ShareX3D, a scientific collaborative 3D viewer over HTTP

Sébastien Jourdain, Julien Forest, Christophe Mouton, Bernard Nouailhas, Gérard Moniot, Franck Kolb, Sophie Chabridon, Michel Simatic, Zied Abid, Laurent Mallet

▶ **To cite this version:**

Sébastien Jourdain, Julien Forest, Christophe Mouton, Bernard Nouailhas, Gérard Moniot, et al.. ShareX3D, a scientific collaborative 3D viewer over HTTP. WEB3D 2008 : 13th International Symposium on 3D Web Technology, Aug 2008, Los Angeles, California, United States. pp.35 - 41, 10.1145/1394209.1394220 . hal-01327149

HAL Id: hal-01327149
https://hal.science/hal-01327149v1

Submitted on 6 Jun 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

ShareX3D, a scientific collaborative 3D viewer over HTTP

Sebastien Jourdain
Julien Forest
Artenum*

Christophe Mouton
Bernard Nouailhas
Gerard Moniot
Franck Kolb
EDF R&D†

Sophie Chabridon
Michel Simatic
Zied Abid
Institut TELECOM;
TELECOM & Management SudParis‡

Laurent Mallet
Oxalya§

Abstract

In scientific visualization data are becoming more and more complex and implies cooperative effort for their post-processing analysis as well as high performance processing resources. Moreover experts are frequently geographically distributed and existing post-processing tools do not provide high performance capability with collaborative features. To solve this limitation, a prospective action has been recently initiated in the frame of the ANR SCOS project, with the development of a web based post-processing framework called V3D. SCOS/V3D is based on two innovating concepts: the sharing of the processing results via the transfer of a X3D data file corresponding to the final virtual reality scene and communications based on HTTP to by-pass standard proxies and firewalls limitations. To test this approach, a proof of concept prototype, called shareX3D, has been developed, especially focusing on the events notification, with an implementation of the long polling technique. The architecture of ShareX3D and first tests are presented and discussed.

CR Categories: J.2 [Computer Applications]: Physical Sciences and Engineering—Physics; D.1.1 [Software]: Programming Techniques—Applicative (Functional) Programming D.1.3 [Software]: Programming Techniques—Concurrent Programming - Distributed programming

Keywords: X3D, Web3D, collaboration, web networking, post-processing, scientific visualization

1 Introduction

1.1 3D scientific visualisation and post-processing: new issues and challenges

The continuous improvement of computer performance over the last decades and the progressive generalization of the High Performance Computing (HPC) in 3D modelling enables to perform simulations on constantly increasing grid size. Systems modelled with more than 10^9 cells and taking into account many tens of variables are becoming common nowadays. The size of such models presents a real challenge for data post-processing and visualisation, because they become too heavy to be processed on local post-processing workstations. In addition with the progressive introduction of multi-physics models, the data analysis requires more and more complex treatments, combining an increasing number of basic filters. The global CPU cost is consequently prohibitive and classic approaches based on local post-processing tools are not possible anymore. To address this issue, one solution currently explored is the migration toward a client-server approach [Henderson and Ahrens 2004; Jean M. Favre, Sathya Krishnamurthy and Swiss Center for Scientific Computing 2000; Ahrens et al. 2001; LLNL 2003]. In such design, the data filters are processed on remote high performance computers, using parallelized pipelines, before the final output is returned to the client, generally as a set of 2D views. However, such approach presents several structural limitations. Each action requested by the client leads to large data transfers thru the network. A simple change of the point of view may quickly introduce a significant latency between the action and the rendering of the results in a web context. Most of these solutions are also based on specific network protocols, poorly adapted to the modern security constraints like NAT, firewalls or restrictions to the HTTP protocol only, limiting their deployment in an Extranet context.

Finally, the generalisation of multi-physics models and the increasing complexity of the simulations require more and more co-expertise. This co-expertise implies collaboration between research and expertise areas. Therefore, remote teams have to work together on the same data and share their analysis at the same time. Such needs require an evolution in post-processing tools towards collaboration. For those reasons, existing post-processing tools are not adapted anymore and new approaches have to be explored.

1.2 The SCOS/V3D Project

The aim of the ANR/SCOS [SCOS-project 2006] project is the standardization and the normalisation of scientific software platforms in order to improve the global competitiveness of R&D academics and industrials.The V3D project, initiated by EDF R&D in January 2007 and gathering more than 12 partners in the frame of the SCOS project aims to developed a lightweight Web-based and high-performance 3D visualization framework. This development will include advanced collaborative capabilities and awareness for multi-users data processing and 3D visualization. The main concept of SCOS/V3D is based on a lightweight client-server approach, easy to deploy in an Extranet context as well as a capability to process large data remotely on high performance clusters. A first quantitative objective is to be able to process data sets up-to 10^9 cells and visualise the result on common desktop computers with a 3D rendering client.

