3DStream
Vision, Graphics and Interactive Systems project

Florent Guillaumie Romain Klein Thierry Plesnar

10^{th} semester, Fall 2010

To be evaluated on June 14^{th} 2011

TITLE:

3D Stream

PROJECT PERIOD:

VGIS10,
From February 1st 2011
to June 31st 2011

PROJECT GROUP:

1025

GROUP MEMBERS:

Florent Guillaumie
Romain Klein
Thierry Plesnar

SUPERVISOR:

Claus Brøndgaard Madsen

CENSOR:

Helge B.D. Sørensen

NUMBER OF COPIES: 5

REPORT PAGES: 62

APPENDIX PAGES: 4

TOTAL PAGES: 66

SYNOPSIS:

This project evaluates several ways to stream huge 3D objects on the Internet. The system had to allow the user to see these objects in his browser and in real time, despite a size that can reach several hundred mega bytes.
This project was made in collaboration with the C2RMF - Louvre museum. It will mainly focus on the streaming of work of arts which are digitized in the PLY format.
The discussion will have to address the performance issues because of the size of the objects which have to be handled.

Acknowledgement

We would like to thank the Aalborg University staff who supervised, supported and helped us during both researches and implementation of our project:

Claus Brøndgaard Madsen, who supervised our work and the general project progress.

We would like also to thank the "Centre de recherches et de restauration des musées de France" of the Palais du Louvre team, especially:

David Kolin and his assistants, Yann Ledudal and Guillaume Blaise.

A lot of work and improvement have also been made thanks to feedbacks from the following persons which we would like to thank. They helped us designing our final application by participating in our usability and user tests, giving us their personal opinion about it, and helping us to adapt our first idea to a useful and practical application:

Gabrielle Tranchet, Paweł Pankiewicz, Anders Nørgaard, Alexandre Majetniak, Branko Plesnar, Iraporan Plesnar, Simone Klein, Dominique Klein, Agnès Guillaumie, Philippe Guillaumie, Jérôme Arlet, Élise Arlet, Ninna Marie.

Preface

Technology notice

This document has been written with LaTeX, the document markup language and compiled with pdftex. The images in this document are all free of rights or have references, and have been partially retouched with Adobe Photoshop CS5 and PixelMator. The graphics have been made with Microsoft Office Project 2007.

The application has been developed in Javascript and WebGL. Linux server uses Apache and MySQL on an Ubuntu 10.10 distribution.

Report outline

This document goes from the very beginning of our project to the final user testing.

First, in a quick introduction, we will present you an overview of our product, the purpose of the study and the different fields involved.

Then, in the requirements analysis part, we will state the basis of the project, identify the stakeholders and give an overview of the technologies encountered such as the WebGL.

Also, in the design section, the way we planned to develop our application will be presented using mostly UML diagrams.

Afterwards, the implementation part will go deeper into our technical solutions to solve the problem.

Finally, the system will be tested and the results will be given in the tests section. To conclude, this report sums up our investment into this project and gives us some working methods for the future.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Project overview

Nowadays, the tendency to go from the 2D to the 3D is real. This comes directly from the fact that the viewer will find it more realistic. The most common example is that more and more movies are now in 3D to enhance the experience of the audience. The use of 3D objects is also increasing at the expense of images, but for different reasons. 3D allows to have way more details on the object than simple images could ever provide because of the interactive control of the viewpoint. But it has a cost, the size of such files can reach hundreds of mega bytes.

The issue with such large files is that you need to entirely download it before being able to observe it thanks to a local application. For instance, the 3D file at page 9, which comes from The Stanford 3D Scanning Repository[1], has a size of 180Mb. If we arbitrarily consider that the average download speed is between 500Kb/s and 4Mb/s, we can estimate that it would take between 45 seconds and 6 minutes to get it. Then you still need to manually open this file with your local application which will take some extra seconds to load it. In the end, it takes quite a long time to be able to examine a 3D object and this is the problem we want to address.

The solution we decided to explore to answer this problem, is the streaming of the 3D object over the Internet. This technique allows the viewer to watch the object appearing on his screen as it is being downloaded and keeps the model from being distributed (for copyright reasons).

[1] http://graphics.stanford.edu/data/3Dscanrep/

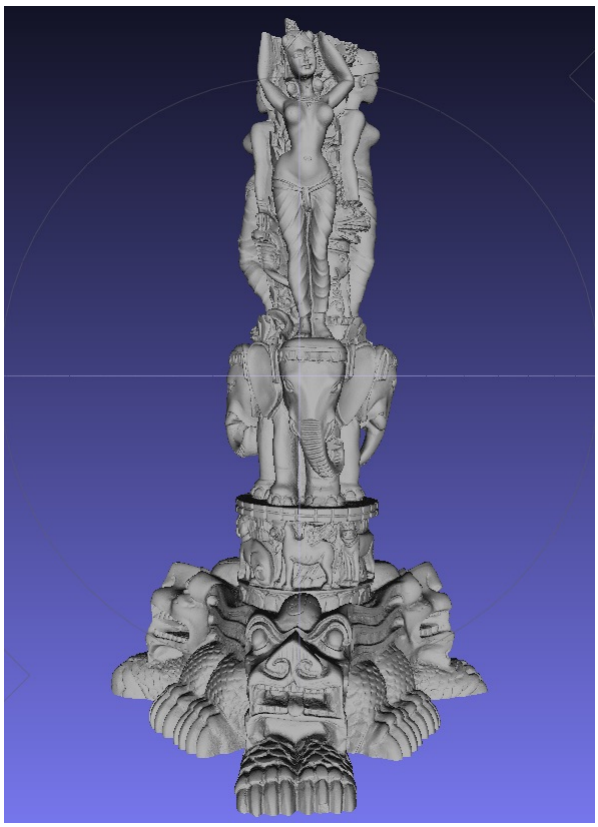

Figure 1.1: Example: A Thai Statue rendered in Meshlab

The web technologies to manipulate 2D images are really advanced and well established. People are finally realizing that the technology for the visualization of 3D models online is now available. Such a technology can be quite useful to improve the user experience on the web. For instance, by having virtual visits of museums, ordering a personalized kitchen online and being able to change any part of it while seeing directly the result, etc.

The first 3D technology for the web was actually presented in 1994 and was called VRML (Virtual Reality Modeling Language). It was simply based on a text file describing a virtual scene. However, this language required a plugin to be installed in the browser of the user and used a lot of processing power. In the end, this technology was only used by a handful of people. The X3D, which was released by the Web3D consortium in 2001, was an improved version of VRML but still required a plugin for the browser. The success of these languages was very limited.

Finally, with the definition of the `HTML5` standard, a new technology non-plugin dependent emerged, the WebGL, which is now supported by the last

version of most of the browsers. It is independent from HTML5 but relies on the <canvas> element of the standard and JavaScript. [28]

Example of use [35]:

```
<body onload="webGLStart();">
  <canvas id="canvas" style="border: none;" width="500" height="500">
  </canvas>
</body>

function webGLStart() {
    var canvas = document.getElementById("canvas");
    initGL(canvas);  //These functions are
    initShaders();   //created to load the WebGL
    initBuffers();   //but not defined in the standard
    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.enable(gl.DEPTH_TEST);
    drawScene();
}
```

We came in contact with the C2RMF - Louvre museum to work on a project for them. The aim is to create a client-server architecture to stream 3D digitalized objects. The main goal is to create a system to stream files produced by the C2RMF (Centre de Recherche et Restauration des Musées de France) which is in charge of creating 3D models of masterpieces provided by the Louvre museum thanks to a scanner.

There are many ways to scan real-life objects, in our case sculptures or crockeries, to obtain a digitized or scanned version in three dimensional models. Each of these technologies has advantages and drawbacks considering limitations, like shiny or transparent objects, cost or precision. Some of them can directly capture the color data, whereas other need to take a 2D picture and apply the texture. Considering the amount of techniques and the slight differences that can sometimes appear between them, we will only describe the ones that we find the most relevant. [29, 13, 33]

- *Contact:* A stylus touches the object to create a point-cloud, because it is pure mechanical, it can be very precise and it is not sensitive to the object aspect (transparency or non-reflection for example). The main drawback of this technique is that it can damage the object, for that reason, it is not used in cultural heritage but mainly in industry.

Figure 1.2: A coordinate measuring machine

- *Destructive slicing:* This method is very simple: the object is grind and at every pass of the grinder, a picture or a 2D scan is taken. of course at the end of the process, the object is destroyed, but this technique provide one of the most accurate result, if has for example been used on frozen corpse to improve medical knowledge of human body. Of course, this method can't be applied on work of art!

- Active

  - *Time-of-light:* The principle of this method is quite simple. The light, even if it is very fast, needs a certain amount of time to go from one point to another. Since the laser has only one precise wavelength, and that we can precisely know when we turn this laser on, we just have to check an impulse of the laser wavelength. The electronic system speed allows a precision of around one millimeter with this technique, and it works for very large distances but it is very sensitive to surface mirroring. The precision of this method can be improved based on the phase shift of the laser.

Figure 1.3: A lidar time-of-light scanner

– *Triangulation:* This method is also laser based but works because we know the distance between the sensor and the laser emitter along with the angle between the laser and the sensor-laser side. Those two values are not supposed to change. The measurement will focus on the angle between the sensor-laser side and the laser dot on the object. By knowing those three information, we can easily deduce the distance between the sensor or the laser emitter. Triangulation is more accurate than time-of-light but can operate only few meters around the scanner.

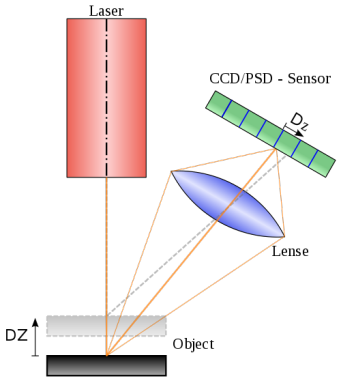

Figure 1.4: Triangulation principle schema

– *Structured light:* This is the main technique used at the C2RMF to digitize objects with a NextEngine scanner. Fundamentally,

a known pattern is emitted in front of the object and a sensor captures the differences between how the pattern should look like and the distorted version reflected by the object. These scanners are very fast because of the pattern which allow to consider many points at the same time. The precision can be quite impressive considering the price and the speed of these scanners, for example, the NextEngine has an accuracy of 0.1 millimeters at its higher resolution.

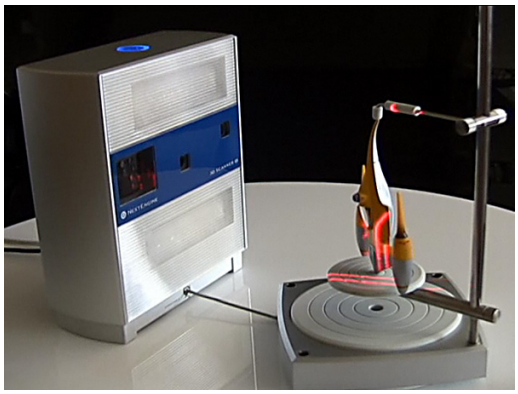

Figure 1.5: The NextEngine scanner

- Passive
  - *Silhouette:* Many pictures of the object are taken in front of a known background with slight angles changes. The different silhouettes are combined to design the shape of the object. This method is slow, not very accurate and cannot detect concavities because it is only sensitive to the maximal surface of the object. The advantages are that the technique is cheap and that the color data can be easily captured.
  - *Stereoscopic* This method is based on the animal stereoscopic vision: two sensors are placed at a known distance. The difference between the two viewpoints, coupled with a treatment algorithm, allows to know the distance between objects and the system. This method can be really fast, however it is not very accurate.

