JavaScript Web Workers. `http://www.whatwg.org/specs/web-apps/current-work/multipage/workers.html`. 2012