To reduce the constraints due to the client-server link, the key part is to transfer the output 3D virtual reality scene, as an X3D/X3Db file, in place of images or video streams. Such model distribution have been done with VRML in [Clifton G.M. Presser 2005]. Thanks to the performances of modern video cards, the final 2D rendering is performed locally and only the 3D navigation operations such as rotation or zooming will be transferred on the network during the analysis time. The 3D virtual scene itself is computed and trans-

*e-mail: jourdain,j.forest@artenum.com
†e-mail:christophe.mouton,bernard.nouailhas,gerard.moniot,franck.kolb@edf.fr
‡e-mail: Sophie.Chabridon,Michel.Simatic,Zied.Abid@it-sudparis.eu
§e-mail: laurent.mallet@oxalya.com

ferred only when a new data treatment is required, for instance when a cutting plane is requested. Once the data processing is done, only the new 3D objects are sent. Thanks to this approach, it is expected to significantly reduce the network usage as well as improving the global reactivity of the application with a lower latency in the scene interaction.

The second innovation is to offer a visualisation service on the Web, via a web portal and a Java based visualization client with collaborative capabilities. This approach should simplify the deployment of the solution and enable distributed experts to work together at the same time on the same post-processing data analysis. The objective is also to provide an application that can be deployed on the Internet with high networking constraints imposed by proxies and firewalls. Therefore, all the networking communication has to be done with HTTP or HTTPS protocol.

2 The ShareX3D demonstrator

A first phase of the SCOS/V3D project was dedicated to the development of a set of demonstrators and proofs of concept in order to identify technological bottlenecks and validate the relevance of chosen designs and selected technologies. Another issue was to explore the new possibilities offered by such tools, especially in the domain of the collaboration, and specify the final user requirements during the prototyping phase.

One of these proofs of concept is ShareX3D, a client-server application that allow groups of users to create and share a visualization study. A name and a virtual reality scene define a visualization study. Once a study is created, the web portal enables users to participate to it in order to share the same point of view with the same virtual reality scene. ShareX3D makes possible the creation and the execution of several studies in parallel, enabling collaborative groups to work on distinct 3D scenes at the same time. The ShareX3D viewer was developed on the XJ3D browser and uses the Java Web Start technology [Sun 2005] to simplify its deployment on most platforms. Practically, the ShareX3D client can only view the 3D scene and share the rotation and zooming actions. Studies can be either dynamic or static, in the sense that the 3D scene can be dynamically updated during the collaboration of a group. This feature will not be discussed in this paper but refers to the goal of SCOS/V3D architecture where the 3D scene is dynamically produced by a distributed post-processing engine deployed on a visualization cluster.

2.1 Goals and issues to overcome

The role of ShareX3D is to validate the global approach based on the transfer of a 3D scene, and identify and test solutions for an asynchronous event based architecture with HTTP protocol. Behind this technical issue, another target is the better understanding of constraints related to a collaborative usage.

ShareX3D remains a proof-of-concept, and does not attempt to be directly reused in operational tools. Regarding the detailed technical issues, ShareX3D focuses on:

3D scene transfer: The first issue is to validate the feasibility to transfer the result under the form of a X3D/X3Db file and more especially to check if the cost of this transfer remains reasonable in time and bandwidth.

Events synchronisation over HTTP: ShareX3D requires an event-driven or server-push technique to offer a low-latency notification, especially from the server to the client. The standard HTTP usage implies synchronous calls from the client to the server and not the other way round. Therefore, we developed a specific asynchronous interaction based on the Comet long polling technique [Russell 2006], which is described below.

Proxy: Another issue is the HTTP proxy compatibility. Proxy can interfere in the communication between the client and the server especially in an asynchronous web architecture. ShareX3D should support and provide a way to configure an HTTP proxy with and without authentication in order to validate the client behaviour in a proxy context.

Simple web deployment: One of the key requirements from the user was a simplified deployment for the client side. The visualization client should be available from the web without any specific skills for its installation and use. The Java Web Start technique was chosen because it is able to install rich applications by a simple click on a Web page and makes possible the usage of native libraries in a transparent manner with automatic system and architecture detection.

Web portal: A web portal was mandatory for the deployment of the 3D viewer, but another notion in the collaboration was needed: the notion of groups of people working together on the same data, that we call a study.

Collaborative post-processing: From the user point of view, one of the goals of ShareX3D is also to explore how a visualisation tool can be used in a collaborative manner.