Based of these scanned objects, this project aims to create a tool designed for the curators. The Louvre Museum and the C2RMF have expressed the need of this tool for some reasons: reducing manipulation of fragile masterpieces, allowing foreign experts to work on a piece located too far and

allowing tags/labels in order to create collaborative work. In fact, some visualizers have been created to display high definition images, but none have approaching functionalities for 3D models. Even if all these features are not going to be implemented, we have to consider them when we will have to make some choices in order to ease their further implementation.

1.2 Purpose of the study / Problematics

The purpose of this study is to find a way to display, in the most efficient way possible, a 3D file which can reach several hundreds mega bytes. This can be interpreted in different ways. The viewer has to process the faces efficiently. The transmission of the data can be optimized. The possibility of pre visualizing the object as it is being downloaded can be implemented. All those paths will be explored for our project.

1.3 Domains involved in the study

1.3.1 PLY files

The C2RMF produces 3D files at the PLY format with such scanners, which is why we will only focus on this format.

1.3.1.1 Introduction to PLY file format

In this part will be introduced the PLY polygon file format, also known as the Stanford Triangle Format. PLY files are used to store three dimensional data, usually generated by 3D scanners, that are described as a collection of polygons. This format has been developed in the 90s by Greg Turk, a researcher in the field of computer graphics. PLY files support a description of a single object as an assembly of vertices, faces and other elements, along with properties such as color, transparency, surface normals, texture coordinates and data confidence values.

1.3.1.2 The digital Michelangelo Project

The Digital Michelangelo Project[2] at Stanford University used this PLY home-grown file format to obtain an extremely high resolution 3D scan of the David

[2] http://graphics.stanford.edu/projects/mich/

sculpture. They obtained a full-resolution 3D model of Michelangelo's 5 meter statue of David with about one billion polygons. This project allowed us to get a better understanding of the progress in the field of 3D modelling and rendering.

Figure 1.6: Laser scan of the David. The Digital Michelangelo Project.

1.3.1.3 PLY file format

PLY file format has two sub-formats: an ASCII representation to easily get started, and a binary version for compact storage, quick saving and loading. The structure of a typical PLY file is the following:

1. Header
2. Vertex List
3. Face List
4. (lists of other elements)

The header specifies the elements of a mesh and their types, followed by the list of elements itself, usually vertices and faces. But both the ASCII and the binary files header are in ASCII text, only numerical data following

the header can be either of those. The header always starts with a line containing the word *ply*, which identifies the file as a PLY file. The second line indicates in which format is the PLY file. Comments may also be added using the keyword comment at the start of the line. The "element" keyword introduces the description of how particular data elements are stored and how many of them there are. Finally, at the end of the header there must always be the line *end_header*. Here is a simple example[10] of a red cube to better understand this format:

```
ply
format ascii 1.0            { ascii/binary, format version number }
comment this file is a cube { comments are keyword specified }
element vertex 8            { define "vertex" element, 8 in file }
property float32 x          { vertex contains float "x" coordinate }
property float32 y          { y coordinate }
property float32 z          { z coordinate, too }
property uchar red          { vertex contains uchar "red" color }
property uchar green        { vertex contains uchar "green" color }
property uchar blue         { vertex contains uchar "blue" color }
element face 6              { there are 6 "face" elements }
property list uint8 int32 vertex_index
                            { "vertex_indices" is a list of ints }
end_header                  { delimits the end of the header }
0 0 0 255 0 0               { start of vertex list }
0 0 1 255 0 0
0 1 1 255 0 0
0 1 0 255 0 0
1 0 0 255 0 0
1 0 1 255 0 0
1 1 1 255 0 0
1 1 0 255 0 0
4 0 1 2 3                    { start of face list }
4 7 6 5 4
4 0 4 5 1
4 1 5 6 2
4 2 6 7 3
4 3 7 4 0
```

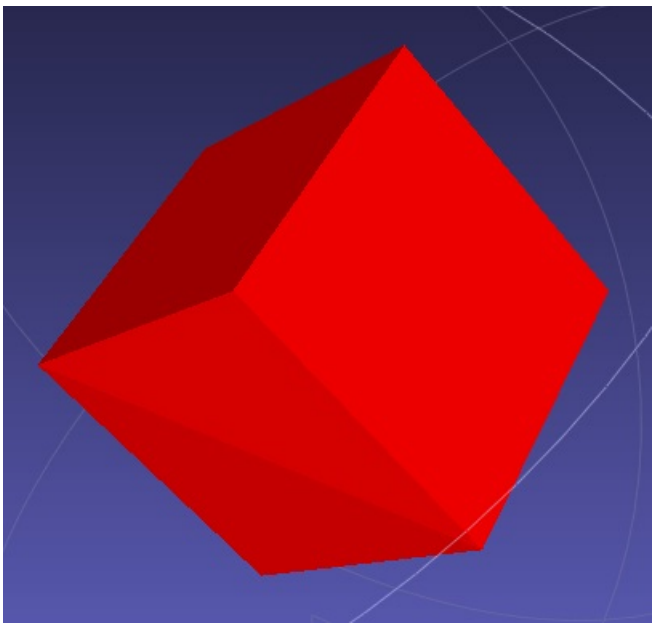

Figure 1.7: Result: The red cube rendered in Meshlab

1.3.2 JavaScript

JavaScript is a language commonly used by every major modern Internet browser. It is based on E.C.M.A.[3] stabilized and standardized implementation (and should be named ECMAScript). [15] Even if every browser has its own implementation of this language, which requires sometimes tricks to make code work properly on every interpreters, JavaScript is nowadays used at a very large scale to give to HTML pages breakthrough capabilities. It provided unbelievable functionalities to common applications at the time of its first version already.

By far, the most common variant of JavaScript is client based, where the script is interpreted in the web browser of the client, but ECMAScript was designed to be embedded in any kind of application [15] which makes this language very versatile, even more with its last specifications and implementations. JavaScript was made to handle, in a simple way, objects provided by the application hosting the interpreter: these objects can be DOM elements but also files not supported by the englobing software.

Right now, JavaScript is considered as the Next Big Language thanks to its architecture (dynamic, object-oriented) but also because last implementations

[3] European Computer Manufacturer's Association

of the client based JavaScript interpreter providse the user with amazingly fast computations. [30] This speed is profitable to interact with web pages, but also for all the "hidden" calculations which are sometimes needed before displaying an object.

We will use this language in our project because rich interactions like the ones provided in our project rely on AJAX technologies completely based on JavaScript objects. Another point to consider is that any 3D file format is natively handled by modern web browsers, and the PLY file is nothing more than a text-file. To convert the information included in these files into pixels displayed on the client screen, calculations need to be done and can be entirely coded in JavaScript.

Chapter 2

Paper overview

About twenty years have passed since the creation of LYNX, the well known text web-browser. During those two decades, browsing the web has radically changed: right now all major bowsers are able to style the text, display images, create scalable vectors, play sounds and videos, interact with plugins and now... display 3D objects! This capability is a great milestone to provide multimedia through the Internet, and because it is standardized and does not rely on any plugin, WebGL allows 3D object to interact not only with the DOM, but also with all the information available on the Internet.

Web-browsers are constantly improving their performances and functionalities, making them more and more complex. Renderers have to comply to requirements for professional workstations, such as color management, as well as mobile devices, with small memory footprint and low energy consumption. It is for sure one of the most complex multimedia softwares because of all the `HTML5` additionnal functionnalities.

WebGL was intrinsically designed to run on mobile devices. We knew from the beginning that the application we wanted to implement was very computational heavy, but we could not predict that we would reach the limits of PHP 5, Chrome (even the canary version) and Firefox 4.

Right now, we are able to stream huge 3D models with the help of WebGL, but the biggest ones made all tested browsers crash or freeze. Since we need to rely on the browser and its JavaScript interpreter to render the shapes, we don't have many ways to optimize the use of memory, and the solutions we tried are closer to hacking behaviour than properly documented methods. We noticed that only a handful of people asked web-browser to treat such a large amount of data and we hope our work can be the foundation of the Internet browsing in few years.

Chapter 3

Requirements analysis

This section presents all the researches made for this project and all the ideas which could potentially solve the problem of displaying large 3D models, while always keeping in mind the requirements.

3.1 Pre-requisites

3.1.1 User characteristics

Stakeholders: Museum curator

A Museum curator is working on the conservation of pieces of art. They are responsible for selecting and buying those objects, to describe them and their values and also agree together for loaning art objects between museums. Moreover, museum curators work involves the safety of the pieces of art in their responsibilities (including not only theft but also destruction by the visitor, or ambient aggression like light or oxygen), and performing corrective treatments if necessary.[34]

Because conservation is becoming more and more technical, conservators usually specialize in a particular type of object, such as paintings, sculptures, photographs, crockeries, etc. Others concentrate on artwork from a specific historical period.

More than 60 curators [27] are working in the eight departments of the Louvre museum in Paris. Specialists in their field, they are responsible of the international diffusion of the museum's collections. They are involved in an international symposia, lectures at the École du Louvre, the National Heritage Institute, restoring works of art at the Ecole Normale Superieure, universities,

selection boards and numerous scientific publications.

The recent improvements in 3D scanning technologies allowed the creation of a numerical database from the collection of the museum. This could help researchers to work on the 3D models without having to touch or moving the objects, and thus damage them.

3.1.2 Software system features

Main feature The main feature of our project will obviously be the 3D stream ability. This viewer will be able to establish a connection with a server which will provide the 3D models directly inside a rendering surface in the client viewer. The user will never have to handle files in order to see the model, and will be able to zoom and rotate it.

First, users would see the model rendering chunk by chunk and then would be able to interact with it or would have a first preview of the object while the full model is loading. This is what we call the pre-visualization process.

Requirements: This viewer would have several requirements. We not only need it to display the 3D object, but also to allow the user to interact with it. The most important requirement being the fact that large 3D files have to be handled correctly by the viewer without too many performance problems (of course, this aspect is very subjective and depends on which device renders the model).

Secondary features

1. *Annotations:* Each curators should be able to put some notes linked to the model by clicking on the current position. Those notes would be saved and available for each curators. There would also be a RSS feed on some selected models in order to know each time someone annotates it or publishes about it.

2. *Security:* Due to security reasons, the curators would need to identify themselves through a portal with a login and password in order to deny those scanned models files to unidentified people.

3. *Digital watermarking:* Copyright information would be embedded into a digital signal, in such a way that it would be difficult to remove. Therefore it would be possible to identify the source of the PLY files by executing a simple algorithm. If the flow of information is copied, then the copyright would be carried with it.

3.1.3 Use case

In this section will be detailed the use case of the system we would like create:

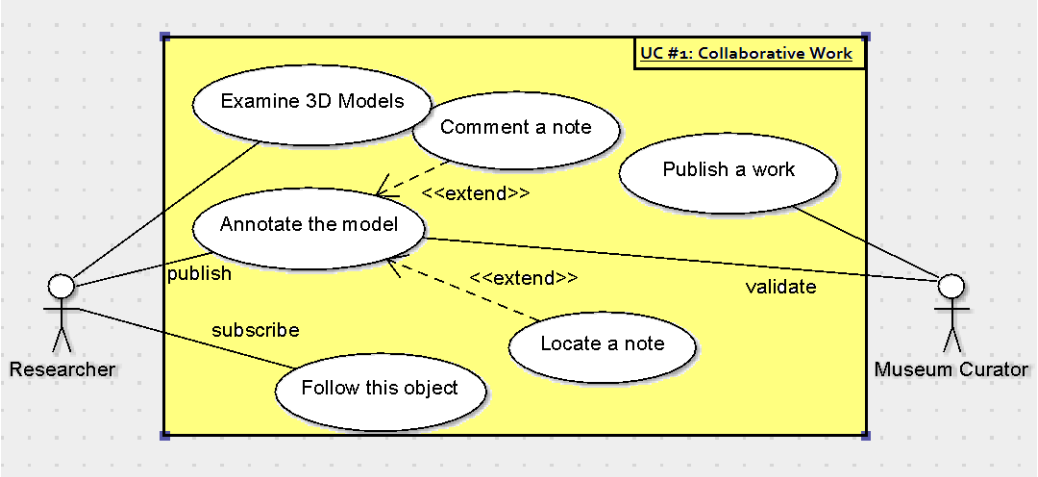

Figure 3.1: Use Case: Collaborative Work between curators and researchers

Collaborative Work between curators and researchers, Figure 3.1

1. Brief Description
   This use case shows us the interaction possible between the actors (curators, researchers) and the streaming of the 3D model.

