Browser-based Medical Image Viewer using WebGL
— Literature Review —

David Basalla
{db913}@ic.ac.uk

Supervisor: Prof. Ben Glocker
Course: CO541, Imperial College London

May 30, 2014

1 Introduction

When creating MRI scans, the resulting image files contain a wealth of data and require specific software packages to view them comprehensively. Among others, the files include high-range images across several dimensions (usually at least X, Y and Z) through a volume. This creates the issue for the users to have the appropriate software installed on their local desktop computer. Additionally software developers have to cater for different operating systems. The goal of this project is to simplify the process of viewing and manipulating these images (specifically the NifTi format) by creating a web-browser-based medical image viewer with an intuitive interface for viewing multiple files and allowing for creation, viewing and manipulation of label maps and other methods of annotating these images. The rationale behind this is to facilitate users to conveniently share and annotate medical image files independently of location or type of computer, following the "Software as a Service" paradigm.

2 Literature Review

2.1 File Formats

- Talk about DICOM/ PACS ??
- NIFTI
- NRRD
- mention slices
- arbitrary number of dimensions, but for scope of the project, only the first 3, X, Y, Z will be looked at.

2.2 Desktop Applications

There exists a variety of desktop software to date allows viewing of medical image data. For this project, a selection of programs have been analysed for functionality.

Firstly, MITK 3M3 by mint medical and German Cancer Research Center allows for easy viewing of a variety of medical image file formats. The layout of the views is customisable and by default shows the X, Y, Z dimension and a 3D view of all 3 dimensions together. (picture) There is a universal range slider that affects all images at once. Labelmaps (called Segmentations) can be added to a scan file, as sub layers and a range of paint tools are provided to colour in regions of the scan. An arbitrary number of layers can be created and saved out as separate files. Furthermore it can create a 3D volume renderings, applying image filters and supplies measuring tools. In general it is a well rounded product with intuitive controls so will serve as a good benchmark test for the final product.

- 3DSlicer (!! open source, plugins, functionality, issues)
- MIview (functionality, issues)
- imview (functionality, issues)

2.3 WebGL

WebGL is a JavaScript API that exposes a computers GPU to the web browser and allows for displaying of complex three dimensional graphics as seen in comparable desktop applications. Although only initially relased in 2011, WebGL is widely supported by most modern browsers (reference), so it's a good time to make use of this for medical visualisation. WebGL is based on OpenGL ES 2.0 and uses the HTML5 canvas element and is accessed using Document Object Model interfaces.

2.4 Web Framework XTK

As suggested in the project specification, the goal of this project is to use the *X Toolkit*(*XTK*), a JavaScript-based framework for visualizing and interacting with medical imaging data using *WebGL*.

It was developed and maintained by Haehn et al the of the Fetal-Neonatal Neuroimaging and Developmental Science Center at Childrens Hospital Boston, Harvard Medical School, US (reference). *XTK* provides an API to load medical images, based on *WebGL* technology for displaying 3D graphics and the *HTML5* canvas elements to display 2D components. It hides a lot of the low-level elements of *WebGL* and is designed to quickly load and configure *DICOM* files of various types. The library is well documented and comes with several tutorials as well as a number of example applications, which will be discussed in the next section. The XTK library also provides a module for creating User Interface (UI), that can be overlaid on top of the image viewers, with easy controls to connect the image content with the UI controls.

2.5 Web Applications

The *XTK* library also provides links to example applications. One of these is *SliceDrop* which has written by the *XTK* developers. It provides part of the desired functionality and shows off the possibilities provided by *XTK*, but also outlines some of its short comings. Like the desktop software *MITK*, the user can load a file and view it in the standard 4 views, split into 3D and 2D windows. The layout of these views is somewhat customisable, although not to the extent of the *MITK*. The user can interact with the images by scrolling the mouse wheel, which changes the index of the current slice. An interesting feature which has been added lately is the incorporation of popular file sharing site *DropBox* to allow users to share files more easily across the internet.