3 HTTP events notification and network strategy

As introduced above, the main difficulty was the need of an efficient event base notification system over HTTP, especially for server-to-client or client-to-server-to-client notification. A specific implementation was developed on the basis of techniques described in [Russell 2006; Engin Bozdag and van Deursen 2007], and more especially the strategy called *long polling* for the viewpoint notification.

In our implementation of the notification service with the long polling strategy, the client performs a request to the server. If no new event is present on this one, the server locks and maintains open the connection till a new event is available. If a new event is present, the client using the opened-connection directly retrieves it. As soon as the current connection is closed, a new one is open for a new cycle. Such technique presents the advantage to avoid the dependency on an arbitrary period of time, which is the case in standard polling system. For this reason, the server is able to send new information to the 3D viewer with a lower latency. The only drawback of this technique is in case of a high event rate, during 3D navigation for instance; long polling may then produce a significant computational overhead due to the cost of the establishment of a new connection for each new viewpoint. Therefore some testing has been done to quantify this overhead in terms of latency and event rate in the Sec. 5.

The other Comet [Russell 2006] strategy, named *Streaming*, can eliminate this overhead. In this approach, the client opens a single persistent connection to the server for all events. The client handles this connection incrementally. Each time the server sends a new event, the client interprets the message, but the connection is maintained open and is never closed. Generally speaking, the streaming technique has better performance, because it eliminates much of the per-message overhead. This Comet strategy seems more interesting than the long polling one regarding our usage but one of its drawbacks is when the client is behind a Proxy that caches the

networking payload. In such case, the expected low latency might become high.

For these reasons, the streaming strategy has not been tested yet in the frame of ShareX3D and the focus was put on the long polling technique as a first step.

In practice, the user requirements of SCOS/V3D have shown that a latency below 600 ms and an event rate higher than 6 $events/s$ were enough to have an application fluid enough to be practically used. Those values have been reached by the long polling technique.

Finally, the best approach would be to enable the client to choose one of the two strategies depending on its network environment.

4 Architecture and design

As introduced above, ShareX3D is a client-server application. Fig. 1 shows the global design of Share3XD. The central component is the collaborative server, which maintains the consistency of the whole system inside studies (viewpoint and 3D scene). ShareX3D provides two different clients: the *ShareX3D Client*, detailed below, and a post-processing client, which can update the 3D scene by a dynamic post-processing generated 3D scene. Regarding this specific point, the design of ShareX3D differs deeply from the architecture of V3D, where the processing engine will be deployed on the server side cluster.

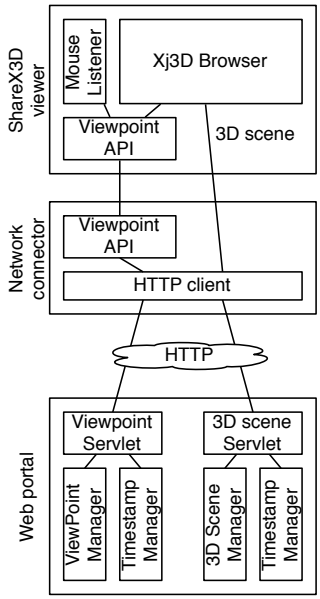

Figure 1: *Component architecture*

4.1 ShareX3D visualization client

Fig. 2 shows a screenshot of the ShareX3D Client. The ShareX3D Client is based on the Xj3D browser [Xj3D-team], which provides the 3D rendering. As the default navigation available in Xj3D did not fit the needs in post-processing data analysis, a new navigation mode based on SAI manipulation of the viewpoint node and a Java mouse listener has been developed and submitted as contribution to the Xj3D developer team. Moreover, for our needs, we had to provide a specific simple viewpoint API enabling viewpoint listening and setting.

The standard ShareX3D Client is a passive client, which does not process data nor generate a 3D scene. It focuses on the 3D rendering and viewpoint sharing. We developed a network connector between the central web server and the client in order to distribute the viewpoint and the 3D scene. Figure 3 illustrates both the network connector and the 3D viewer application listening each other for a global viewpoint sharing. When a navigation action (e.g. rotation) occurs in the 3D scene, the network connector is notified by the application. The network connector sends this new viewpoint to the web portal for a global distribution. Reciprocally, when the network connector receives a new viewpoint from the web portal, the corresponding viewer is notified and updates its viewpoint.