2. Context
   A famous researcher is working on a publication involving a piece of art included in the collection of the Louvre museum. After asking for the permission, he logs himself in the system which allows him to interact with a 3D model of the object. He will also be able to access the annotations which have already been published by other researchers, or by the museum curator himself. Those annotations could be commented by others to make the process more interactive. There is also an option which allows the viewer to center the 3D model on a special part of the object which has been annotated. Finally, the researcher will be able to subscribe to the RSS feed of the publications made on this object to receive the latests information about it.

3. Flow of Events

   (a) Basic Flow

   The researcher receives an email notifying that his note has been published

      i. The streaming starts on the web page and the user sees the model rendering point by point.
      ii. The annotations and the optionnal comments corresponding to this object appear on the side of the streaming canvas.
      iii. A list of the museum curator's publications is also available at the bottom of the canvas.
      iv. The researcher interacts with the 3D object within the canvas (zooming, rotating).
      v. The user has something to publish and posts an annotation related to this piece of art.
      vi. The museum curator validates the researcher note.
      vii. The annotation will be updated on the feed of the other publications with a publishing date, author and optionnaly a location on the model.
      viii. The researcher receives an email notifying that his note has been published.

   (b) Alternative Flows

      i. The user can subscribe to the RSS feed of the related object to automatically receive the latest annotations and publications about it.
      ii. The researcher will also be notified if the museum curator decides to reject an annotation.
      iii. If the user is already subscribed to the RSS feed of an object, he will have instead the possibility to unsubscribed to it.

3.2 Candidate technologies

3.2.1 Technologies based on heavy clients

A Meshlab plugin MeshLab is an open source application used to process and edit 3D triangular meshes. It helps the processing of models coming

from 3D scanning, by providing functions such as cleaning and converting this kind of meshes, and many others [1]. A solution might be to download from the web the 3D file thanks to a plugin in Meshlab which would have to be created. But this could work for any other open source application similar to Meshlab. This technique would have the advantage of providing all the functionnalities of this software to the user, and only the features concerning the collaborative work would have to be added. However, the application would not provide any previsualization while the file is being downloaded and most of tools provided by Meshlab are not relevant for users defined in the analysis. Also, it would be very dependent of Meshlab, which means that after any architecture modification of the software (after an upgrade, ...), the plugin could potentially not be compatible with it anymore. Moreover, it would require to download and install Meshlab with its plugin, and for that you need the administrator privileges.

A whole new viewer Since many features of a conventional local 3D viewer are not relevant in the context of streaming 3D objects, we can also create a client-server system based on a viewer entirely developed from scratch. A programing language like Java is perfectly designed for this application: it provides reliable client-server architectures and tools, but also patterns to ease the development of such a system. Moreover, OpenGL capabilities are easily implementable with this language and one of its greatest advantages is the portability of the bytecode. Such a viewer would allow previsualization in opposition to the plugin based implementation. Whereas the Meshlab-plugin will have only to handle the communication with the server, this vision will ask the complete creation of the 3D model viewer. Even if this solution has many advantages, the C2RMF prefers a model visualization embedded in a web-browser.

3.2.2 Technologies displaying 3D shapes in a web browser

3.2.2.1 Flash

The Adobe Flash technology can describe a player, an application file format or a development environment. Sometimes described by Adobe as a web-standard[1], its implementation in a web-page requires a plug-in. The most commonly used player is closed source and provided by Adobe, the

[1] Creative Suite 4 Web Standard applications ReadMe

Adobe Flash Player, and can be easily found on most of computers browsing the web [5]. This player has some open-source alternatives like Gnash or Lightspark, but also commercial alternatives like Scaleform which handles hardware acceleration on GPU. Nevertheless, the most common flash player is the one provided by Adobe, even if it is only available in 32 bits and lacks hardware optimizations.

One of the great advantage of Flash applications is their great consistency. Because they are displayed in a plugin, on the same computer, whatever the browser, the animation displayed will be exactly the same. But, this will also be true on different operating systems or even architectures! This advantage is reinforced by the proportion of computers browsing the Internet able to handle SWF files.

One of the great criticises of the Flash development is that it requires an expensive license to allow developers to create applications based on it, which is totally false. Adobe provides an open-source compiler for Flex and ActionScript 3, the language used to create SWF [6] and Flash applications can be created without giving any money to Adobe. But the flash player is slow and resources consuming, especially on other platforms than Windows, and moreover on mobile devices, because no hardware acceleration is used to render graphics on this platform (using the Adobe Flash Player). Nevertheless, Molehill, codename of the 3D GPU-accelerated API for Flash, will speed up this technology but it is still in development. [24]

3.2.2.2 Java

Java is an object-oriented language allowing to produce byte-code able to run on several JVM[2], available on most platforms and operating system. Thanks to this portability, JVM can run in a web-browser to create applets to integrate in web page applications with complex functionality and server interaction, and for a long time, it was the only way to provide RIA[3]. Java has a strong community and is one of the most used computer language. [36]

Java development can be fast and easy thank to the community, but also with the number of libraries available. Moreover, thanks to a clear specification of its functionality, many open-source implementation os JVM can be found, and Java itself can be considered as a free software since the IcedTea project passed TCK test in june 2008. [32] Another advantage of this solution is that J2SE, the most common implementation of this language, has directly

[2] Java Virtual Machine
[3] Rich Internet Application

embedded OpenGL graphics routines. This means that handling 3D models in applets will be easy and will provide great performances. Finally, all the members of this project have extensive knowledge of this language and are comfortable creating a program with.

JVM requires an installation to be used as a web-browser plugin, but it is not the most pervasive one, and is losing market shares comparing to Adobe Flash or web-standard base RIA. [5, 12]

3.2.2.3 O3D

O3D is an API provided by Google to create rich and interactive 3D applications in the browser. At the beginning it was designed as a browser-plugin able to communicate directly with the client DirectX9 and PS 2.0 capable graphic card, but this implementation of the API is right now deprecated. [17] On May 07, 2010, the developers announced the decisions of stoping the plugin development and the will to create a WebGL JavaScript library. [26] This radical change was decided because of the JavaScript interpreters performance to request low-level API used by WebGL was considered too slow. Their reservations appears to be false and they decided to build their tool on the 3D standard. From our project point of view, O3D is right now, no more than another library to ease the usage of WebGL and that is how we will consider this technology. [18]

3.2.2.4 VRML / X3D

VRML[4] is a language presented at the world wide web conference of 1994, and was designed to represent 3D virtual scenes. We cannot consider the first version of this language as a programing one, but more as a markup or a modeling language, like HTML. The scene will be described and the interpreter plugin will display a scene, this characteristics forbid the creation of animations and interactivity, whereas the second version VRML97 can. [37] The creation of real applications was still impossible with this implementation.

The X3D, or Extensible 3D, is a superseded version of VRML providing better tools for animations or shape creation. The original VRML syntax, xml or binary can be interpreted by an X3D browser plug-in, and even if X3D allows to render finer scenes, and optimizations like oct-tree, it still doesn't provide any function to create real applications. [38]

[4] Virtual Reality Markup Language

Right now, X3D has become a WebGL library thanks to the Fraunhofer institute, called X3DOM. [8, 16] This library has extensive functionalities, and even if X3D was designed only to create interactive models, X3DOM allows to create RIA. As or O3D, we will consider this language more as another WebGL library than for the language itself.

3.2.2.5 WebGL

WebGL is a 3D computer graphics API, basing its calculations on JavaScript and its specifications on OpenGL ES 2.0, which as been standandized by the Khronos group. [22, 23] This API uses the canvas HTML5 element, which provides an area allowing the client to draw graphics, and rely on two well established technologies. As we described it in the introduction, JavaScript is very well established in the recent browsing habits and all the majors web-browser are doing their best to improve capabilities and speed of their JavaScript renderer. This fight can be easily noticed by the number of benchmarks aiming to evaluate JavaScript perfomance (like PeaceKeeper, SunSpider, V8 benchmark suite, Dromaeo, Kraken JavaScript benchmark...) Right now, this computation is done so fast that even for handling a low level graphic API, they are enough fast, and reduce the need to create a plugin to enable 3D capabilities for browser, according to the O3D creator. [26]

OpenGL ES 2.0 is a standard, also created by the Khronos group, and was designed to display graphics on mobile device like SmartPhones or Tablets. The graphics capabilities of this kind of device are good and can provide complex scenes without asking too much computational resources. This is great because it allows users with modest client to have access to this technology, like on the way to obsolescence computers or mobile devices for which OpenGL ES was designed. [21]

Since WebGL is a standard, and quite all majors web browsers creators have taken part in its specification, it has a great chance to become the reference in the next months (see O3D and X3D port to WebGL). In May 2011, this technology is only available in the latest release versions of Google Chrome and Firefox 4. Opera, Apple are planing to release it in few months or weeks. The only major browser that is not planing to support it is Internet Explorer, even if its JavaScript acceleration is graphic card based.

3.2.2.6 Image displaying, server rendering

Another technique to have 3D objects displayed in a browser is to base the rendering not on the client but on the server, and only sends to the client an

image file. To do so, the server requires a graphic card, which needs to be very powerful, and a complex AJAX, or RIA, client-server architecture has to be built. This is used to know exactly where the user is looking. The client sends this information to the server which has to render the image depending on the position and then send it back to the client. This operation has to be done so that the interaction with the object appears to be smooth. There is no standard for this technology, but you can find examples on the website of Real-scan[5].

Even if server rendering has very great performances, if the connection between the server and the client has a low latency and a high speed, we decided not to consider it for two reasons. First of all, it requires a deep knowledge in 3D programing, client-server architecture and RIA and members of the group do not have the time to acquire the technical skills and develop such a complex system. Second of all, the servers from the C2RMF, like most of the servers, don't have any graphic cards.

3.2.2.7 Decision based on the requirements

The first two technologies that have been rejected from our project are VRML/X3D and O3D, because they are not really supported anymore (they have been ported as WebGL libraries but do not exist as a plugin anymore). For this reason, we will only consider them as WebGL libraries and not as a technology anymore.

We also decided not to rely on plugins because they require administrator privileges to be installed, and also their performances are not really consistent depending on the platform or the operating system. This leads to stop considering Java and Flash (moreover, the most used flash player has very low performances on 3D rendering).

The last technology allowing to display 3D objects in a web-browser is the WebGL. The standard has been defined by the Khronos group [23] and is based on openGL ES 2.0 to allow its portability on most 3D graphic capable devices (like Android or iOS ones). The durability of this tool is assured not only by its standardization, but also by the fact that it is supported by all the major web-browsers, except Internet Explorer. It relies on very strong foundations. The choice of this technology appears to be, efficient, durable, portable and versatile.

