International Journal of Computer Information Systems and Industrial Management Applications.
ISSN 2150-7988 Volume 3 (2011), pp. 870–877

ParaViewWeb: A Web Framework for 3D Visualization and Data Processing

Sebastien Jourdain, Utkarsh Ayachit and Berk Geveci
sebastien.jourdain@kitware.com, *utkarsh.ayachit@kitware.com*, *berk.geveci@kitware.com*

Kitware Inc.
28 Corporate Drive, Clifton Park, NY 12065, USA

Abstract: **Since the early days of the Internet, web technologies have been at the forefront of innovation with many traditional desktop applications migrating to the web and new ones continually being invented. In this paper, we present a 3D visualization framework, ParaViewWeb, which enables interactive large data processing and visualization over the web. To enable large data processing, ParaViewWeb uses ParaView, an open-source, parallel data visualization framework, to generate visualizations on the server-side while rapidly delivering images to the client. ParaViewWeb makes it easier to develop customized applications for the web that cater to a wide variety of scientists and domain experts who can use such a web-based system to analyze their datasets without leaving the familiar and omnipresent environment of a web browser. In this paper, we present the ParaViewWeb framework and its features and discuss some of the application fields that can benefit from it.**

Keywords: Web3D, Scientific visualization, Collaboration, Remote rendering, VTK, ParaView

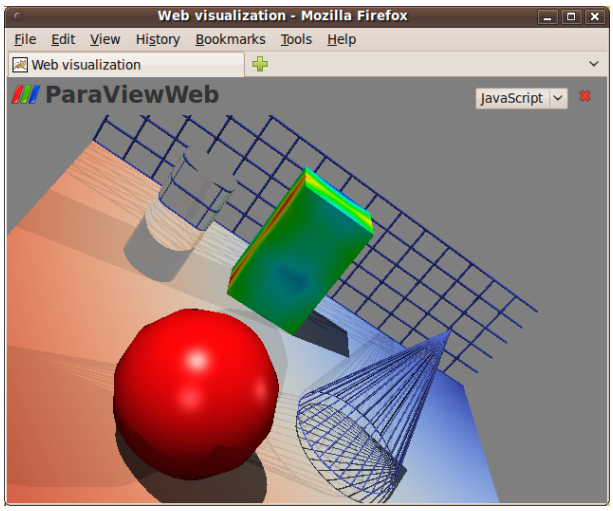

Figure. 1: Ray-casting web application

I. Introduction

In recent years, organizations across the globe have increased development efforts on high performance computing infrastructures. These are often distributed across several sites. This enhanced compute power has made it possible to run large simulations, producing large datasets. The data sizes have made distance visualization a necessity; it is no longer possible to copy datasets to your desktop for analysis. ParaView [1] is an open-source, multi-platform data analysis and visualization application. ParaView was developed to analyze extremely large datasets using distributed memory computing resources. It can be run on supercomputers to analyze datasets of terascale as well as on laptops for smaller data. In this paper, we present an exciting new approach for visualizing data over the web. We present a ParaView-based web-visualization framework that allows developers to build custom web applications that incorporate interactive 3D visualization. These web applications can leverage ParaView's parallel data processing and rendering capabilities on the server side, while presenting easy-to-use, highly customized webpages to the users to control and interact with their visualizations; all without leaving the familiar and omnipresent environment of a web browser.

II. Objectives

The web has been evolving rapidly since its inception. It has permeated the modern way of life so much that people are now migrating traditional desktop applications to the web, e.g. document editing [2], finance management [3] and tax filing [4]. Part of the appeal of the web is its ease of access. One simply types the URL in the web browser and can access the application, regardless of location or type of device. Additionally, web makes it easier to share and collaborate with other users no matter where they are located. The same is true for visualization over the web. There are several frameworks that have been developed that focus on visualization, e.g. Protovis [5] and Many Eyes [6]. However, most of these are dealing with 2D visualization and use client-side rendering to generate the visualizations, which works well for smaller data sizes but is not feasible with large datasets.
Our main objective is to leverage that growing usage and be able to easily access HPC resources and allow cross domain expertise on large scale simulation projects where expert teams may be scattered across the globe.
Moreover ParaViewWeb aims to provide a set of web-oriented components to allow web designers to easily develop their own ad-hoc visualization applications for the web or extend their existing portals or monitoring tools with inter-

active 3D data analysis capabilities. The Figure 1 illustrates a simple 3D web application that use the ray-tracer rendering engine of ParaView through the ParaViewWeb framework.
The rest of this paper is organized as follows. In the Section 3, we give a general overview of the system and the different components involved. Section 4 covers the different components of the framework in detail, while in Section 5 we will demonstrate the use of the framework in different applications.