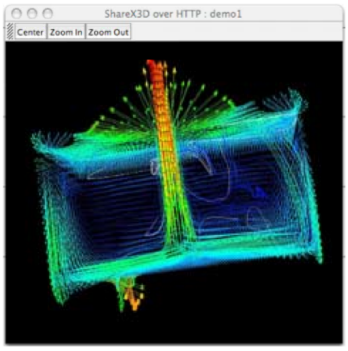

Figure 2: *View of the* Share3D Client*, based on XJ3D. This client can be launched by a simple link on a Web page using the Java Web Start technology. All exchanges with the* ShareX3D Server *are done through the HTTP protocol and pass firewalls and proxies.*

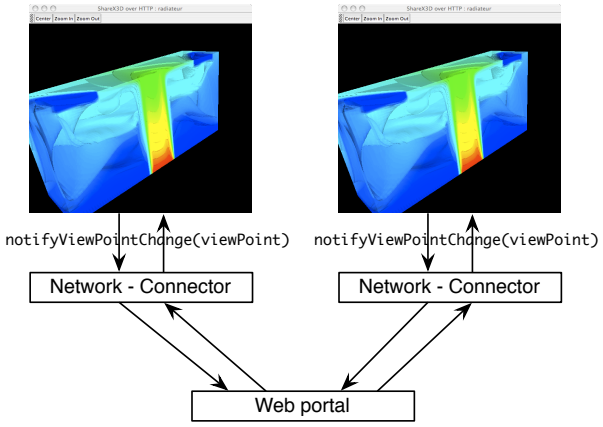

Figure 3: *Viewpoint listener architecture*

4.2 ShareX3D Server side

The server side of ShareX3D includes a Web portal and a collaboration service.The web portal enables the user to reserve collaboration space where people can meet and share their 3D scene as well as their navigation experience. Fig.4 gives a view of the Web portal with a launched client. The Web portal offers all the lifecycle management features, like creation, suppression and participation to a study.

Regarding the collaboration service, the first approach was to use an XMPP server with an HTTP binding such as Bosh [Ian Paterson, Dave Smith and Peter Saint-André 2007]. Some XMPP [IETF 2002-2004; Jabber-community 1999] servers were providing this

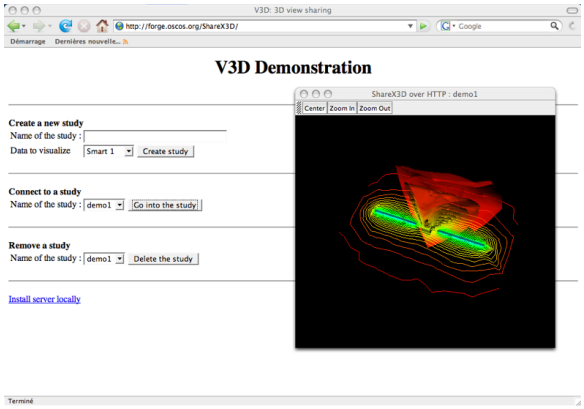

Figure 4: *View of the* Share3D Web Portal *with a client launched. The left panel shows the access to the various studies and the links to launch the various clients.*

support but none of the Java clients was implementing this XMPP extension. Therefore, we look at the specification of Bosh [Ian Paterson, Dave Smith and Peter Saint-André 2007] and we roughly implemented it through simple Java objects such as servlet and HttpURLConnection to avoid heavy external libraries on client side. Moreover, we have chosen a REST [M. zur Muehlen, J. V. Nickerson and K. D. Swenson 2004] architecture regarding our needs of simplicity and efficiency. Three controllers were developed as servlets for this service:

1. The ActionServlet manages the study's actions such as creation and suppression.
2. The SceneServlet manages the update and the access to the latest 3D scene available for a study.
3. The ViewPointServlet manages the broadcast of viewpoints. The corresponding source code is given in Fig. 5 and 6.

```
void doGet(HttpServletRequest req, HttpServletResponse resp)
              throws ServletException, IOException {
   if (manager == null)
      manager = SessionManager.getInstance();

   // Extract infos from the request
   String sessionId = getSessionId(req);
   long localTimestamp = getTimestamp(req);

   // wait for the next time-stamp
   manager.getViewPointTimestampManager()
          .waitForNews(sessionId, localTimestamp);
   PrintWriter writer = resp.getWriter();
   ViewPointBean tmp = manager.getViewPointManager()
                              .getViewPoint(sessionId);
   // Write viewpoint
   if (tmp == null) {
      DEFAULT_VIEW_POINT.writeViewPoint(writer);
   } else {
      tmp.writeViewPoint(writer);
   }

   // Write time stamp
   writer.write(
       Long.toString(manager.getViewPointTimestampManager()
                            .getSessionTimestamp(sessionId)));
   writer.write("\n");
   writer.flush();
}
```