[5] http://www.real-scan.com/

	Kept	Rejected
Java		•
Flash		•
O3D		•
VRML/X3D		•
WebGL	•	

Table 3.1: Technologies used to display 3D models in web-browsers

3.2.3 Streaming optimization

3.2.3.1 Bandwidth concerns

Introduction to streaming Streaming is the fact of constantly presenting some data to a user while the rest is being delivered. For example, in this way you can start watching a youtube video before it gets fully downloaded. It is usually applied to the medias which are distributed over networks such as the radio, the television or even books and audio CDs. The television on the Internet is a commonly streamed medium.

Bandwidth requirements In 2010, the "Broadband for all" in Europe ensures that all european governments support the idea that all citizens should have "affordable access to essential electronic communication services". [20] So far, only the UK, Finland, France and Spain have passed legislation to provide broadband to all citizens. Data speeds are different but are usually between 1Mb/s and 2Mb/s.

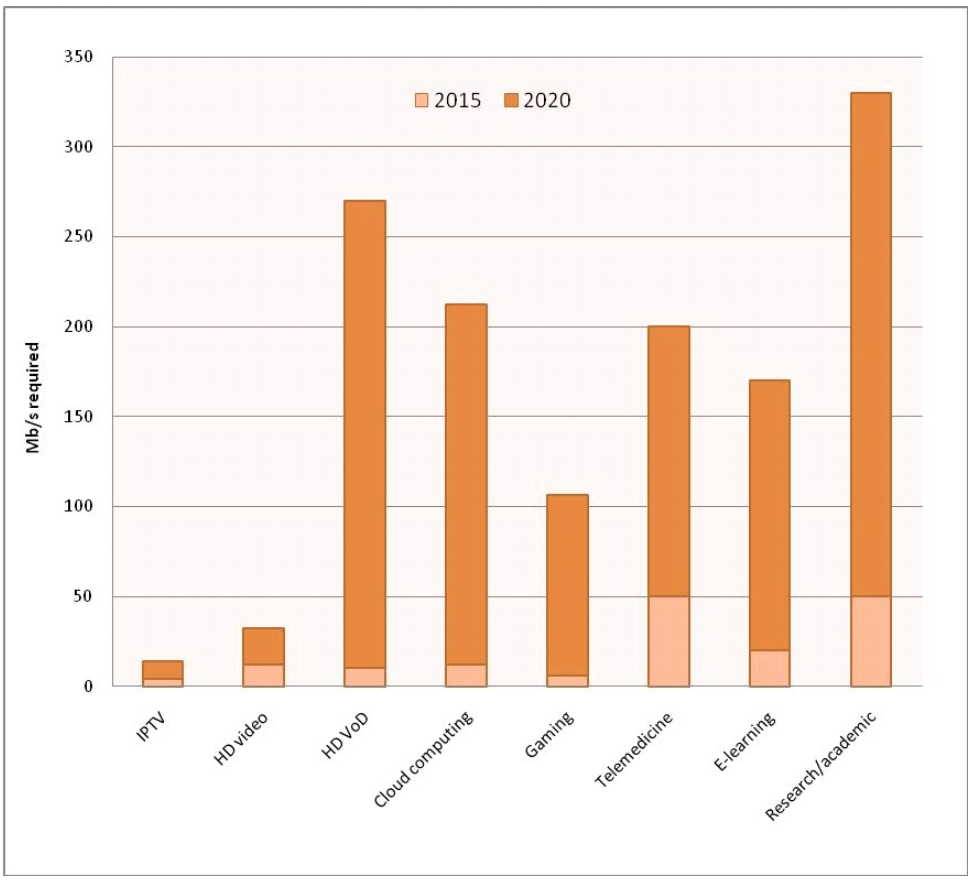

Figure 3.2: Anticipated bandwidth requirements, BuddeComm based on NTA data.

Regarding to the surveys, a broadband speed of **2.5 Mbps** or more is recommended for streaming movies (e.g. Youtube videos), and **10 Mbps** for High Definition content.

Streaming 3D There are two parameters that could cause some troubles by streaming 3D objects. First, the size of the 3D model. Those files are very large and their size can reach more than hundreds of mega bytes. Second, the user, depending on his bandwidth, could wait several minutes before being able to display the object. However, most of the ways to reduce this time would also reduce the quality of the model. Knowing that the stakeholders are curators who need all the details of the object, losing too much quality is not really an option. We can make their waiting time as pleasant as possible thanks to several features which will be introduced later in this report.

3.2.3.2 PLY file reduction

A PLY file reduction can be relevant to solve performance issues. In our case, the computers that the curators from the Louvre would use are powerful enough to visualize the full model at interactive rates. Which is why the full model has to be available at some point in the session. However, in order to potentially extend the users of our project for future purposes, it would be convenient to address the performance issues, due to the size of the file, so that computers which are not powerful enough can interact with the 3D object. This could be done by either creating a viewer which only handles points or reducing the number of faces and vertices contained in the PLY files.

A PLY file reduction can also be relevant to answer the bandwidth concerns. This technique can be applied to two different situations. First, it allows to reduce the file in order to provide the user with a lower resolution object which he can manipulate while a bigger resolution is loading on the background. Second, it is possible to only send the parts of the PLY files which are needed depending on what the user is looking at and the level of zoom.

Ellipsoids This technique uses overlapping ellipsoids instead of vertices and faces to represent a 3D object. This allows to have a good idea of its shape while the size of the file was quite lowered. [9]

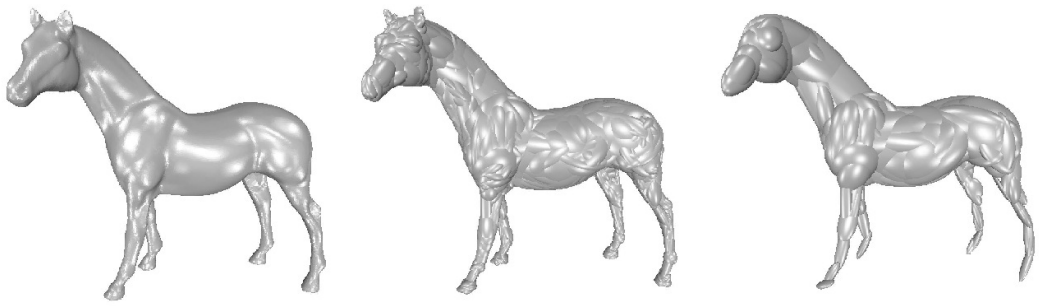

Figure 3.3: Comparison between ellipsoids and vertices [9](Fig.3)

From left to right: Original horse model (49000 vertices corresponding to 18500 bytes), ellipsoid approximations consisting of 400 and 100 ellipsoids respectively (corresponding to 14400 and 3600 bytes).

The problems with this technique are that it doesn't handle colors, the quality of the object is quite low even if it allows to send a lighter file. Also,

when using an average bandwidth, streaming all the vertices instead of trying to lower the number is done at a satisfying speed.

Billboard clouds The billboard clouds principle has been introduced in 2002 by four researchers [11]. This technique is an extreme 3D-models simplification using an optimization algorithm to reduce the input primitives to a set of planes. Each plane is rendered as a textured quad that replaces the geometry mapped by the plane. All the planes together approximate the shape of the original geometry. The difficulty of this simplification is to find a minimum set of planes, which results in a complex optimization problem. Since the frame rate has to stay above 60 frames per seconds for the movements and animations to be perceived smoothly, this method which requires "a very computationally expensive and time consuming task and can only be done in a preprocessing step" [25] should not fit our needs. It is important to note that the rendering is made on the server and requires a graphic card, which standard servers don't have. Moreover another difficulty resides in preserving visual accuracy of the simplified representation which is not fitting with the stakeholders needs.

Backface culling and octrees The backface culling determines if polygons from a model, are visible to the viewer or not. To do so it calculates their normal vector, which is the cross vector from two of the sides of the triangle in our case, and will not display the faces which are not oriented towards the camera. It is a relatively low computational process which is only relevant on the server side.

An octree is a tree data structure in which each node has exactly eight children. You can use them to partition a three dimensional space by subdividing it into eight smaller spaces. By doing so, it is possible to know which part of the octree are in the field of view of the camera and only send those.

These solutions help improving the rendering speed of a 3D object by not treating the polygons which are useless to the viewer, but require some calculations before being able to send any data.

Lossless data compression A lossless data compression is a compression which allows to recover the exact original data from the compressed data. The gzip compression is one of these and is handled by all modern web browsers[7]. It can be used to reduce the amount of data that has to be transferred to obtain the exact same result. However, it implies to lose some time to compress and then decompress the file and you need to wait to receive the

whole file to be able to extract the data. Because of this, slicing the file in smaller parts is relevant but it reduces the efficiency of the compression since its algorithm is mostly based on the redundancy of the bits from the file.

3.3 Additional information

3.3.1 Data storage

Different types of data have to be stored for our project. The first and most important of all is the PLY files which can weight several hundred mega bytes, for which we have the possibility to either store them in a database or as normal files which we can access through Apache. Then, we have to store all the names of the PLY files, their descriptions, and potentially all the annotations from the curators in a database.

3.3.2 Planning

The time allocated for our thesis project is pretty short and that is why we needed a planning to stay on track. As you may know, our project started at the beginning of February and ends with the delivery of the report at the end of may.
The chart (fig. 6 on Appendices) presents the major steps in our project timeline. As you can see, the Research and the Requirements analysis takes almost 50% of the global time allocated for this project. Then, the design and implementation parts take another full month. We also planned to do some improvements and maintenance after the delivery of the report.
The chart (fig. 5 on Appendices) introduces each tasks and tries to evaluate the ones which can be developed in parallel. Also we have been able to draw the critical path: the tasks which can not be delayed. We can also see on this well detailed timeline that our streaming application will be fully working around the second week of May.

Chapter 4

Design

This section provides an exhaustive list of WebGL libraries, each of them being analyzed to determine if they can fulfill the project needs. It also presents some choices made for the implementation of the project and includes a mock-up of the application.

4.1 WebGL

4.1.1 WebGL libraries benchmark

Since the number of WebGL vertices in a same object is natively limited by the library, because of a 16 bits index, which represents 65 536 vertices. This limitation exists because the Khronos Group wants WebGL to run exactly with the same specifications on all platforms. [19] Because we are handling a huge number of vertices and faces in our model, we need to find an API able to handle more than 65K vertices.

- C3DL, canvas 3D JavaScript Library | Website: *http://www.c3dl.org/*

  This library can't handle Collada models which go higher than 20MB (and will make Firefox 4 crash). [2]

- Copperlicht | Website: *http://www.ambiera.com/copperlicht/index.html*

  This library has very high performances but doesn't handle more than 65K vertices. Coppercube, one of the tools of Ambiera, allows to split huge models into smaller chunks [3]. But this tool is not free and we need to pay to obtain the source code of this library, even if it is free of use. Moreover, the functionality of adding color to vertices is not provided.

- Curve 3D | Website: *http://codi.st/pages/curve3d/*

  Curve 3D is a 3D engine, only written in JavaScript, which does not rely on the acceleration capabilities of the graphic card to render the shapes it designs, but works in all major current browsers (Firefox 3.6+, IE8+, Chrome 5+, Opera 10.5+). Since all the calculations are handled by the CPU and through an interpreter, the performances are pretty low.

- CubicVR.js | Website: *http://www.cubicvr.org/*

  CubicVR.js is a rewriting of CubicVR C++ 3D engine in JavaScript / WebGL, and even if performances and capabilities are interesting for our project, the number of vertices is limited to 65K vertices by chunk.

- GammaJS | Website: *http://gammajs.org/*

  The purpose of GammaJS is to create 2D models and provide the tools to interact with them in order to produce a game. Since we want to interact with our models with 3 axes and 2 rotation angles, this library cannot fulfill our needs.

- GTW | Website: *http://blog.graphtech.co.il/gtw-—rich-user-interface-library-for-web-applications/*

  This is not really a library but it provides interactive elements to improve the GUI of RIA. For this reason, it cannot be adapted to our needs.