III. Architecture Overview

To address the need for visualizing large datasets in 3D, we have developed a framework based on ParaView. We can leverage all of ParaView's visualization and data processing features such as parallel processing and rendering, volume rendering and ray-casting. To provide interactive 3D content in real time to the client, we use server-side rendering. This approach makes it possible to render large geometries without putting any requirement on the client's devices and enables the use of mobile devices such as iPhone and iPad. To manage the 3D rendered content on the client side, the framework provides a set of components for viewing it in an interactive manner. Those components are available in Java [7], Flash [8] and JavaScript [10] and can be used to rotate, zoom or pan the 3D data.
As a framework, ParaViewWeb uses proven technologies and standards such as Java Servlet, Java Server Pages (JSP) [12], Java Messaging Service (JMS) [11], and Java Persistence API (JPA) [13] on the server side to do most of the binding between the browser and the ParaView framework. We use JavaScript and JSON-RPC [14] for communication protocol on the client side. Being a web-based application, all communication follows either HTTP or HTTPS protocols which make it easier to manage firewall and proxy issues. ParaViewWeb provides a collection of components that can be used directly or easily integrated into an existing web portals. These components range from server side to client side. Figure 2 gives an overview of the complete system.
On the server side, the visualization server (PWServer) is a ParaView-based engine that does the actual visualization either by itself or by connecting to a remote ParaView server running over a cluster with MPI. Then, the web service component named PWService manages communication between remote visualization servers (PWServer) and clients. It also includes administration webpages, allowing a user to monitor running visualization session as well as browsing logs and information of the previous ones. On the client side, a JavaScript library is provided for creating remote visualizations and managing them. It also contains several visualization components enabling users to look at 3D content in the browser interactively.
Using these components, developers can build websites or web portals with visualization and data processing capabilities. These components can be easily integrated into Rich Internet Applications (RIAs) developed using popular web design infrastructures including qooxdoo [15], Dojo [9], Google Web Toolkit [16], jQuery [17], Flex [8], and Java [7]. Figure 2 gives a schematic of the various components involved. Our implementation requires any supporting Java-based Web Application server, such as Apache Tomcat, an open-source, freely available implementation.
In the next section we will see each of these components in detail.

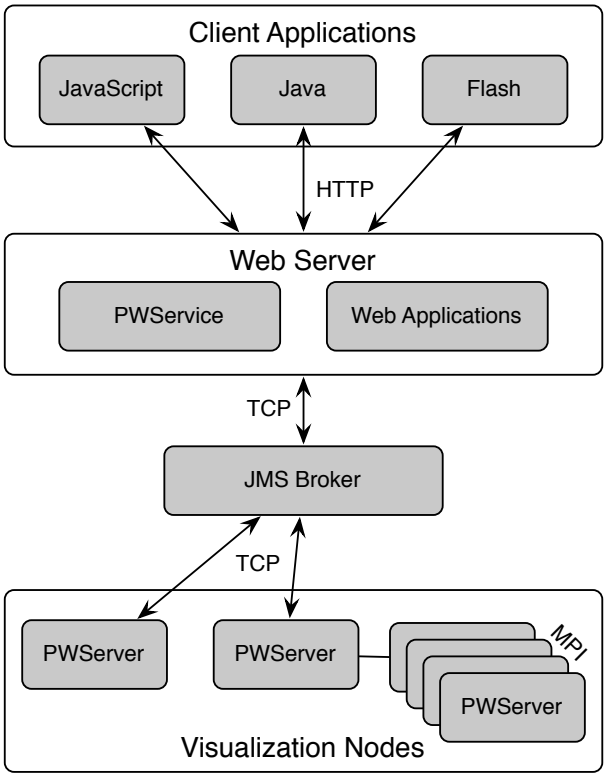

Figure. 2: Schematic of the ParaViewWeb Visualization System

IV. Components

As described earlier, there are 3 major components in the ParaViewWeb architecture.

- Web Service provides the facade exposed to the external network including the web browsers. It enables accessing visualization as a service over the network.
- JavaScript Visualization API is used by web-browser to create and update visualization pipelines. Internally, it uses the services provided by the Web Service to perform the requested tasks.
- Visualization Engine is a ParaView application that runs on the visualization nodes to do the actual data analysis and processing.

