Real-time Collaborative Scientific WebGL Visualization with WebSocket

Charles Marion* Kitware SAS

Julien Jomier† Kitware SAS

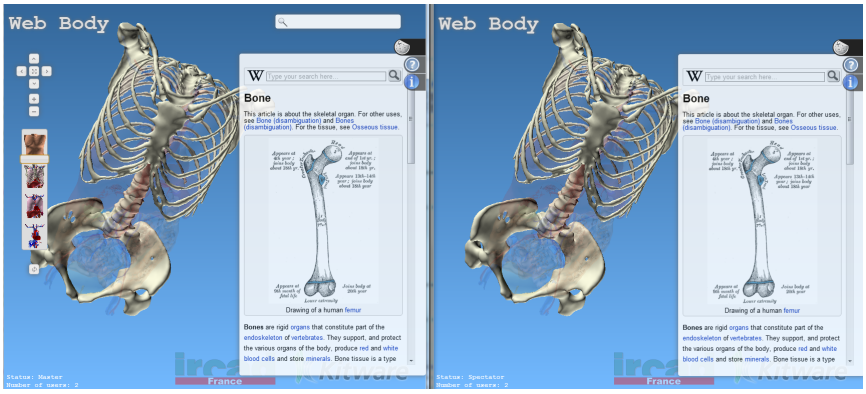

Figure 1: *Collaborative visualization integrated to the Visible Patient website. The panels represent two remote instances of the application.*

Abstract

In scientific visualization, data are becoming more and more important and usually implies a cooperative effort. Moreover experts are usually geographically distributed, therefore they need collaborative tools to work efficiently. To solve this limitation, a prospective action has been recently initiated, with the development of a web based application for collaborative interaction based on two innovating technologies: WebGL and WebSocket. To demonstrate this approach, a prototype has been developed based on the Visible Patient project. In this paper we present the architecture of the proposed system, the initial implementation experiment and a comparison with current technologies. Finally we discuss the future work and potential improvements.

CR Categories: H.3.5 [Information Storage and Retrieval]: Online Information Services—Data sharing H.3.7 [Information Storage and Retrieval]: Digital Libraries—Dissemination;

Keywords: Web 3D, remote rendering, collaborative, WebSocket, WebGL

1 Introduction

In the past decade, data visualization has become more and more common thanks to research endeavors and innovative infrastructure. In this context, several tools have been developed for scientific visualization of large datasets such as ParaView [Moreland et al. 2008] or Ensight [CEI International 2012]. However, as the datasets get larger, it becomes challenging to visualize and interact with the data locally. In the past five years, several efforts have been pushed to solve this problem such as ParaViewWeb [Jourdain et al. 2010], ShareX3D [Jourdain et al. 2008] and Collaviz Framework [Dupont et al. 2010]. Furthermore, other research projects have been focusing on interactions [Chu et al. 2006] and [Nam and Wright 2001] to develop products in a collaborative environment. However, these projects often require a complex infrastructure or the use of external plugins or program to be installed on the client side.

The system described in this paper tries to solve the limitation of the current collaborative projects by focusing on the following aspects:

- make use of client hardware to render the 3D scene and obtain optimal performances with a low latency, as opposed as the remote rendering
- create applications designed for different types of scientific visualizations: medical, architecture, biochemical, etc.
- enable interactive and participative collaboration
- allow for the optimal accessibility without the use of plugins
- rely on mainstream technologies to enable service access

Nowadays, the additional challenge is to provide also searchable, redundant and on-demand access to the data. In fact, researchers want to easily and quickly visualize the data without the need of installing complex tools and as possible from the convenience of a web browser. In order to provide a collaborative and efficient system, we propose to combine the WebGL and WebSocket technologies which do not require any complex architecture on the server side or external plugins on the client side. These technologies are now available in most of the current web browsers.

Next we present in more details the technologies used as well as the integration we have achieved for real-time online collaboration with a 3D scene.

2 Technologies

In this section, we describe the two technologies to be integrated: WebGL and WebSocket.

*e-mail:charles.marion@kitware.com

†e-mail:julien.jomier@kitware.com

Web3D 2012, Los Angeles, CA, August 4 – 5, 2012.

2.1 WebGL

WebGL (Web Graphics Library) is a JavaScript API for rendering interactive 3D graphics within any compatible web browser without the use of plugins. Based on OpenGL ES 2.0 and using HTML5 canvas, WebGL provides a way to execute shader code on a computer's Graphics Processing Unit (GPU) via a web browser. WebGL is a web standard developed by the Khronos group [Khronos Group 2011].