- JS3D | Website: *http://www.wxs.ca/js3d/coordexample.html*

  Like Curve3D, it is not a WebGL library, but a pure JavaScript one.

- Kuda | Website: *http://code.google.com/p/kuda/*

  Kuda is a "library and editor for authoring interactive 3D content for the web", which is a purpose entirely different from the one this project is trying to achieve.

- OSG.js | Website: *http://osgjs.org/*

  OSG.js is a low level library that doesn't handle faces and vertices buffer splitting, and for this reason is limited to 65K vertices by facemesh. Moreover the documentation of this project is quite inexistent.

- PhyloGL | Website: *http://senchalabs.github.com/philogl/*

  PhiloGL is a WebGL Framework created to offer ways to visualize data in 3D, create games and beautiful creations. This library does not

provide any easy way to handle faces and does not handle more than 65K vertices.

- Pre3D | Website: *http://deanm.github.com/pre3d/*

  Like Curve3D, it is not a WebGL library, but a pure JavaScript one.

- Processing.js | Website: *http://processingjs.org/*

  Processing.js is a rewriting of Processing visual programming language (based on Java) in order to make it work on the web and which integrates WebGL for 3D capabilities. This library does not provide an easy way to create faces and its performances are lower than genuine WebGL optimized library [31].

- O3D | Website: *http://code.google.com/p/o3d/*

  At the beginning, O3D was an external plug-in and even if the O3D development team knew the creation of WebGL, they thought JavaScript will not be fast enough to handle computations of a low level library. Few months later, they realized that WebGL could allow them to do what they wanted to and decided to rewrite all the library into the new standard. Even if it is a Google project, the official, and of course the non official, documentation is not reliable because we cannot be entirely sure if it is for the plug-in or the WebGL based version.

- SceneJS | Website: *http://scenejs.org/*

  SceneJS is a JavaScript / JSON based API created in order to manipulate elements of a 3D scene in WebGL. Thanks to the JSON architecture and functionalities, creation, modification and destruction of a scene or an element is very easy and modifications can occur directly on the model or the way to displaying it. With this API, models are limited to 65K vertices.

- StormEngineC | Website: *http://code.google.com/p/stormenginec/*

  This WebGL library does not provide easy ways to create faces, it doesn't handle more than 65K faces and is halted. Moreover, the lack of documentation is very perceptible.

- SpiderGL | Website: *http://spidergl.org/*

  SpiderGL is a library which is still in alpha development. However the last update on their Sourceforge was made the 29 january 2010. Moreover,

it does not support more than 65K vertices and some of their examples are buggy.

- ThreeD Library / TDL | Site: *http://code.google.com/p/threedlibrary/*

  This is a low-level library which "focuses on speed of rendering rather than ease of use" but which cannot be downloaded and does not have any documentation...

- Three.js | Website: *https://github.com/mrdoob/three.js/*

  Three.js focuses in creating a 2D and 3D engine, very easy to learn and handle. Of course, the 3D part relies on WebGL but this library limits its mesh to 65K vertices.

- WebGLU | Website: *https://github.com/OneGeek/WebGLU*

  WebGLU is described by its creator as a set of low-level tools designed to have the best performances. Because of that, it does not handle colored faces and vertices buffer splitting.

- X3DOM | Website: *http://www.x3dom.org/*

  X3DOM is a WebGL library created by the Fraunhofer institute in order to give the ability to render X3D objects in a WebGL compatible browser without the need of installing any plug-in. It provides the same capabilities of the original implementation but in a more extensible way, because it is based on standards and can interact with the DOM. It supports the splitting of an object in chunks of 65K vertices and provides a primitive interaction with the displayed model.

- XB Pointstream | Website: *https://github.com/asalga/XB-PointStream*

  This is not really a library, but more a point cloud displayer which we modified and debugged to fit our needs. The purpose of XBPS is to take a point cloud, split it if it has more vertices than 65K and quickly display points. But it only handles points and the face implementation is not planned.

4.1.1.1 Decision

After considering all the advantages and drawbacks, we decided to use two WebGL libraries. The first one being XB-PointStream for its very good performances on huge files, since this library only focuses on displaying clouds of magnified points, and because it displays the data as soon as it receives it.

The other library being X3DOM. The purpose of X3D and X3DOM, which is the WebGL version of X3D, is to create animated 3D scenes and was never created to handle huge models. Considering the size of objects manipulated in this project, X3DOM performances appear to be acceptable and its functionalities fit our needs.

4.2 Database / File

The two options available were to either use a database or keep the PLY file in its original format to display it in the 3D viewer. Both these solutions have their advantages and drawbacks concerning this project, and those are listed below:

Database

1. *Advantages:* The database has the capacity to easily apply some algorithms on the PLY file to optimise the data flow (ex: backface culling, billboard clouds...). It can also provide us different statistics, like the number of times a table has been accessed or how many times a PLY file has been requested. Moreover, it allows us to compress the data before sending it to the client, and the browser will handle by itself the decompression.

2. *Drawbacks:* Using the database would imply the need to generate a PLY file to display before being able to send any data, which would therefore increase the total time needed to obtain the full model.

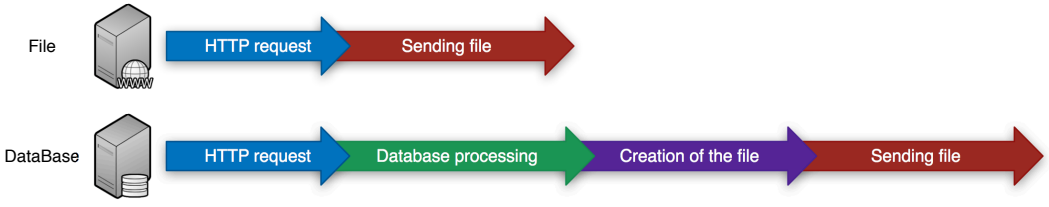

Figure 4.1: Request comparison Database / File

File

1. *Advantages:* Just like the database, the fact to simply store the data in a PLY file would also allow us to compress the data before sending

it, thanks to the `mod_deflate` provided by Apache [7]. But the main advantage of using the PLY format would be the access speed to the data, which is higher than with the database.

2. *Drawback:* The major drawback would be the difficulty to apply any algorithm on the data to optimise the data flow.

Decision Whatever the solution, the network would have to send an equal amount of data in around the same amount of time, so this parameter was not part of the decision making. However, the server would be much more requested and more space would probably be required with the creation of SQL *ids*, if the solution based on the database was implemented. For these reasons and because of the higher access speed to the data, we decided to simply store the data in files.

4.3 Mock-ups

These mock-ups describe a full-feature system that the C2RMF plans to create. Our project has the only aim to stream 3D objects in a web-browser, which is why some of the presented functions will never be discussed in this report. Nevertheless, some choices we have made were clearly oriented to ease implementation of these functionalities in the future.

In the following pictures, ways to comment, interact or annotate the model are described, as well as a method to contact the curator in order to obtain further informations on the object.

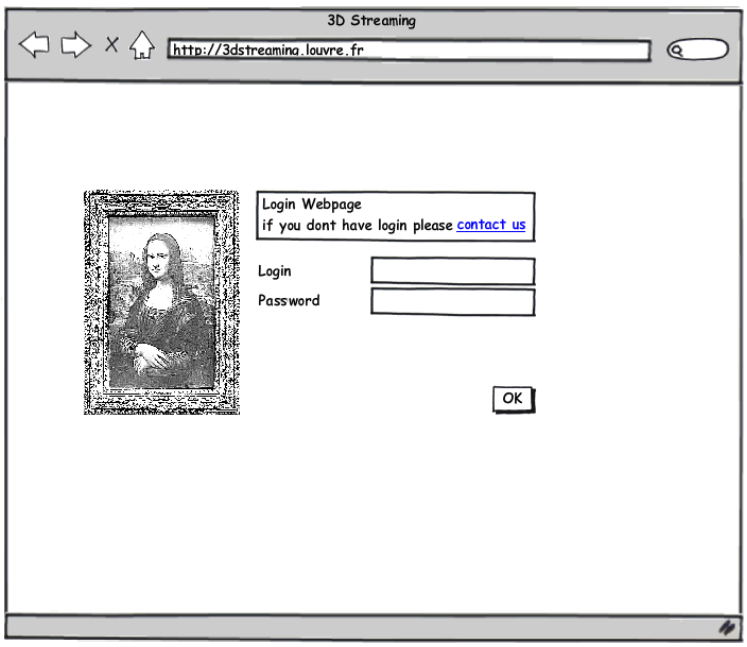

Figure 4.2: Mockup(1): Login page [14]

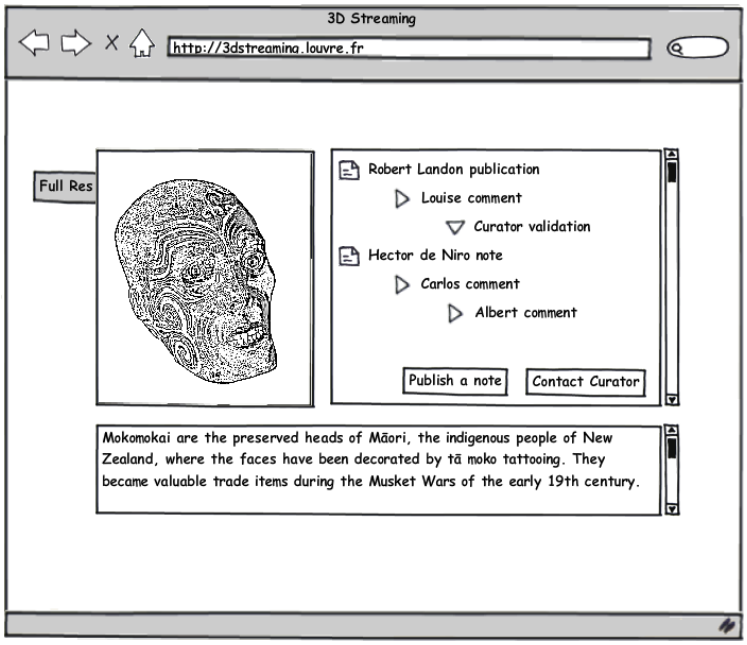

Figure 4.3: Mockup(2): 3D model main page [14]

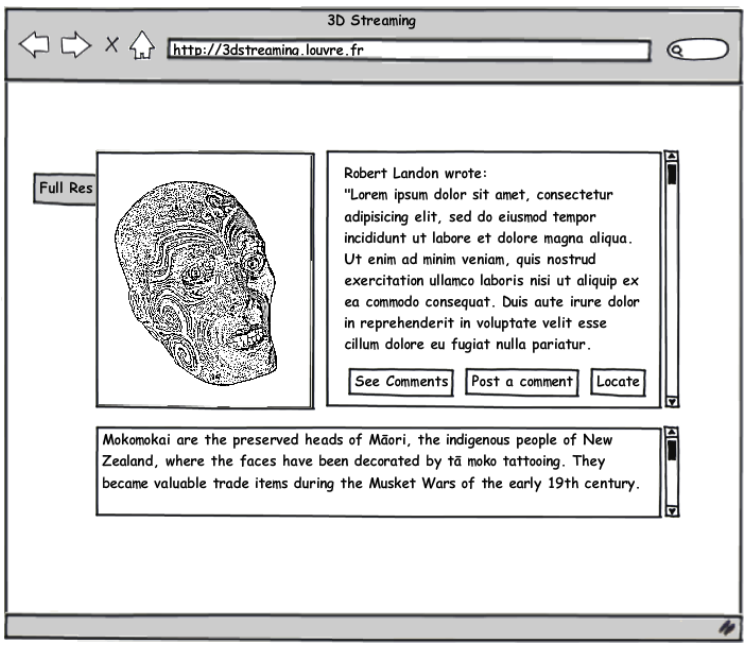

Figure 4.4: Mockup(3): 3D model annotation [14]