A. Web Service

The crux of a typical web framework is its collection of web services. The web services provide the facade that is accessible by the client applications including the webpages loaded in web browsers. The collection of services provided by ParaViewWeb framework are called PWService. PWService is a web application that is deployed on the web server accessed using a particular URL that is determined at configuration time. Using JSON-RPC [14], which is a simple JSON [18] (JavaScript Object Notation) based protocol for remote procedure calls, clients can make calls on

PWService to start new instances of PWServer, monitor running PWServer instances, and even send JSON messages to PWServer. The messages sent to PWServer range from those constructing the visualization pipeline to those that communicate the mouse interactions. PWService is responsible for delivering responses from the visualization engine (PWServer) to the web clients.

The PWService also makes it possible for the clients to request rendered images from the PWServer. To address the concerns of highly secure domains, we do not expose any additional ports through which the clients can communicate with the PWServer. Hence, their only access is via the PWService. This provides an interesting challenge especially when providing rendered images as the latency becomes critical. Any overhead inherent in the web service or the HTTP protocol itself can adversely affect the perceived framerate on the client side. HTTP protocol [19] in itself does not support the server sending data to the client, unless the client requested it. This implies that the client has to periodically check with the server if a new rendered image is available and then fetch it. Also every time there are new requests, a handshake as per the protocol ensues, which can affect the latency. To address these issues a few techniques are gaining widespread acceptance in the community under the umbrella term of server push [20] techniques. Each of rendering components uses a different technique to get the best performance. The long-polling [21] (for JavaScript rendering) and streaming (for Java rendering) techniques are the two mechanisms currently implemented inside ParaViewWeb. ParaViewWeb uses Abodes BlazeDS [22] remoting technology for the Flash renderer which also employes similar techniques under the covers. Next we'll see the techniques employed by each of the renderers implementation.

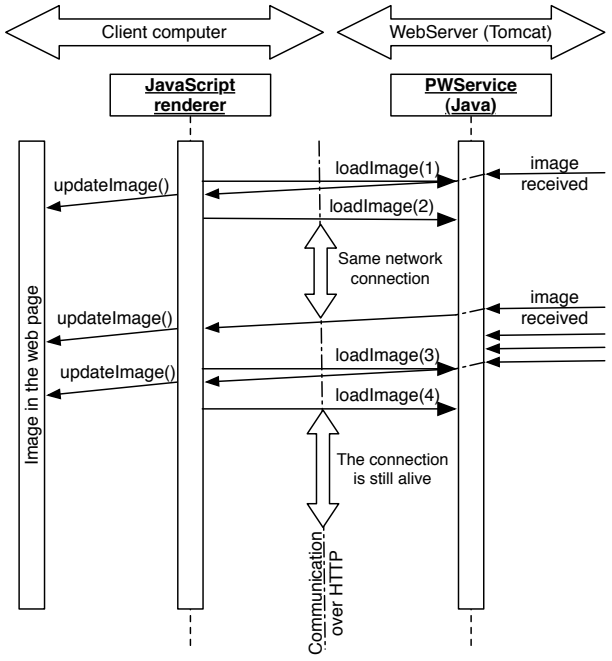

Figure. 3: Image delivery with the long-polling technique for JavaScript renderer

1) Image delivery for JavaScript renderer

The standard polling technique allows the client to fetch server data periodically with some overhead and latency. In order to reduce that latency and to allow JavaScript clients to fetch binary content such as images, we implemented a long-polling communication style, which is also known as Comet programming [23]. Instead of using this technique for small payload messages, we used it for image delivery in the JavaScript renderer. Figure 3 illustrates the communication between the JavaScript renderer of ParaViewWeb through the web service and the ParaView rendering engine.

Moreover, that technique has an interesting side effect for heterogeneous clients: the image delivery is driven by each individual client. Therefore, the ParaViewWeb service has been designed to deliver only the latest image available to the client and keep the connection open if nothing new is available since the last data transfer. In this manner, each client gets the best possible framerate without affecting other clients. For example, if the server is able to produce 80 frames per second for a given visualization, some clients may get only 30 frames per second compared to others who would receive 60 frames per second.

2) Image delivery for Java Applet renderer

Java Applet [7] is not the most supported platform on the web but compared to JavaScript or Flash, it provides more advanced capability. In order to provide the most efficient environment for user interactivity, we developed a renderer for ParaViewWeb written in Java that could be used as an embedded applet inside a webpage or any other standard Java application. In order to improve the communication between this component and the server side we used the multi-threading capability of the environment to better separate the image management from the user interaction. Moreover, as Java has some built-in tools for stream management, we used custom protocol on top of the HTTP one. For the mouse interaction, the web service allows the clients to use an up-stream persistent network connection where interaction events are written using an XML structure. This allows any client to use a single up-stream channel for the user interaction and by doing so prevents the client from creating new connections while the user is interacting. For the image stream, the web service is producing a single zip stream where the images are pushed one after the other as a separate entry in the stream. Using a zip stream is also another standard way to communicate with a client a stack of binary content. This allows any other heavy client to benefit from that existing communication channel.