In WebGL, like in most real-time 3D graphics, the triangle is the basic element with which models are drawn. Therefore, the process of drawing in WebGL involves using JavaScript to generate the information that specifies where and how these triangles will be created, and how they will look (color, shades, textures, etc). This information is then fed to the GPU, which processes it, and returns a view of the scene.

The main limitations of the WebGL technology are the performance of the JavaScript engines, the dependency on the OpenGL 2.0 drivers and the early stage of the development of the developer libraries such as XTK [Harvard Medical Group] and SceneJs [Xeolabs]. However as the standard gets more and more used, we are seeing the development of very promising libraries for developers as well as powerful contributions.

2.2 WebSocket Protocol

The WebSocket specification [W3C 2012] introduced the WebSocket JavaScript interface, which defines a full-duplex single socket connection over which messages can be sent between client and server. The WebSocket standard simplifies much of the complexity around bi-directional web communication and connection management. This technology allows to develop real time synchronization between multiple users over the Internet via the web browser.

The technologies runs via port 80/443 allowing to bypass firewalls or proxies. It uses TCP handshakes which are HTTP compatible allowing the use of a cookie-based authentication. To obtain a high performance system, the message headers have been kept small and the latency has been reduced by using a single persistent connection.

The persistent connection allows the server and the client to push messages to each other at any given time. WebSocket is not limited in its nature the way that AJAX (or XHR) is; AJAX technologies require a request to be made by the client, whereas WebSocket servers and clients can push each other messages. There are many practical applications for WebSocket. WebSocket is ideal for most client-to-server asynchronous purposes, chat within the browser being the most prominent.

The WebSocket protocol is implemented in many browsers, runtime environments and libraries acting as clients or servers. Here are listed some implementations:

- The cWebsocket implementation is lightweight Websocket server library written in C.
- The pywebsocket project aims to provide a WebSocket standalone server and a WebSocket extension for Apache HTTP Server written in Python.
- The Node.js platform is built on Chrome's JavaScript runtime for easily building fast, scalable network applications.
- jWebSocket Server a pure Java based WebSocket server.

3 Concepts

The idea behind collaborative data visualization is to allow several users to share the exact same view and to efficiently communicate additional information during the visualization process. The location of the users should not be relevant. Collaborative visualization can be used as educational tools in a large variety of domains such as the surgery where a trained surgeon can demonstrate how to proceed during a chirurgical operation. A colaborative tool can also be used to cooperatively design a new product online or attend a online conference where each user interact with a virtual character, ala Second Life.

However, three main challenges arise with the integration of such a collaboration solution. The first challenge is to make the data available to anyone's visualization engine, the second challenge is to create a 3D scene information protocol to share the visualization information between the distributed users, and the last challenge is to share this information with a high frequency and a low latency.

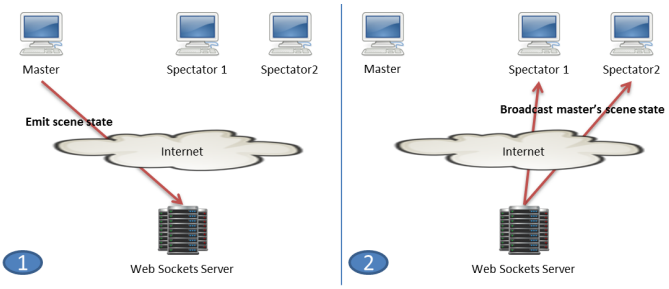

Figure 2: *Sharing of the visualization state using a Web Socket server.*

In a typical scenario, the user browses the data using a web interface and selects the dataset he wants to visualize. Then the user joins a collaborative session which triggers the visualization server to create different collaborative visualization views depending on the type of dataset. The master user is then the only one able to modify the state of the visualization. The rendering state of the 3D scene is shared in real-time to the spectators over the network. When a spectator enters the collaborative view, he receives in real time the current visualization state and upcoming updates which are used to synchronize the rendering views. The number of participants should not affect the other participants, which means that a broadcast mechanism would be suitable.

In the proposed system, to obtain a real time synchronization, only a small set of information is sent to the spectators. The 3D models are first downloaded using an HTTP request which is initiated by the collaborative visualization server. The collaboration process starts only after the data has been downloaded. The 3D engine allows us to process the scene information and reproduce the 3D scene on the client browser using the rendering pipeline described in the figure 2.