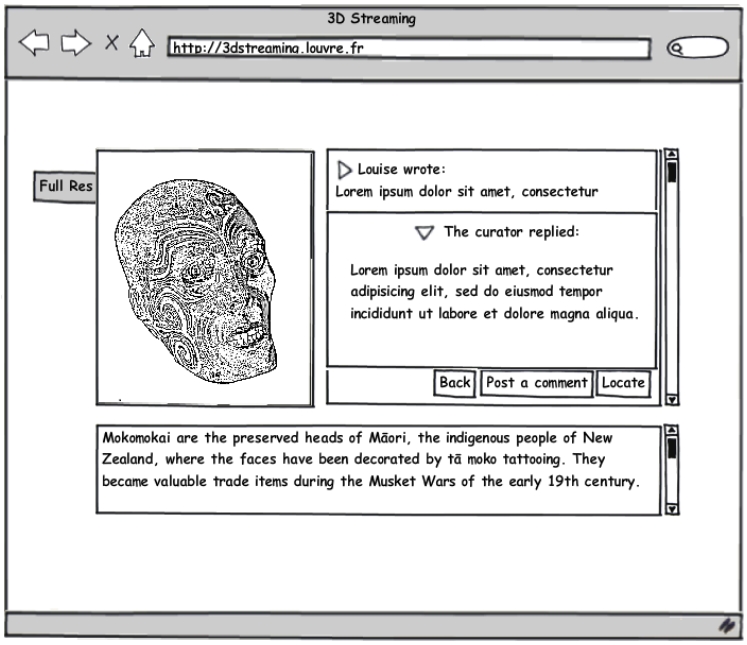

Figure 4.5: Mockup(4): 3D model comments [14]

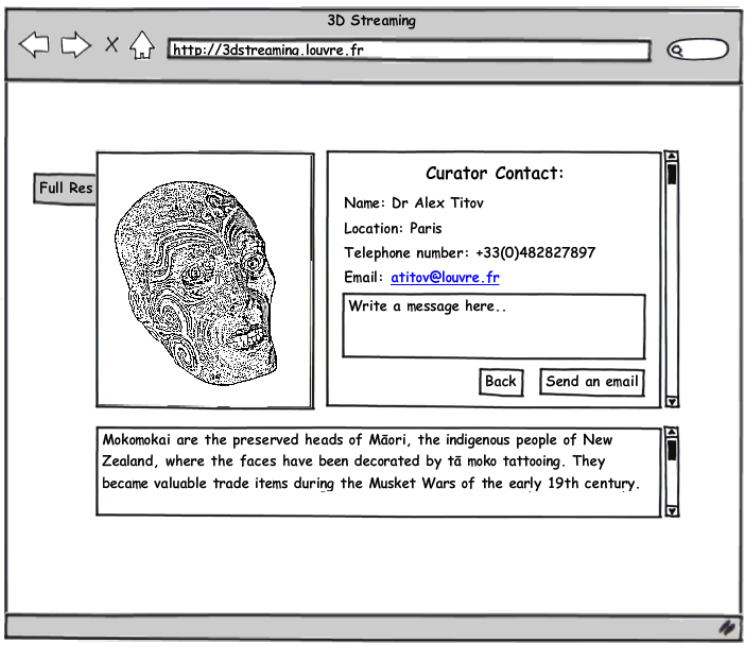

Figure 4.6: Mockup(5): 3D model curator contact [14]

Chapter 5

Implementation

5.1 Server side

This part will explain the different server-side implementations with their own characteristics and limitations. The first part will introduce the testing server settings, running at our place, which is mostly used for developing and demonstration purposes. Then we will describe the Louvre server configuration which is quite different than the one used for development. The approach used for this thesis is close to the management in professional projects: all development, testing and documentation have been done on a testing setup before pushing it "live" to the C2RMF servers.

Figure 5.1: Asus EEEPC 701 4G

- Test server: Asus EeePC 701 4G[1]
  - *Configuration:* the server is running on an Asus EeePC which has a very light configuration but fits the project needs. It is powered by an Intel Celeron M353 with a 900 MHz frequency underclocked to 630 MHz, has 1 GB of DDR-2 RAM and a 4 GB SSD for stockage. It also comes with a Wi-Fi b/g Atheros card and an Ethernet 10/100 Mbps. As you can see, it has some limitations such as the lack of RAM and storage, moreover, the processor is quite slow. But it was still more than needed to implement and test the prototype versions.
  - *Installation:* LAMP, which stands for Linux, Apache Http Server, MySQL and Perl/PHP/Python, has been installed on that computer and offers many advantages: firstly, it is free of cost configuration since it is an open source software and it fits the project needs. Secondly, since the time allocated for the project is quite short, we focus on an easy to install and deploy installation, and on something which the team members know well. Moreover, it allows us to program locally before having the application available online. Many softwares where used:
    - Ubuntu 10.10 Server Edition
    - Apache 2.2.16
    - MySQL 5.1
    - PHP 5
    - phpMyAdmin 3.3.7
  - *Bandwidth:* Since the test server is located at home, the bandwidth is the one provided by the house keeper. After running a few bandwidth tests, here is an average of its global performances. It has a ping between 5 and 10ms and a very good upload speed (around 70Mbps)[2]. In other words, the server has the proper bandwidth to stream huge 3D files.

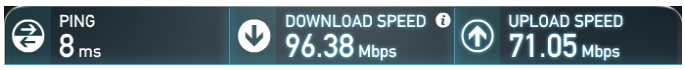

Figure 5.2: Screen capture of test server speed test

[1] http://www.notebookcheck.net/Asus-Eee-PC-701-4G.6745.0.html
[2] Ping, download speed and upload speed from http://speedtest.net

- *RENATER CNRS network, Louvre configuration:* The Louvre is plugged to RENATER which is a french network composed by a metropolitan infrastructure and international connections to broadband. RENATER is handling more than 1000 establishments engaged in the areas of Research, Technology and Education. The characteristics of this network [4] are:
  - An architecture based mostly on dark fiber
  - An architecture with links to 10 Gb/s on almost all points of network presence
  - An opportunity to respond to the needs of very high speeds of large research projects by establishing optical paths from end to end between the points of presence

  Moreover RENATER is interconnected to other European research networks and the U.S. via the European network GÉANT which is the largest structure of its kind.

5.2 Client side

5.2.1 File preparation

Like it was previously said in this report, the WebGL cannot handle 3D models with more than 65K vertices, even though some libraries can automatically split models if needed. In order not to be limited to libraries which provide this feature and to improve performances of the ones which are able to divide models in smaller chunks, a program was implemented to split PLY files. Since the 3D model will already go through a program to be sliced, we might as well format the data so that it is easier to use it for the viewer. Here is a quick reminder[10] of what a PLY file looks like:

```
ply
format ascii 1.0
element vertex 8
property float32 x
property float32 y
property float32 z
property uchar red
property uchar green
property uchar blue
element face 6
property list uint8 int32 vertex_index
end_header
0 0 0 255 0 0
0 0 1 255 0 0
0 1 1 255 0 0
0 1 0 255 0 0
1 0 0 255 0 0
1 0 1 255 0 0
1 1 1 255 0 0
1 1 0 255 0 0
4 0 1 2 3
4 7 6 5 4
4 0 4 5 1
4 1 5 6 2
4 2 6 7 3
4 3 7 4 0
```

One model will be split into several smaller models which are independent from each other, but when put together shape the original model. The program allows to split the models in two different ways.

5.2.1.1 Javascript arrays

The first way is by splitting the file into javascript arrays. An array of vertices will never have more than 65K vertices to respect the WebGL limit. The main advantage of this technique is that when the javascript file containing the array is included, the data is directly available for the viewer. Here is an example, even though the actual arrays contain way more data:

```
var vertices0 = new Array("-57.4653", "-41.9127", "-23.114",
"-57.6114", "-41.6757", "-22.6852",
"-57.5898", "-41.9012", "-23.5612",
"-58.4987", "-37.3321", "-18.2672",
"-58.6951", "-37.086", "-18.3145");

var vertices1 = ...
etc.

var faces0 = new Array("0", "1", "2" , "-1",
"3", "4", "5", "-1",
"6", "4", "3", "-1",
"7", "8", "9", "-1",
"10", "8", "7", "-1");

var faces1 = ...
etc.

var colors0 = new Array("0.27450982", "0.27450982", "0.27450982",
"0.047058824", "0.047058824", "0.047058824",
"0.15294118", "0.15294118", "0.15294118",
"0.32941177", "0.050980393", "0.11372549",
"0.31764707", "0.019607844", "0.101960786");

var colors0 = ...
etc.
```

The main problem of this method is that some browsers cannot handle that much data for an array declaration, such as Firefox. To solve this, slicing the file in even smaller chunks would work, but the huge number of HTTP requests addressed to the server would decrease the performances of data transmission. Moreover, the method used to inject the JavaScript code in the page is not compliant to good practices commonly accepted.

5.2.1.2 File chunks

The second way is splitting the file into small chunks which are already formatted for the viewer to use them. Here is an example with only a small amount of data formatted for X3DOM:

```
File: 0vertices
3299 68.5992 12.0763 -3.60006 67.59 11.7546 -1.09796 66.6631
11.5234 -0.137346 66.1669 11.5951 0.219926 66.0071 11.3723
2.0417 65.4909 11.0471 2.66579 65.1336 25.7053...

File: 1vertices
...

File: 0faces
0 1 2 -1 3 4 5 -1 6 4 3 -1 7 8 9 -1 10 8 7 -1 11 8 10 -1 12
13 14 -1 15 14 13 -1 14 15 16 -1 16 17 18 -1 18 19 20 -1 21
20 19 -1 22 23 24 -1 23 22 25 -1 26 25 22 -1 27 25 26 -1 27
28 29 -1...

File: 1faces
...

File: 0colors
0.27450982 0.27450982 0.27450982 0.047058824 0.047058824
0.047058824 0.15294118 0.15294118 0.15294118 0.32941177
0.050980393 0.11372549 0.31764707 0.019607844...

File: 1colors
...
```

The main advantage of this technique is that, by simply reading the files in JavaScript, it is possible to know when all the data from the file was loaded to then inject it into the DOM.

5.2.2 Viewer

5.2.2.1 XB-PointStream PLY parser

This method is directly based on the original PLY file, saved in ASCII, and stored without any treatment on the server. The client requests the file with AJAX methods, processes it chunk by chunks, and displays points while the data is received.

To work, this method relies on the fact that in the PLY file, the vertices information (color and position) are stored before the faces and that they are independent from other vertices or any other data: a line in the vertices part

only provides the information to display a vertex and nothing more. Moreover, the number of vertices which is included in the header allows to exactly know which part of the file is composed of vertices and which part is composed of faces.

Because of the use of AJAX objects, downloading and treating the data does not sluggish the webpage: the user is still able to interact with the page (scroll, type, click...) but also with the 3D model in the canvas (zoom and turn it).

XB-PointStream is already able to treat many point cloud files (asc, psi and pts), but we created a parser in order to make this library able to process ply files, even if only one part of a ply file can be considered as a point cloud. This library, based on the file extension, choses the right parser: we just need to provide the complete path of the file in order to make the application work. The PLY parser will find the needed informations in the header and process the data in order to display points.

The download time of the considered point cloud is quite long compared to the time needed to process the data by the client's browser, so the information will be treated and displayed chunk by chunk. The model will appear on the client screen by group of points which can easily be either randomized or ordered. This method gives the user a true feeling of receiving the model in a streaming way.