The up-stream channel is only established at the beginning of the user interaction and gets released when the user stop its interaction. Conversely, the down-stream connection stays live for the entire session.

By using a separate up-stream and down-stream communication channel we almost mimic a standard socket communication. But even with that communication capability, if we do not provide any extra care in the image management, we could get a poor interactivity result with that design. In fact, if all the images produced on the server side are sent through the web, we might simply overflow the bandwidth and induce an increasing lag between the current user interaction and the image that is currently rendered locally. So depending on the

network capability of the client connection, if we do not drop images we will create an increasing image stack that will still be playing after the user has stopped its interaction. Therefore, in the same way as we did for the image delivery for the JavaScript client, we allow the service to skip images on the server side. As the client and the server remain connected all the time, it is unnecessary to write all the images on the communication socket. Therefore what we are doing is only writing or waiting for the latest image available after each image has been fully written on the stream. That technique prevents this buffering lag and reduces the bandwidth usage.

3) Image delivery for Flash/Flex renderer

Flash [8] is a commonly used technology on the web to deliver rich and interactive content. It was the first choice for ParaViewWeb for its rendering component since all the infrastructure existed and is widely used. BlazeDS [22] is the technology developed by Adobe [24] to support asynchronous messages between client and server communication and specifically in a server push [20] paradigm. Based on those technologies the ParaViewWeb Flash rendering component has been designed to exchange both mouse interactions and server images. Unfortunately, we did not implement any image dropping technique along the image delivery chain which induces some lag with low connection bandwidth during interaction.

4) Future Extensions

Web standards are rapidly evolving and new ones will become available with the new HTML 5 [25] standards. ParaViewWeb could benefit from several of these either for image delivery or client side rendering. The communication mechanism in the JavaScript layer could be even more efficient than the current Java one by using WebSocket [26]. On the other hand, WebGL [26] could allow the browser to directly render small geometry models without heavy communication payload.
Those standards are making their way into the current web browsers and some early implementations are available in advanced browser such as Chrome or Safari. Some experimentation has been done using those new standards [26] to render a 3D scene that has been generated by ParaView into a web browser using the WebGL framework. This work could easily be used to extend ParaViewWeb to support such client-side rendering mechanisms.

B. Visualization Engine

The actual data analysis and visualization in the ParaViewWeb framework is done by the visualization engine, called PWServer. PWServer is a ParaView-based process that can be thought of as a headless ParaView application that responds to the JSON messages relayed via the PWService. In typical configurations, the web-server and the visualizations nodes are separate machines. This means that the PWService has to use network communication to relay the messages to PWServer. In our design, we use Java Message Service (JMS) for this communication. JMS is an API for sending messages between two or more clients which was proposed as part of the Java Platform. There are several proprietary as well as open source implementations available for the both the relaying server (known as the JMS Broker) as well as the message sending and receiving libraries. The protocol employed is still JSON-RPC-based even though the communication is over JMS.
PWServer is the visualization engine. It can be set up to use MPI to perform parallel data processing and rendering over a cluster or as a single process and it does all the data processing and rendering. For each separate visualization session, we created a separate PWServer process. PWService handles the management of these instances of PWServer, whether they are local or working on a remote cluster and involving MPI internal communication.
To ensure that the client application perceives the best possible framerate when interacting, the PWServer employs a couple of optimizations.

1) Asynchronous Processing

As most users would expect in remote rendering, every user interaction that affects the 3D scene should trigger the rendering of an image which should be immediately shipped to the clients. However, there undoubtedly is some delay between the client requesting an image and the server delivering one. If the server continuously processes the requests received from the client while the user interacts, the user is bound to see a lag in the movement. To overcome the issue, the client send asynchronous requests to the server. The server receives the first requests and start rendering. While the rendering is happening, it keeps on receiving any additional interaction updates in a separate thread, from the client and combines them into a single update. Once the rendering is over the image is dispatched and the server starts processing the most recent update.