The rendering process is asynchronous of the sharing of the states allowing the rendering frame rate to be unrelated to the computer's performances of the other users. As described in the figure 3, the system is based on client-server architecture. The master sends scene information to the WebSocket server which broadcasts the same message to all the connected users. A solution using a central system simplifies the connection between the users and allows connection/disconnection management. Figure 4 shows a use case demonstrating the initial connection of a user during the collaborative visualization.

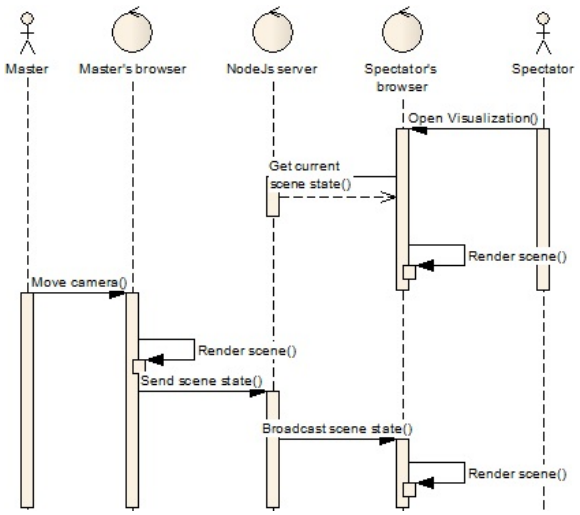

Figure 3: *Collaborative use case.*

4 Experimental setup

Based on the WebGL technology and on the lightweight open source framework, ThreeJS [ThreeJS Community 2012], we developed a system which allows users to instantly visualize 3D datasets online without the need for external plugins. The data is hosted on an open-source cloud based solution called MIDAS Platform [Jomier et al. 2010] to store the datasets and manage the 3D visualization; therefore making the use of WebGL transparent to the users.

The real-time interactive collaboration is based on Node.js [Tilkov and Vinoski 2010]. Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. The Node.js library also allows to use the WebSocket and the AJAX technologies depending on the browser of the clients. This provides a fallback solution when the protocol is not available on the web browser.

Using the javascript client capabilities, the application shares the scene information by sending a Javascript object to the WebSocket server. An example source code is shown below:

```
if( isSceneModified() )
  {
  socket = io.connect(collaborativeServerURL);
  var scene = new Object();
  scene.camera = getCameraInformation();
  scene.3DObjects = getObjectsInformation();
  socket.emit('sendSceneInformation', scene);
  }
```

The server then broadcasts the scene information to all the available clients. The following source code shows the implementation.

```
socket.on('sendSceneInformation', function (scene)
  {
  currentScene = scene;
  socket.broadcast.emit('getSceneInformation',
      scene);
  });
```

In order to provide a real world experience we decided to base our proof of concept on the Visible Patient [IRCAD and Kitware SAS 2012] project. The French Research Institute against Digestive Cancer (IRCAD) has decided to offer free access to a set of 3D virtual anatomies modeled from medical images of anonymous patients. The resulting Visible Patient website offers a complete visualization solution of 3D patient-specific anatomy. The data are freely available from the visible patient website under a creative commons license.

As a proof of concept, the experimental setup supports only a subset of collaborative features. We decided to focus our experiment on the sharing of camera movements and minimal rendering states. The test of the experimental setup was done in two steps. The first step was to evaluate the users' experience with the prototype and collect feedbacks. The second step was to benchmark the performance of the application during the experimental evaluation. To benchmark the application, we logged all the messages sent by the master user and all the messages received by the spectator. The comparison of these information allows us to derive the latency and rate of message transfer which is directly linked to the synchronization rate of the visualized scene.

In order to test our system in real world condition we installed a server in New York (USA) and the users were located in France.

5 Results

The overall user's feedbacks were good. Most of the users were surprised by the reactivity and the fluidity of the system and how easy it was to initiate a collaborative visualization session. As shown on the Figure 1, the panel on the left is the view of the master user and the panel on the right is the corresponding view for the spectators. In the ideal case they should always be synchronized.

From the different benchmarking sessions, we compiled three measures, the average latency, the synchronization rate and the master rendering rate. The average latency is the average end-to-end delay of a message transmission between two clients (via the Node.js server). The synchronization rate is the number of queries received per second by one client. The master rendering rate is the number of time the application renders the 3D scene per second. These measures are reported in the table Figure 5.

	AJAX	WebSocket
Average latency	332.4ms	149.5ms
Synchronization rate (per second)	5.89	59.1
Master rendering rate (per second)	60	60

Figure 4: *Benchmarking results.*

Next we discuss these results and present a conclusion.

6 Discussion

The results presented in this paper demonstrate that the proposed system is easy to use and provides good overall performance leading to an optimal user experience.