But even if this method is very simple to implement and has several advantages, it has also huge drawbacks. The first one is that it does not handle faces, and because of that they have to be ignored once all the vertices have been displayed. The other reason is that a lot of unneeded data is transferred in the process. For instance, MeshLab adds at the end of every vertex the information for the alpha channel, which is not used at all in our case. For these reasons, this method has been rejected in our final implementation.

5.2.2.2 X3DOM with JavaScript arrays

X3DOM allows to manipulate X3D formatted information in the DOM of a webpage with the help of JavaScript tools (the same which call the WebGL functions in order to display 3D in canvas). The information is placed in a non-standardized tag call `<X3D> ... </X3D>` like in this example:

```
<x3d width="500px" height="400px">
  <scene>
    <shape>
      <appearance>
        <material diffuseColor='red'></material>
      </appearance>
      <box></box>
    </shape>
  </scene>
</x3d>
```

This is a very simple syntax compared to the code which would have to be written in WebGL in order to display the same scene (a red cube at the center of a `<canvas>` of 500*400px). In the end, the object won't be displayed between the X3D tag, but X3DOM will create a context in the webpage to render the "world" in a proper way.

The tag `<IndexedFaceSet coordIndex='data'/> </IndexedFaceSet>` provided by X3DOM is very useful in our context since it uses all the data inserted in a usual PLY file, only the format is different. We can slice the original file in smaller PLY files and format them in order to properly put them into these tags. The small chunks provided will progressively be loaded, giving to the user an impression of streaming, keeping him from waiting for the full file to be downloaded.

The first version of this method was to format the files in JavaScript arrays as it is described in the subsection 5.2.1.1. It worked properly on small models, but there were some memory management problems with higher definition files. We tried to release the memory by destroying the arrays (removing the include of the javascript file), but the implemented JavaScript garbage collector of the browser would still not free the memory. This only works if the object is destroyed right after its creation, in other words by keeping the browser from starting to use it. In the case of our biggest models, the amount of memory required made all tested browsers crash. Moreover, the method used to implement these JavaScript objects into the DOM is not very reliable and can lead to malfunctions. For all these reasons, this method has been rejected in our final implementation.

5.2.2.3 X3DOM with direct DOM injection

This method is similar to the previous one: the original PLY file is sliced in many files, like it was described in 5.2.1.2. These files are formatted to be directly injected in the DOM, and more particularly into the <X3D> tags of X3DOM.

Thanks to XMLHttpRequest objects, one of the most famous elments of AJAX, the needed information is downloaded while at the same time allowing the user to interact with the page, and of course the rendering surface. Since all the needed information for one chunk has been received, It is directly injected into the DOM and rendered by X3DOM. The model is displayed chunk by chunks, exactly like in the method described in the subsection 5.2.2.2, however this method is more robust and well known. Moreover, the amount of memory required is lower.

Nevertheless, this method is still very stressful for the browser and the biggest models cannot be handled by all tested bowsers which cashed after receiving and treating too much information. Since this method is the one which provided the best results, it will be benchmarked in next section.

Chapter 6

Performance test

In this section of the report, the global performances of our implementation will be presented.

First, we will study the persistency of the implementation between several browsers. Then, the time to render the model between two relatively different configuration running on the same OS will be examined. In addition, two distinct models of different size will be tested on the same machine.

6.1 Comparison between several browsers

For the comparison between several browsers, we took a pool of three major browser vendors: Windows (Internet Explorer 9), Mozilla (Firefox 4) and Google (Chrome). By far the most used browsers according to statistics from w3schools.com which prints browser statistics month by month. It shows that Firefox is in the lead with about 42%, following by IE and Chrome which are both around 25% in 2011. Before starting the tests, it is interesting to note that only Google and Mozilla are members of the Khronos consortium's WebGL Working Group working with many 3D graphics developers.

6.1.1 Firefox 4

Firefox just came up with a new release: Firefox 4. The previous release of Firefox did not support WebGL.

In this latest version of Firefox, WebGL is fully supported and is running smoothly on our machines. There is a screenshot of the test during the streaming:

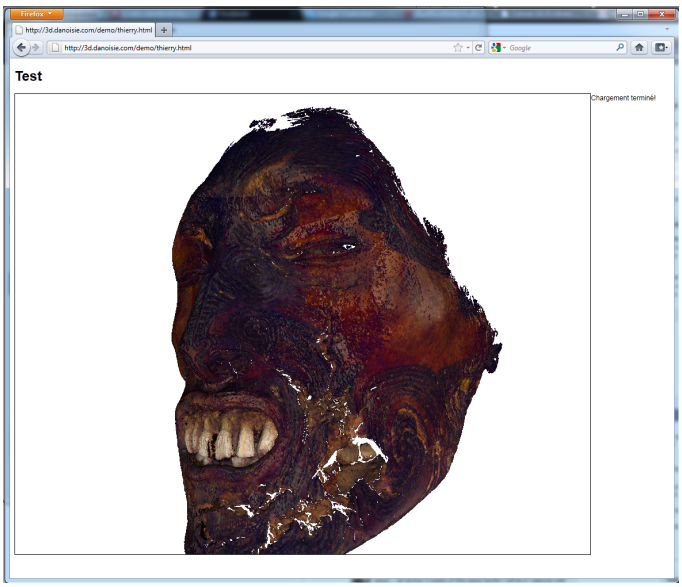

Figure 6.1: Firefox WebGL support testing

6.1.2 Internet Explorer 9

As expected, since Microsoft is not a member of the Khronos consortium's WebGL group, it does not support WebGL at the moment and it is far from being certain that it will. Here are an example of the streaming test window:

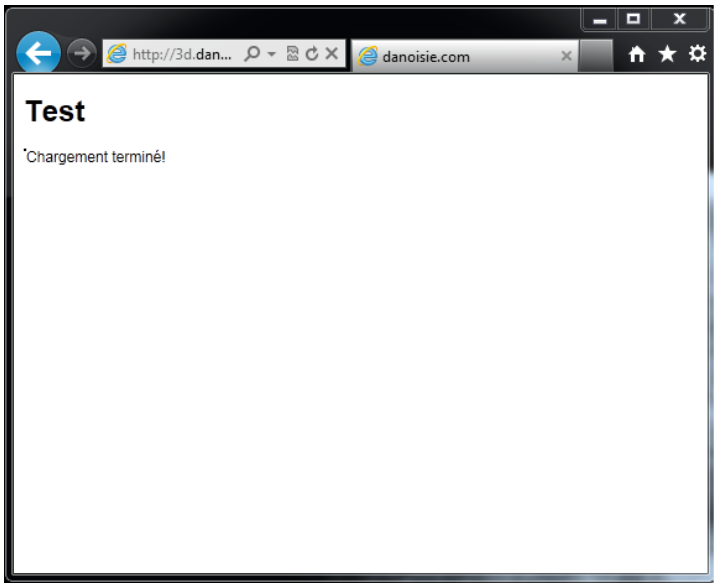

Figure 6.2: IE 9 WebGL support testing

6.1.3 Google Chrome

Chrome was the first browser to support WebGL. Also, there was no problem in any tests of the 3D streaming. Thus, Chrome has been chosen for the two following tests as the reference browser.

6.2 Comparison between two different configurations

6.2.1 Testing machine configurations

The first machine is a heavy powerful Alienware laptop. It runs on Windows 7, has an Intel Core2 Quad CPU Q9000 at 2.00GHz. Also coming with 4096MB DD2 RAM, two NVIDIA GeForce GTX 260M graphic cards and around 500Go 7200RPM hard-drive. The second machine is a less gifted one: Samsung N510. This small netbook also runs on Windows 7. Its CPU is an Intel Atom N270 at 1.60GHz, with a NVIDIA ION LE graphic card, about 1024MB RAM and a 120Go 5400RPM hard-drive. As you can imagine, those two configurations are different enough to allow us to do some comparison between the performance of the 3D streaming project on a gaming machine and a small netbook. Another computer will be used in our detailed benchmark, a MacBook Pro running Mac OS X 10.6.7 with a Intel Core i5 2.53 GHz with 8 GB of memory, a 7200 RPM 500 GB hard-drive and using two graphic cards, an Intel HD Graphics and a NVIDIA GeForce GT 330M with 256 MB of memory.

The following part describe two different behaviors of our application in two different configurations, when the system is able to handle this amount of memory and when it is not. Extensive benchmark will be provided in section 6.4.

6.2.2 Results

Here are the results of this particular testing: The first one has been tested on the Alienware laptop and, as you can see, it uses almost all the memory available (85%) which slowly raises as the streaming progresses. Also you can notice that the usage of the UC is moving between 5% and 25%. Therefore the streaming of that model is very smooth and we can easily interact with the model without any inconvenience.

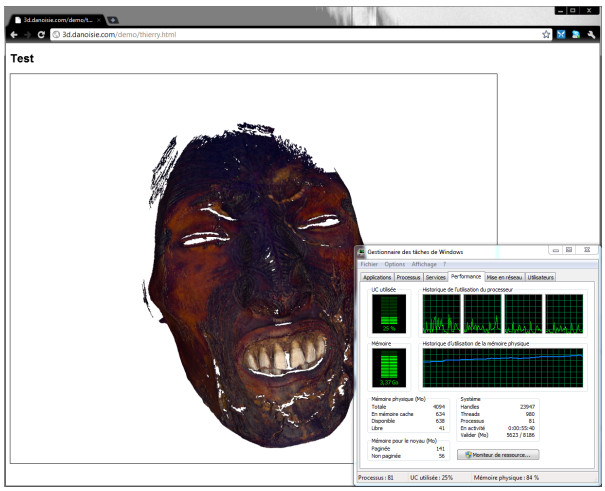

Figure 6.3: Alienware laptop streaming performance test

The second one has been evaluated on the Samsung netbook. The results are way more critical: The memory usage is almost saturated with 92% used and is also slowly increasing. The real problem on this machine was the UC usage which is very unstable between 30% and 100%. Which caused the streaming process to be slug and also the whole machine to respond very slowly.

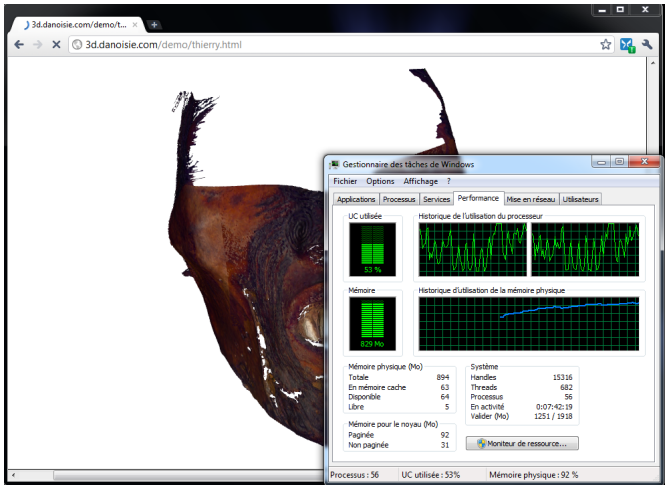

Figure 6.4: Samsung netbook streaming performance test

Therefore, as the quantity of the memory will impact on the time of the global streaming process, the computation capability of a client will impact on how the machine will react to it.

6.3 Comparison with two different model sizes

The first file is a full resolution model with four million of faces, without any process to decrease the number of vertices and faces. The only treatment applied is to divide it into small chunks (195 pieces which represent 585 files). The second file is the same model but the "quadric edge collapse decimation" in Meshlab was applied to it in order to have a file with 2 millions of vertices and which weights around 90 MB instead of 180 MB. The second file can still be considered as a huge file. But as you will see in the following results, browsers are handling those in a very different manner.