2) Image Resolution/Quality Tweaks

Another trick to improve the perceived frame rate on the client side is interactively adapt the image resolution and/or quality during interaction. As one interacts on the client side, the image resolution and/or quality is reduced. Once the interaction is stopped, the server sends the optimal quality and resolution image.
In our implementation, we have chosen to reduce the image quality via two parameters. The first one is the image size and the second one is the JPEG [29] compression option. By reducing each dimension by two and by setting the quality to 50 for the JPEG compression, we can produce images that could be 70% smaller than full size.

C. The JavaScript library

JavaScript has become the language of choice for developing web applications for the browser. ParaViewWeb provides a JavaScript API that can be used within web browsers to access the PWService as well as communicate with the PWServer. The JavaScript library provides remoting capabilities, enabling the web browser to access remote visualization objects present on the PWServer.
ParaView is a post-processing application that can either work on laptops or larger distributed computers [30]. ParaView is using the VTK [31] library for its data processing

but the client side code never directly manipulates those VTK objects. Instead the client is using proxies to abstract the communication and the deployment of those VTK objects. Thanks to that design, ParaView is using the same code on the client side to work either in a standalone manner or in a distributed mode across several nodes of a super computer.
On the client side, ParaView uses proxies to abstract the management of objects on the server side. The ParaViewWeb JavaScript library allows the user to create and manipulate the ParaView proxies directly inside JavaScript code. This facilitates the definition and the configuration of visualization pipelines. Figure 4 illustrates how proxies can be created and how we can retrieve and set their properties in JavaScript. Alternatively, Figure 5 illustrates how a full integration of ParaViewWeb can be achieved inside a static webpage, which embeds a 3D interactive renderer.

```
/* Let's start by creating a Cone object: */
var cone = paraview.Cone();

/* Get the full list of properties. */
var propertiesArray = cone.ListProperties();

/* Get the resolution of the cone */
var resolution = cone.getResolution();

/* Double the resolution of the cone */
cone.setResolution(resolution * 2);

/* Creating another cone with a resolution of 64 */
var cone2 = paraview.Cone( { Resolution:64 } );

/* Assign an array value */
cone2.setCenter(4, 5, 6);

/* Apply a shrink filter and show the result */
var shrinkFilter = paraview.Shrink( {Input:cone} );

/* Getting data information of the shrink filter */
var dataInformation =
    paraview.GetDataInformation({proxy:shrinkFilter});

var numberOfCells = dataInformation.NbCells;
var numberOfPoints = dataInformation.NbPoints;
var bounds = dataInformation.Bounds;
var memoryUsage = dataInformation.Memory;
var dataType = dataInformation.Type;

/* Show the result of the shrink filter */
paraview.Show( {proxy:shrinkFilter} );

/* Manage camera and view */
paraview.ResetCamera();
var view = paraview.CreateIfNeededRenderView();
view.setCenterOfRotation(view.getCameraFocalPoint());
```

Figure. 4: Sample JavaScript pipeline setting

By providing the full access to the Proxy Manager of ParaView, we allow web developers to design any kind of application, which could range from a very simple application like the one illustrated in Figure 5 and 6 to a more complex one that could mimic the current Qt one. On the other hand, developing ad-hoc applications that target a very specific community on web could be a valuable endeavor. Moreover, it might be easier to develop several simple applications and have a dedicated URL for each of them.

1) Plugins for server side API extension

Managing Proxies with the JavaScript layer may result in a lot of communication overhead where you only want a default setting to be applied on a large number of proxy objects. To overcome this limitation we designed a server-side plugin

```
<html>
  <head>
    <meta http-equiv="content-type"
          content="text/html; charset=ISO-8859-1">
    <script src="seb_files/ParaViewWeb.js"
            type="text/javascript"></script>
  </head>
  <body onunload="paraview.disconnect();">

    <div id="renderer">
       <!-- Renderer will be inserted here-->
    </div>

    <script type="text/javascript">
      var url = "http://paraviewweb.kitware.com/PWService";

       // Create a paraview session
      var paraview = new Paraview(url);
      paraview.createSession("name", "comment");
      var view = paraview.CreateIfNeededRenderView();
      var viewId = view.__selfid__;

      // Load 3D file
      var cow = paraview.OpenDataFile(
                      {filename:'/server-path/cow.vtp'});
      // Cut data
      var normal = [1,2,10];
      var origin = [0,0,0];

      // Red side of the cow
      var clipRed = paraview.Clip({Input:cow});
      var planRed = clipRed.getClipType();
      planRed.setNormal(normal);
      planRed.setOrigin(origin);
      var repClipRed = paraview.Show({proxy:clipRed});
      repClipRed.setDiffuseColor(1,0,0);

      // Blue side of the cow
      var clipBlue = paraview.Clip({Input:cow,InsideOut:1});
      var planBlue = clipBlue.getClipType();
      planBlue.setNormal(normal);
      planBlue.setOrigin(origin);
      var repClipBlue = paraview.Show({proxy:clipBlue});
      repClipBlue.setDiffuseColor(0,0,1);

      // Make sure we update the view once
      paraview.Render();

      // Add a green bell to the cow
      cone = paraview.Cone({Center:[4.5,0.25,0],
                      Direction:[0,1,0], Resolution:16 });
      var repCone = paraview.Show({proxy:cone});
      repCone.setDiffuseColor(0,1,0);

      // Create and bind 3D interactive renderer
      var renderer = new JavaScriptRenderer("0",url);
      renderer.init(paraview.sessionId, viewId);
      renderer.setSize('400','400');
      renderer.bindToElementId("renderer");
      renderer.start();
    </script>
  </body>
</html>
```