The benchmark demonstrated the large gain in term of networking performance using the WebSocket protocol compared to AJAX. We also tested internally the maximum number of messages received per second and we managed to easily get more than a thousand messages per second. This means that the prototype synchronization rate was only limited by the rendering rate of the master client.

Furthermore, the system can be used in a large variety of use cases but we recognized that it has two main limitations. First, the rendering of the 3D scene depends directly on the performance of the client and the proposed system is not suitable for the visualization of large datasets since it would require downloading the entire

model's geometry. The second limitation is related to the implementation of the system which currently does not include any advanced post processing visualization such as clipping, decimation, etc.

The solution presented in the paper implements an easy to use collaborative system which uses browsers' embedded technologies but would require a hybrid solution to support all the types of visualizations, mainly large datasets requiring intensive pre/post processing computation for visualization. This hybrid system would be based on the combined use of remote and local rendering.

In the context of this project, we confirmed that providing high performance visualization is essential but an easy to use tool is almost as important. The system must also provide an easy way to select the datasets of interest and to start the collaborative visualization.

7 Conclusion

In this paper we have presented a novel approach to collaborative visualization by combining two new technologies: WebGL and WebSocket, therefore allowing for a faster and easier way to remotely collaborate with 3D datasets.

In the future, the protocol used in the experimental setup to describe the scene could be improved to share more information regarding the scene, such as material, texture, lighting and other interactive features. We also plan to use this technology to integrate a collaborative visualization on mobile devices, not necessarily using WebGL. For instance we could synchronize a WebGL visualization with a native visualization on an iPad using embedded technologies such as the VES library (VTK OpenGL ES Rendering Toolkit) [Kitware 2012].

Acknowledgements

The authors would like to thank the Research Institute against Digestive Cancer (IRCAD) for providing the data used for this experiment.

References

CEI INTERNATIONAL, 2012. Ensight gold software. http://www.ensight.com/.

CHU, C.-H., CHENG, C.-Y., AND WU, C.-W. 2006. Applications of the web-based collaborative visualization in distributed product development. *Computers in Industry 57*, 3, 272 – 282. ¡ce:title¿Advanced Computer Support of Engineering and Service Processes of Virtual Enterprises¡/ce:title¿ ¡ce:subtitle¿Advanced Computer Support Special Issue¡/ce:subtitle¿.

DUPONT, F., DUVAL, T., FLEURY, C., FOREST, J., GOURANTON, V., LANDO, P., LAURENT, T., LAVOUÉ, G., AND SCHMUTZ, A. 2010. Collaborative Scientific Visualization: The COLLAVIZ Framework. In *JVRC 2010 (2010 Joint Virtual Reality Conference of EuroVR - EGVE - VEC)*.

HARVARD MEDICAL GROUP. The x toolkit: Webgl for scientific visualization. https://github.com/xtk/X.

IRCAD, AND KITWARE SAS, 2012. Visible patient: Freeware for 3d visualization of 3d models of patients. http://www.visiblepatient.eu/, March.

JOMIER, J., BAILLY, A., GALL, M. L., AND AVILA, R. 2010. An open-source digital archiving system for medical and scientific research.

JOURDAIN, S., FOREST, J., MOUTON, C., MALLET, L., AND CHABRIDON, S. 2008. Sharex3d, a scientific collaborative 3d viewer over http.

JOURDAIN, S., AYACHIT, U., AND GEVECI, B., 2010. Paraviewweb, a web framework for 3d visualization and data processing, 07.

KHRONOS GROUP, 2011. Webgl specification. https://www.khronos.org/registry/webgl/specs/1.0/, February.

KITWARE, 2012. Kiwiviewer. http://www.kiwiviewer.org/.

MORELAND, K., ROGERS, D., GREENFIELD, J., GEVECI, B., MARION, P., NEUNDORF, A., AND ESCHENBERG, K. 2008. Large scale visualization on the cray xt3 using paraview. *Cray User Group 2008 2* (01).

NAM, T.-J., AND WRIGHT, D. 2001. The development and evaluation of syco3d: a real-time collaborative 3d cad system. *Design Studies 22*, 6, 557 – 582.

THREEJS COMMUNITY, 2012. Threejs: Lightweight javascript 3d library. http://threejs.org.

TILKOV, S., AND VINOSKI, S. 2010. Node.js: Using javascript to build high-performance network programs.

W3C, 2012. Websocket api specification. http://dev.w3.org/html5/websockets/, March.

XEOLABS. 3d scene graph engine for webgl. http://www.scenejs.com/.