6.4 Complete benchmark results

The following tables show the difference in rendering medium and huge models in two different systems (Mac OS X & Windows 7), with three different platforms (the testing machine configurations described in subsection 6.2.1). With small files (around one million of faces and below) our application render the streamed model on every browser. With higher definition, the situation depends mostly on which browser is used to render the model, and also on the memory available on the computer, the processor always have a reasonable occupation. If the streaming was not completed, it was always because of a bad memory management by the browser.

	Memory used by process	Free memory	Streaming completion
Samsung	564 MB	63 MB	✗, 64 on 96 chunks loaded
Alienware	1.1 GB	1.7 GB	✗, 71 on 96 chunks loaded
Mac	698 MB	4.90 GB	✗, 65 on 96 chunks loaded

Table 6.1: Chrome medium resolution model

	Memory used by process	Free memory	Streaming completion
Samsung	527 MB	68 MB	✗, 63 on 196 chunks loaded
Alienware	1.0 GB	1.7 GB	✗, 65 on 196 chunks loaded
Mac	700 MB	4.91 GB	✗, 63 on 96 chunks loaded

Table 6.2: Chrome high resolution model

	Memory used by process	Free memory	Streaming completion
Samsung	597 MB	52 MB	✓
Alienware	2.7 GB	237 MB	✓
Mac	2.61 GB	3.85 GB	✓

Table 6.3: Firefox 4 medium resolution model

	Memory used by process	Free memory	Streaming completion
Samsung	602 MB	64 MB	✗, 97 on 196 chunks loaded
Alienware	2.8 GB	122 MB	✗, 102 on 196 chunks loaded
Mac	4.08 GB	2.63 GB	✗, 162 on 196 chunks loaded

Table 6.4: Firefox 4 high resolution model

Conclusion

When we started this project, WebGL was not standardized by the Khronos group yet, and was not supported by any major web-browser in their final version: the technology was at its very beginning. Of course our knowledge in web 3D programming has greatly increased, but also in classical 3D programming, since nobody in the team was familiar with such a technology. We were complete beginners, who did not know anything about vertices, faces or shader but we could rely on an associated extensive knowledge in web-programming. Some of us were confident in database structure, others in PHP or JavaScript, client-server interaction and server administration and we all knew very well Java.

We were expecting a success because we knew that the skills of the team covered all the web or streaming part of this project, but we did not expect the problems which we faced because of handling files of this size. The first version of the PLY splitter, made in PHP, was unable to provide us a simple chunk because of memory problems. As you have seen in the benchmark, even in the method which needs less memory in the browser, all of them are unable to handle the models with a too high resolution. We really think that browser rendering engines have to be improved to handle that much data, which is very likely to become a fact in a few months.

With this project, our skills have not only increased in 3D programming, but also in memory management, server and client side and in technology watch because WebGL was really a new one. We are glad the Aalborg University gave us the opportunity to work on a technology in development and we hope this project convinced you that WebGL will play a major role on the Internet in the days to come.

Bibliography

[1] Meshlab. Available from: http://meshlab.sourceforge.net/.

[2] C3dl stability issue on firefox, May 2011. Available from: http://www.c3dl.org/index.php/tutorials/tutorial-4-models/.

[3] Copperlicht support of more than 65k vertices, May 2011. Available from: http://www.ambiera.com/forum.php?t=1734.

[4] National telecommunication network for technology, education and research, May 2011. Available from: http://www.renater.fr/?lang=en.

[5] Adobe. Flash player penetration [online]. May 2011. Available from: http://www.adobe.com/products/player_census/flashplayer/.

[6] Adobe. Open source flex sdk, May 2011. Available from: http://opensource.adobe.com/wiki/display/flexsdk/Flex+SDK;jsessionid=994D5D961333BD2704CAFB0359011FA3.

[7] Apache. Apache module mod_deflate, 2011. Available from: http://httpd.apache.org/docs/2.2/mod/mod_deflate.html.

[8] Johannes Behr. X3dom - a dom-based html5 / x3d integration model. In *International Conference on 3D Web Technology (WEB3D)*. Fraunhofer Institute for Computer Graphics, June 2009.

[9] Stephan Bischoff and Leif Kobbelt. Streaming 3d geometry data over lossy communication channels. 2002. Available from: http://citeseer.ist.psu.edu/538355.html;.

[10] John Burkardt. Ply files - an ascii polygon format. Available from: http://people.sc.fsu.edu/~jburkardt/data/ply/ply.html.

[11] Xavier Décoret, François Sillion, Frédo Durand, and Julie Dorsey. Billboard clouds, June 2002. Available from: http://artis.imag.fr/Publications/2002/DSDD02.

[12] Dreamingwell.com. Rich internet applications statistics, May 2011. Available from: http://riastats.com/.

[13] Wikipedia english. 3d scanner, May 2011. Available from: http://en.wikipedia.org/wiki/3D_scanner.

[14] Wikipedia english. Mokomokai [online]. May 2011. Available from: http://en.wikipedia.org/wiki/Mokomokai.

[15] David Flanagan. *JavaScript, the definitive Guide*. O'Reilly, 5th edition, August 2006.

[16] Fraunhofer Institute for Computer Graphics. X3dom instant 3d in the html way, May 2011. Available from: http://www.x3dom.org/.

[17] Google. O3d plug-in api (deprecated). Available from: http://code.google.com/intl/fr/apis/o3d/.

[18] Google. Webgl implementation of o3d, May 2011. Available from: http://code.google.com/p/o3d/.

[19] Khronos group mailing list. Public webgl, gl oes element index uint [online]. February 2011. Available from: https://www.khronos.org/webgl/public-mailing-list/archives/1102/msg00094.html.

[20] GSMA. Future challenges for universal service, broadband for all, April 2011. Available from: http://www.europarl.europa.eu/document/activities/cont/201104/20110420ATT18222/20110420ATT18222EN.pdf.

[21] Khronos. Opengl es 2.0 release announcement, March 2007. Available from: http://www.khronos.org/news/press/releases/finalized_opengl_es_20_specification/.

[22] Khronos. Khronos releases final webgl 1.0 specification [online]. March 2011. Available from: http://www.khronos.org/news/press/releases/khronos-releases-final-webgl-1.0-specification.

[23] Khronos. Webgl specification [online]. March 2011. Available from: http://www.khronos.org/registry/webgl/specs/latest/.

[24] Adobe Labs. 3d apis for adobe flash player and adobe air [online]. May 2011. Available from: http://labs.adobe.com/technologies/flashplatformruntimes/incubator/features/molehill.html.

[25] Christian Luksch. Implementation of an improved billboard cloud algorithm, July 2009. Available from: http://www.cg.tuwien.ac.at/research/publications/2009/LUKSCH-2009-BBC/.

[26] Vangelis Kokkevis Matt Papakipos. The future of o3d [online]. May 2010. Available from: http://blog.chromium.org/2010/05/future-of-o3d.html.

[27] Louvre museum. About the louvre: Behind the scenes, museum curator, May 2011. Available from: www.louvre.fr.

[28] NeA. Webgl - part 1: A new challenger appears..., February 2011. Available from: http://insanitydesign.com/wp/2011/02/13/webgl-part-1-a-new-challenger-appears/.

[29] NextEngine. Nextengine scanner specifications, May 2011. Available from: http://www.nextengine.com/products/scanner/specs.

[30] Christophe Porteneuve. *Pragmatic guide to JavaScript*. The pragmatic programmers, November 2010.

[31] Andor Salga. Webgl browser stress tests using processing.js, May 2010. Available from: http://asalga.wordpress.com/2010/05/24/webgl-browser-stress-tests-using-processing-js/.

[32] Rich Sharples. Java is finally free and open [online]. June 2008. Available from: http://blog.softwhere.org/archives/196.

[33] Simple3D. Simple3d [online]. 2006. Available from: http://www.simple3d.com/.

[34] StateUniversity.com. Museum curator job description, May 2011. Available from: http://careers.stateuniversity.com/pages/548/Museum-Curator.html.

[35] Giles Thomas. Webgl lesson 1 - a triangle and a square, 2011. Available from: http://learningwebgl.com/blog/?p=28.

[36] Tiobe. Tiobe programming community index for may 2011, May 2011. Available from: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html.

[37] Web3D. Vrml97 functional specification and vrml97 external authoring interface (eai) international standard iso/iec 14772-1:1997 and iso/iec 14772-2:2002. Available from: http://www.web3d.org/x3d/specifications/vrml/.

[38] Web3D. X3d international specification standards. Available from: http://www.web3d.org/x3d/specifications/x3d/.

Appendices

Planning

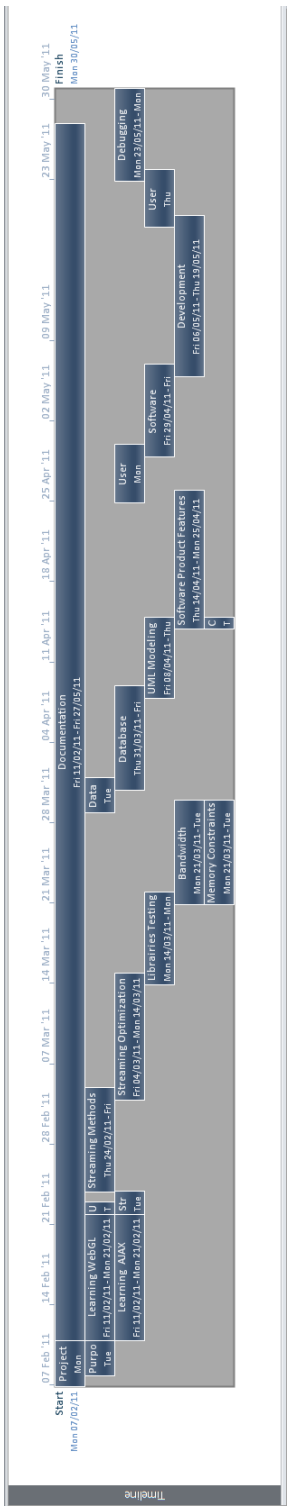

Figure 5: Specific planning

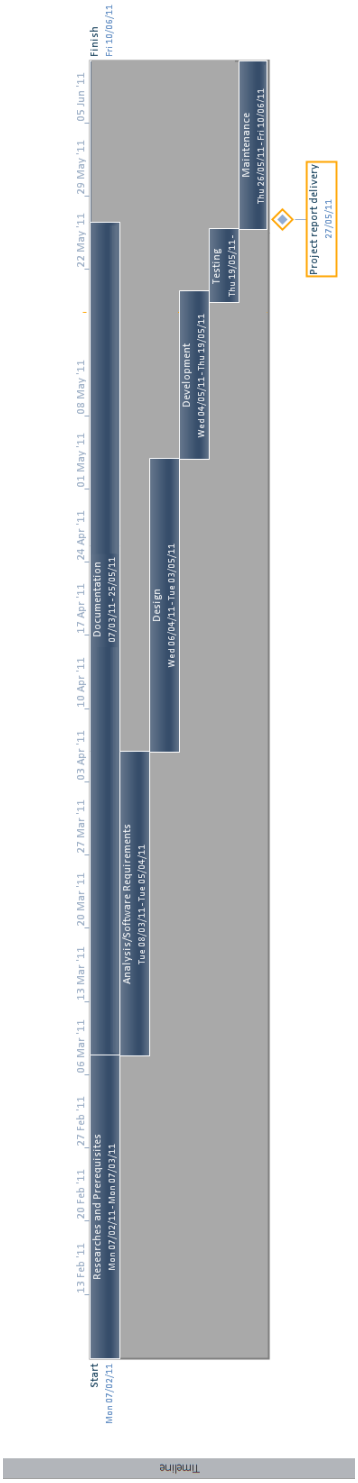

Figure 6: Global planning