Figure. 5: Sample HTML & JavaScript code

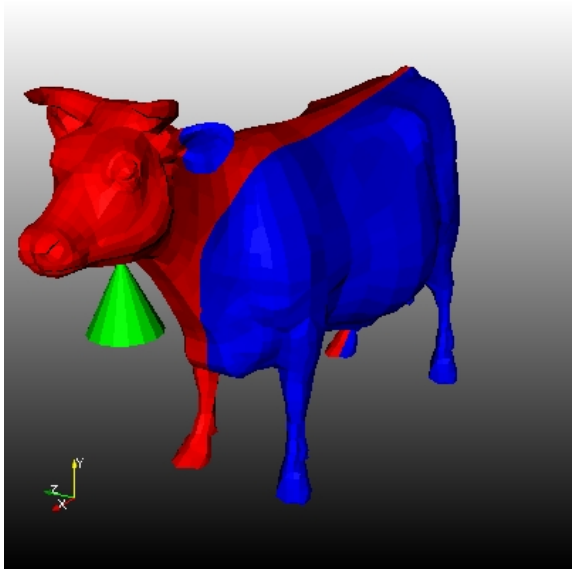

Figure. 6: View of the HTML & JavaScript code

mechanism that allows the user to call any custom remote methods like they could do with proxies.
Those plugins are python module that lie on the server side and get wrapped into a JavaScript object at runtime. Due to that extension capability, the client code written in JavaScript can use a simplified and dedicated API to implement complex behavior that could be painful to do on the client side. However, this also allows the web developer to build smaller and simpler JavaScript code where the most complex code stays and gets executed on the server-side with no networking overhead.

V. Applications

Web visualization has several real world applications. As mentioned earlier, it facilitates development of web portals for job submission and then result analysis using ParaViewWeb for supercomputing sites. These visualization applications can be customized to the types of datasets that are being visualized, making them easier to use for the domain experts.
In order to help web designers benefit from this library, several basic applications have been developed to either illustrate the capability of the framework or to provide some tutorial code on how the framework works and how it can be used through several contexts and third-party JavaScript libraries.

A. *Post Processing*

As a demonstration prototype, Figure 7 illustrates a web application that has been developed using the framework. That application has been designed to illustrate how ParaView can be used as a backend for relatively complex 3D data processing on the web. The Google GWT [16] JavaScript toolkit has been used to build this application. The purpose of that application is to illustrate what can be done on the web in terms of interactivity with data processing involved. It has not been built as a ParaView replacement, but as an example of what can be done with the ParaViewWeb framework.

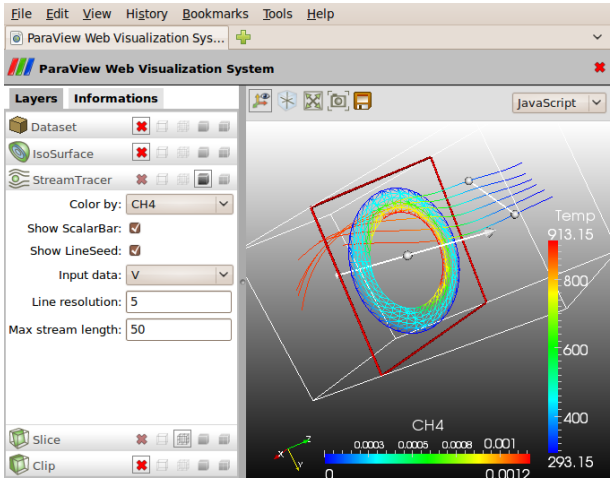

Figure. 7: Simple post processing Web application

In regards to data, the user can either select a dataset on the server or upload their own. The supported data types are the one supported by ParaView which range from simple geometry one such as VRML and PLY to real simulation dataset using VTK, Exodus, Xdmf or other formats. Once the data is loaded on the server, it can be processed by a set of classical scientific visualization filters such as clipping plane, slicer, iso-contours and, depending on the data, a stream tracer which is used to see the flow based on the integration of a vector field. Moreover, interactive 3D objects can be used to move or rotate the plan used by the slicer and the clipping plane filter as well as the line seed where streaming lines start their computation.