Figure 5: *Servlet method used for Server to Client viewpoint notification*

The viewpoint synchronisation uses four components to implement

```
void doPost(HttpServletRequest req, HttpServletResponse resp)
                  throws ServletException, IOException {
   if (manager == null)
      manager = SessionManager.getInstance();

   // Extract infos from the request
   String sessionId = getSessionId(req);

   // Read the sended data
   BufferedReader reader = req.getReader();
   ViewPointBean vp = new ViewPointBean();
   vp.readViewPoint(reader);
   reader.close();
   manager.getViewPointManager().setViewPoint(sessionId, vp);
   manager.getViewPointTimestampManager()
    .nextTimeStamp(sessionId);
}
```

Figure 6: *Servlet method used for Client to Server viewpoint notification*

the long polling technique presented in Sec. 3:

1. The ViewPointServlet is the HTTP connector of the server that uses the following components.
2. The Session manager centralizes the access to each component of the application.
3. The ViewPoint Manager stores the last available viewpoint for a given study.
4. The TimeStamp Manager keeps track of the last timestamp of a viewpoint for a given study. It is used in the ViewPointServlet to wait till a new point of view is available.

4.3 Collaboration sequence

A concrete collaboration sequence based on the technique described in Sec. 3 can be summarized as follows. This scenario is described in Fig. 7. Each point of view is labelled with a timestamp in order to determine which client has the latest point of view. The server incrementally updates theses timestamps after each new viewpoint submission. Therefore, when a client participates to a study, it asks the server for a new viewpoint regarding its local timestamp and study. The study name and the local timestamp are given to the server by the request URL following the REST philosophy. When the server receives this request, the servlet asks to the TimeStamp manager if it has a timestamp equal to the client's one. In such case, the client is up-to-date and should wait for a new viewpoint, so the TimeStamp manager stops the servlet thread. Otherwise, if the timestamp manager has a higher timestamp, the servlet is not stopped and asks for the last stored viewpoint in the ViewPointManager. Both viewpoint and timestamp are then written on the response to the client. The client can then update its viewpoint and its local timestamp.

When a 3D viewer sends its viewpoint to the server, only the study name is needed. Once a viewpoint is received by the server, it is stored in the ViewPoint Manager. The TimeStamp Manager increments its timestamp and wakes up all the waiting servlet threads.

Please notice that a viewpoint is simply defined by the camera orientation, which is a simple quaternion, its position, the centre of rotation (i.e both vectors of dimension 3) and an identifier to track the origin of a viewpoint. Therefore, the weight of the message transferred on the network remains very light. The update of the 3D scene is made exactly the same way with an open connection, which is locked until a new 3D scene is dropped on the server. The 3D scene is then downloaded by the connected clients and displayed locally.

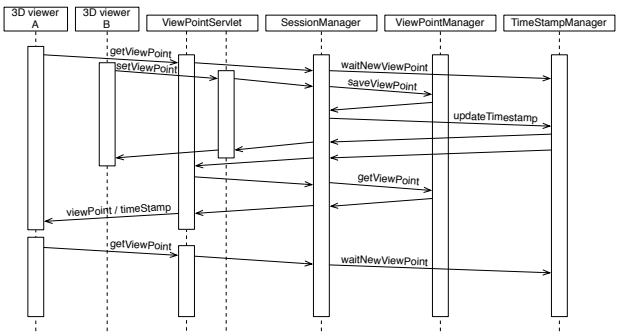

Figure 7: *Networking sequence diagram for a viewpoint sharing.*

Finally by sharing the point of view, every 3D viewer is sharing the same view even if they do not share the same application size or screen resolution.

5 First tests and results

A first benchmark was performed in order to evaluate the global performances of the selected approach from a user point of view. More precisely, three aspects were studied in details: the download time for the 3D scene, the general latency of the system regarding the events dispatching and the event rate, i.e. the maximum number of events that can be transferred to a client per second.
The download time of the 3D scene corresponds to the time needed by the client to download the 3D scene that is provided by the server. This time can be crucial for the usability of the system where the post-processing will be done during a collaboration session on the server side.
The latency of the event propagation is especially critical in co-expert mode, where the discussion is related to a specific viewpoint and not the previous one. Viewpoints as well as collaborative events have to be synchronized in line with the analysis talk or at most desynchronized with a small latency.
The event rate directly characterizes the fluidity of the navigation or the interaction, for instance during the remote rendering of a rotation in the 3D scene. In some way, this point can be seen as a frame rate.

In order to be close to a real operational environment, benchmarks have been done on standard hardware (server and local computer). The server side was deployed on a simple Tomcat Web server, without specific settings or optimization. Two different environments where tested in order to determine the boundaries of the system. The first evaluation was done in poor network conditions (i.e. latency, weak bandwidth) where a simple public ADSL connection was used and the server side was behind an Apache reverse proxy. In this evaluation, it is important to notice that the server and the clients were distant of several hundreds of kilometers.
The second benchmark was done on a local network loop of a computer. This local network has been tested the same manner as the remote one. The table 1 summarizes both the local and the remote networks characteristics on which the tests have been performed.
All the tests and performance analysis have been done with the same computer with specific testing applications, using multi-threading technology to simulate several concurrent clients. Those threads were behaving like a 3D viewer regarding the network operations. All the values given in figure 8 and 9 are average values computed on 100 testing events and among the whole connected clients.

	localhost	forge.oscos.org
Ping	0,1 ms	$51 \pm 1\ ms$
Download	35.44 MB/s	260,75 kB/s
Upload	35.44 MB/s	17 kB/s

Table 1: *Network testing environment*

5.1 Reactivity and fluidity

The figure 8 displays the latency versus the number of connected clients for event transmission from one client to another, in local and remote contexts. Events have been generated by a unique thread sending as many possible events per second. That latency has been computed by the difference between the emission and the reception time by the clients. The displayed value corresponds to the average for all the receiving clients on 100 testing events. The error bars are based on the standard deviation of the test case. One should notice that in the present implementation of event broadcasting, an event may be masked by a more recent one if the receiver did not have the time to download the previous event before the new one is available. The measured latency does not take into account the possible loss of events between two received events and only focuses on the events received on the client side. The influence of the message size was not studied, regarding that this implementation was limited to viewpoint events.

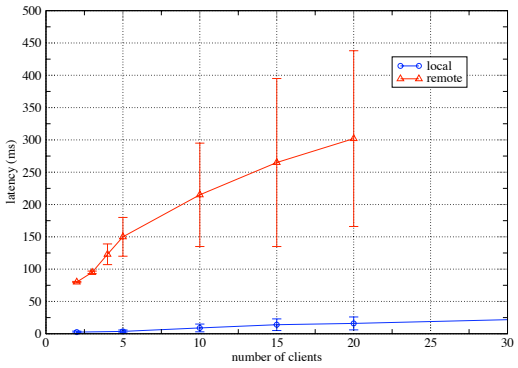

Figure 8: *Latency versus number of connected clients.*

In local mode, this test has been done with one hundred clients and even with that case, the latency remains below the $80\ ms$. This test gives us a first idea of the scaling capability of a centralised architecture like the proposed one in a corporate context. Due to the level of expertise required to use scientific visualisation tools, it is not expected to have more than ten clients working on the same study at the same time. However, in a large company or research centre and in an Intranet context, one can easily imagine a quite large number of users working at the same time on various studies. The future SCOS/V3D server should support such load and be able to maintain a global reactivity for a large group of users. This first result confirms that the proposed design may address this issue.

In remote mode, the cost of the network is visible with an average latency rising up to $300\ ms$ for 20 concurrent clients. However, the testing was made on the same computer with the same network connection to the remote server. Therefore, each client connection was sharing with others their poor networking connection which

could explain the drop of efficiency which was not highlight by the local testing. Actually, the latency is mainly critical in co-expertise mode, where different users have to react in real time to their reciprocal actions. It seems difficult to imagine such mode with more than three or four connected clients. In this case, the latency remains good with values below 150 ms.

The figure 9 shows the events rate for remote server condition in both sending and receiving mode. The event rate illustrates the fluidity of the collaborative interaction. Like in the latency test, the set of events was generated by a single thread sending as many as possible events per second. In such case, for instance, the similitude with the frame rate on the remote scene is quite straightforward.

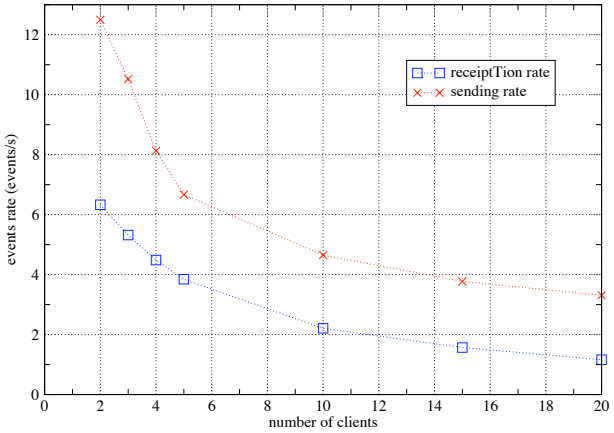

Figure 9: *Event rate: The lowest curve gives the practical event rate, as felt by the user. Up to six clients, the event rate remains high enough (5 events/s) to give an impression of fluidity in continuous actions on the 3D scene.*

The difference between the sending and the receiving event rates is difficult to explain especially in the remote case. No hazardous explanation will be given in this paper, and we will focus on the resulting values. Therefore only the limiting factor, which is the lowest rate, should be taken into account to evaluate the global event rate.