B. *Knowledge Sharing*

Web visualization also presents exciting new possibilities for modern classrooms. Students no longer have to look at static images in a textbook. They can study the fluid flow properties, for example, by interacting with a visualization online. Similarly, with online journals, instead of merely presenting static images in papers, we can develop online journals and publication establishments that enable the reader to simply click on the image to start interacting with the real dataset.

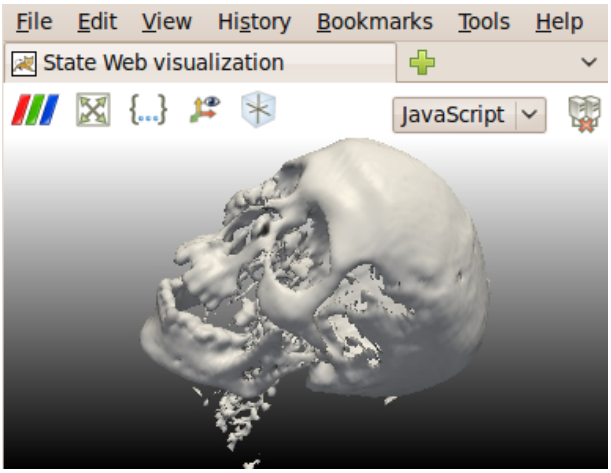

Figure. 8: State based generated Web application

In fact, an advanced web application has been developed along the framework to allow users to build a standalone web application based on a ParaView state file. That state file is used to load and set up the full processing pipeline. Moreover, if that state file has been extended with some additional information, a control panel can be shown to enable users to configure at runtime some filters parameters. Figure 8 illustrates a running state application where the user can interact with the 3D scene and, by enabling the control panel on the left, change the iso-contour value of that scanned mummy skull with a slider.

C. *Online Data Publishing*

ParaViewWeb can also be used as a rendering engine for online 3D databases. Through ParaViewWeb, users will be able to browse datasets and visualize them directly inside their browser. In fact, some first integration steps have been done with MIDAS [32], a multimedia server for storing massive collections of scientific data where users can now explore their data interactively.

D. Visualization on Mobile Devices

Since most modern mobile devices come with built-in web-browsers, its very easy to develop web apps accessible from devices such as iPhone and iPad. The JavaScript renderer supports multi-touch capability of the devices and allows the user to zoom with the appropriate gesture. Also since the web browser using simple JSON based protocol to communicate with the web service, its possible to create custom Apps for the mobile platform using the native SDK.
To this end, we developed a prototype running on Android^{TM} [33] phones and tablets that use the JSON-RPC protocol to set up the visualization pipeline and use the advanced streamed image delivery channel that the Java applet is using to reduce the communication overhead.

VI. Conclusions

In this paper, we have presented ParaViewWeb, a framework for large data visualization over the web. ParaViewWeb is a parallel visualization framework that is comprised of components that make it easier to develop websites that enable interactive analysis of large datasets. It is based on ParaView, a popular open-source visualization tool which uses distributed data processing and rendering. ParaViewWeb uses server-side rendering to avoid complications with delivering large geometries to the client and complications with rendering 3D geometries in a web-browser in a cross-browser and cross-platform manner. To provide a responsive 3D visualization system, we optimized image processing and delivery to reduce latency.
ParaViewWeb is not just a visualization framework; it is a web extension of ParaView. This allows any web developer to build applications inside a web page that leverage the computation and rendering capability of ParaView. Although ParaViewWeb has a focus on remote rendering for large data visualization, some of its first usage has been done on small data models where WebGL technology could have been used. However, this evolution can still be achieved once WebGL is mature enough and if the community feels the need for it.
Like ParaView, ParaViewWeb will be released under a BSD license, enabling anyone to extend and customize it to their specific needs.

Acknowledgments

Portions of this work were supported by DOE SBIR Phase II Award DE-FG02-08ER85143.

References

[1] Henderson A. et al, 2007. ParaView Guide. A Parallel Visualization Application. Kitware Inc.