The reception rate curve shows that up to five connected clients, the application remains fluid enough, with more than 4 $events/s$, for a continuous and smooth rendering, which is mostly critical in co-expert mode. Up to 10 connected clients, the event rate reaches 2 $events/s$, which can be seen as insufficient in first reading but the latency in that case is still lower than a $1/4$ s which allows the client to follow the same view as the master one with few jerks.

Moreover, the present implementation is not optimised and should be seen as a worst implementation case. For instance, the connection cost on the receiving connection is still currently prohibitive. A better management of connections and/or the implementation of a streaming approach may induce a significant speed-up with a global event rate close to the sending rate curve and even a bit higher.

These results should also be compared to the performances of more standard architectures such as "remote display", where the data exchange is based on image streaming. Some tests made in a study of EDF R&D on an Image Streaming system have shown an average of 20 MB/s of consumed bandwidth on a local network during 3D scene rotation with a frame rate of 20 fps with only one connected client. This frame rate linearly decreases regarding the number of connected clients. With four connected clients, an Image Streaming system would provide only 5 frames per second with the same consumed bandwidth. Regarding those results, a technology based on 2D image transfer is not possible in collaborative context even on a local and high performance network.

Number of clients	localhost	forge.oscos.org
2 clients	333 ± 110 e/s	6 e/s
3 clients	-	5 e/s
4 clients	-	4 e/s
5 clients	166 ± 63 e/s	3 e/s
10 clients	90 ± 35 e/s	2 ± 1 e/s
15 clients	62 ± 22 e/s	1 ± 1 e/s
20 clients	47 ± 15 e/s	1 ± 1 e/s
100 clients	12 ± 3 e/s	-

Table 2: *Event rate: Number of events that can be received in 1 second*

If we compare an Image Streaming system with ShareX3D on a local network with the same networking configuration, the Image Streaming system is able to serve one remote client with a frame rate of 20 frames per second whereas ShareX3D, after an initial download, will serve 50 clients with the same frame rate. Moreover an Image Streaming system will be able to serve 20 clients with 1 frame per second on a local network of 1Gbits/s where ShareX3D will be able to serve 20 clients with the same frame rate on the Internet with a 2Mbits/s ADSL connection. In the case of 3D rotation, ShareX3D requires 500 times less bandwidth than an Image Streaming system.

5.2 Data transfer and 3D scene update

Another important point is the initial downloading time of the 3D scene, which could be crucial, especially on an Extranet. On a local network, the worst case given in the table 3, with a X3D file of $11.9MBytes$ corresponding to an original data size of 8.10^6 elements, will only take 336 ms to download. However, on an Extranet, the same file will require 46 s on a 2 Mbits/s ADSL connection. Such time could be seen as prohibitive for an operational use, but this requires a deeper analysis. First of all, table 3 shows that the size of the final X3D file depends on the complexity of the scene. For simple scenes, like a cutting plane as in case A, the final X3D file may be 6 times smaller than the most complex one and 65 times smaller than the initial data set. By such approach the compression is done without any loss of relevant information. The tests have voluntary been performed on a very poor network. Speed-up of a factor ten can be expected nowadays and it does not seem impossible to download scenes of tens of MBytes in less than 20s.

Another point regarding limit cases is the fact that the complex scenes are composed of several 3D objects, progressively computed through several requests during the collaboration period. This means that with an incremental update mechanism, as provided in V3D, each Internet client will have to wait less than a few seconds after each new post-processing action. Moreover, with partial update, a 3D scene transfer based approach becomes fully acceptable in an extranet context.

It is also important to notice that the protocol used in ShareX3D is HTTP and not a UDP or TCP optimised connection and the scene is still currently basic. Works on direct compression of the 3D structure are pending in the frame of SCOS/V3D and are very promising.

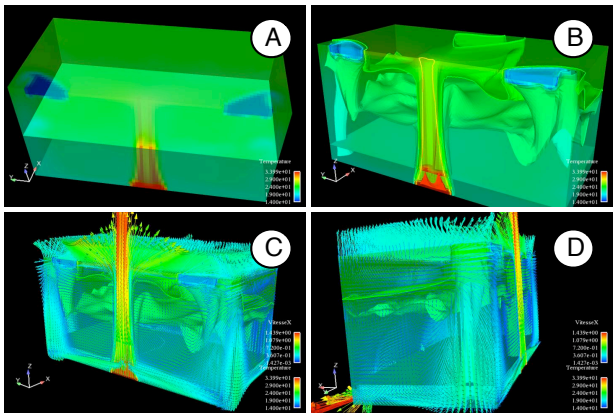

Figure 10: *Sample of 3D scene for file size comparison*

Number of hexahedron	128.10^3	1.10^6	8.10^6
Initial numerical data size	8,84 MB	70,2 MB	559 MB
Scene A	128 kB	540 kB	2,30 MB
Scene B	331 kB	1,40 MB	5,85 MB
Scene C	620 kB	2,71 MB	11,2 MB
Scene D	630 kB	2,86 MB	11,9 MB

Table 3: *Post-processing X3Db file size*

6 Conclusion

ShareX3D is a first implementation of a collaborative 3D viewer based on HTTP communication. It aims to validate architectural choices made in the specification of V3D and removing technical issues. ShareX3D has proven that collaboration over HTTP is possible with high network constraints in terms of bandwidth and security. Firsts tests have also shown that the long polling technique provides enough fluidity and reactivity for collaborative scientific visualisation. X3D files download tests have shown that very large scenes remain costly to update in one piece. However, ShareX3D has confirmed the global approach and presented interesting performances in respect to a classic image transfer architecture. The observed limitations might be pushed back by a partial update of the 3D scene and by introducing data steaming and compression, as expected in V3D. Moreover the tests made around ShareX3D have clarified the needs in terms of collaboration and interactivity for V3D. ShareX3D has also opened new possibilities for scientific collaborative visualization tools over HTTP.

Acknowledgements

This work was partially founded by the French National Research Agency (ANR) in the frame of the RNTL SCOS project and an internal R&D effort of each actor. We would like also to thanks all members of this project and the ANR/Part@ge project for their support and the fruitful technical exchanges and discussions.

References

AHRENS, J., BRISLAWN, K., MARTIN, K., GEVECI, B., LAW, C., AND PAPKA, M. 2001. Large-scale data visualization using parallel data streaming. *Computer Graphics and Applications, IEEE 21* (Jul/Aug), 34–41.

CLIFTON G.M. PRESSER. 2005. A java web application for allowing multiuser collaboration and exploration of existing vrml worlds. *Web3D symposium*, 10 (March), 85–92.

ENGIN BOZDAG, A. M., AND VAN DEURSEN, A., 2007. A comparison of push and pull techniques for ajax. Delft University of Technology Software Engineering Research Group Technical Report.

HENDERSON, A., AND AHRENS, J. 2004. *The ParaView Guide*. Kitware.

IAN PATERSON, DAVE SMITH AND PETER SAINT-ANDRÉ, 2007. Bidirectional-streams over synchronous http (bosh). XMPP Standards Foundation, XEP-0124, February.

IETF, 2002-2004. Xmpp standards foundation. http://www.xmpp.org/.

JABBER-COMMUNITY, 1999. Open instant messaging and presence. http://www.jabber.org/.

JEAN M. FAVRE, SATHYA KRISHNAMURTHY AND SWISS CENTER FOR SCIENTIFIC COMPUTING. 2000. Visualization tools and environments for very large data. *EPFL Supercomputing review*, 12 (November), 9–12.

KITWARE. VTK File formats. http://www.vtk.org/pdf/fileformats.pdf.

LLNL, 2003. Visit getting started manual - lawrence livermore national laboratory. https://wci.llnl.gov/codes/visit/1.1/GettingStarted.pdf, February.

M. ZUR MUEHLEN, J. V. NICKERSON AND K. D. SWENSON, 2004. Developing web services choreography standardsthe case of rest vs. soap. Decision Support Systems.

RUSSELL, A., 2006. Comet: Low latency data for browsers. O'Reilly Emerging Technology Conference presentation, March.

SCHROEDER W. J., MARTIN K. M. AND LORENSE W. E., 1996. The design and implementation of an object-oriented toolkit for 3d graphics and visualization. Proceedings of the 7th conference on Visualization 96, IEEE Computer Society Press, Los Alamitos, CA, USA, 93ff.

SCOS-PROJECT, 2006. Scos project's web portal. http://www.oscos.org/.

SUN, 2005. Java web start overview. White Paper, May.

WEB3D-CONSORTIUM, 2005. Extensible 3D (X3D) encodings. ISO/IEC 19776:2005, http://www.web3d.org/x3d/specifications/.

WEB3D-CONSORTIUM, 2006. Extensible 3d (x3d) - part 1: Architecture and base components. ISO/IEC 19775:2004/Am1:2006, http://www.web3d.org/x3d/specifications/.

XJ3D-TEAM. Xj3d open source vrml/x3d toolkit. http://www.web3d.org/TaskGroups/source/xj3d.html.