[2] Google Docs, 2010. http://docs.google.com

[3] Mint Software, 2007. Mint, Inc. http://www.mint.com

[4] TurboTax, 1997. Intuit, Inc. http://www.turbotax.com

[5] Bostock M., Heer J., 2009. Protivis: A graphical toolkit for visualization. http://vis.stanford.edu/protovis

[6] Many Eyes, 2004. http://manyeyes.alphaworks.ibm.com/manyeyes/

[7] Sun, 1995. Java-Oracle. http://java.sun.com

[8] Adobe Flex, 2006. Adobe. http://www.adobe.com/products/flex

[9] Dojo, 2009. Dojo foundation. http://www.dojotoolkit.org/

[10] Ecma, 1999. Standard ECMA-262, ECMAScript Language Specification, 3rd edition

[11] JSR-914, 2002. Java Message Service. http://jcp.org/en/jsr/summary?id=914

[12] JSR-245, 2006. JavaServer^{TM} Pages. http://jcp.org/en/jsr/summary?id=245

[13] JSR-220, 2006. Enterprise JavaBeans 3.0. http://jcp.org/en/jsr/summary?id=220

[14] JSON-RPC, 2006. http://json-rpc.org

[15] Qooxdoo, 2005. 1&1 Internet AG. http://qooxdoo.org

[16] GWT, 2007. Google Web Toolkit. http://code.google.com/webtoolkit

[17] jQuery, 2009. Software Freedom Conservancy. http://jquery.com

[18] JSON, 1999. http://json.org

[19] Hypertext Transfer Protocol, 1990, http://www.w3.org/Protocols/rfc2616/rfc2616.html

[20] Server Push. http://en.wikipedia.org/wiki/Push_technology

[21] Jourdain Sebastien, Forest Julien, Mouton Christophe, Mallet Laurent, Chabridon Sophie, *ShareX3D, a scientific collaborative 3D viewer over HTTP*, Web 3D symposium, 2008.

[22] Blazeds, 2008. http://opensource.adobe.com/wiki/display/blazeds

[23] Comet, 2006. http://en.wikipedia.org/wiki/Comet_(programming)

[24] Adobe, 2010. http://opensource.adobe.com/wiki/display/site/Home

[25] HTML 5, 2011. http://dev.w3.org/html5/spec/

[26] WebSocket, 2011. http://dev.w3.org/html5/websockets/

[27] WebGL, 2011. http://www.khronos.org/registry/webgl/specs/latest/

[28] x3dom sample, http://www.x3dom.org/?p=821

[29] JPEG, http://en.wikipedia.org/wiki/JPEG

[30] SANDIA Report, September 2010.
http://www.cs.unm.edu/ kmorel/documents/MilestoneFY10Sandia.pdf

[31] The Visualization Toolkit. http://www.vtk.org/

[32] Midas, 2009. Kitware Inc.
http://www.kitware.com/products/midas.html

[33] AndroidTM, http://www.android.com/

Author Biographies

Sebastien Jourdain received a Master's Degree in Computer Science, from the ESSTIN Engineering School (France) in 2002. Since then he worked in software development involving high performance, 3D scientific visualization and collaboration. In February 2010, he joined Kitware where he is currently working as Research and Development engineer on ParaView and its collaborative aspects.

Utkarsh Ayachit Mr. Ayachit is a technical lead at Kitware Inc. He is one of the leading developers of the ParaView visualization aplication and leads the ParaViewWeb development team. Mr. Ayachit received his Master's Degree in Compture Science from Univerity of Maryland, Baltimore County in 2004. His interests include large scale parallel data analysis and visualization, collaborative remote visualization and application frameworks.

Berk Geveci Dr. Geveci leads the scientific visualization and infomatics teams at Kitware Inc. He is one of the leading developers of the ParaView visualization application and the Visualization Toolkit (VTK). His research interests include large scale parallel computing, computational dynamics, finite elements and visualization algorithms. Dr. Geveci regularly publishes and teaches courses at conferences including IEEE Visualization and Supercomputing conferences.
Dr. Geveci received his B.S. in Mechanical Engineering from Bogazici University in 1994, his M.S. and Ph.D. in Mechanical Engineering from Lehigh University in 1996 and 1999, respectively. While at Lehigh University he conducted research on subsonic and supersonic flow induced nonlinear vibrations, developing a new procedure for the solution of coupled flow and structural equations. In addition, he authored software for the study of separation in unsteady boundary layer flows and the visualization of the numerical and experimental results. After graduating from Lehigh University, Dr. Geveci completed a post-doctoral fellowship at the University of Pennsylvania during which he worked in the area of optimal control investigating applications in the control of hydrothermal instabilities.