There appear to be some bugs in *SliceDrop*. When loading *NRRD* files, as the image is offset and not centered. The image brightness controls seem not calibrated correctly, as with little mouse interaction, the brightness will clamp to white or black and can not be reset other than restarting the program (refreshing the page for web-apps). Furthermore the app does not allow to load custom labelmaps. Also the viewers provide some functionality which is not clearly communicated to the user, such as holding the 'shift' button and move the mouse will adjust the slice index of the other viewers. In general *SliceDrop* has less than the minimum functionality required for this project, but shows the potential of the *XTK* library.

Brainbook (reference) is a web-app which builds on and extends the features of *SliceDrop* by adding painting tools. It allows the user to paint onto a given file with a standard brush a number of auto selection tools which will fill out a region in 2D or 3D space. The web app offers to save the file, but at time of writing this feature was not working correctly. Since heavily based on *SliceDrop*, it features similar advantages and disadvantages, but provides the painting functionality that is desirable for this project.

- Other Examples (functionality, issues)
- Various DICOM viewers, not compatible with NII/NRRD files tho... - Oviyam, requires server installation, not tried as too cumbersome

2.6 Web Frameworks for Single Page Apps

Modern Webpages which require more complex behaviour than just displaying information can get quite involved. Generally it takes at least HTML, CSS and JavaScrip to create a useful website. It is possible to include the code from each language in the main index.html file, but this is undesirable as it lacks modularity and will be hard to test and debug.

At the time of writing, Backbone seems to be a good choice to manage the complexity and structure of the project. Backbone is used to structure client-side applications (those that run in a web browser). It was written by Jeremy Ashkenas and is designed for developing single-page web applications, and for keeping various parts of web applications synchronized. It is based on the model–view–presenter (MVP) application design paradigm, providing Events, Models, Collections and Views. Models represent the information in class like structures, Collections are lists of Models and Views handle the visual representation on the actual HTML site. Finally Events are used to keep track of state changes across the website elements, models and views. It builds on JQuery.js and Underscore.js, both libraries which simplify dealing with HTML elements.

Regarding the design aspect, it looks like twitter-bootstrap.js supplies a convenient library for creating visually pleasing user interface elements. This will cut down on having to spend time to design custom elements with CSS and JavaScript. Bootstrap.js supplies buttons, drop-down menus and commonly-used icons.

As this project will make of several modules, it will be important to manage the dependencies of the modules. For example Underscore.js and JQuery.js will need to load before Backbone.js is used. XTK and twitter-boostrap are additional modules that will have to be loaded. Require.js is a javascript library that provide the ability to asynchronously load nested dependencies. Traditionally javascript files or modules are loaded sequentially, where the order matters in case a module depends on another module. Require.js will make it possible to split up everything into neat modules and facilitate testing of all the components.

3 Requirements

The goal of this project is to build a web browser based Medical Image Viewer. This will be achieved by making use of WebGL to display 2D and 3D graphics. In general the project aims to provide similar functionality and user experience as the desktop application MITK (discussed earlier). The essential requirements are as follows:

- The user can load and view a scan file (of type NIfTI)
- The user can view a file in 2D and 3D
- The web-app provides controls for viewing the scan files at different brightness and contrast levels
- The user can load and view a labelmap for a given scan file
- Alternatively the user can paint a custom labelmap for a given scan file
- The user can save a labelmap to his computer
- The user can load additional scan or label maps into the viewer and compare them to previously loaded files

Given enough time, secondary requirements can be implemented, which are considered to be more complicated and not essential to the working of the web app:

- The user can create custom annotation maps to label given scans
- The web app provides a method for sharing means of sharing data across the internet (similar to the DropBox implementation in SliceDrop)
- The web app provides image filters that can be applied to a scan or label map
- The web app provides measuring tools that can be applied to a scan or label map
- A three dimensional, parameterised model of the scan be viewed in the 3D view
- Provide support for other file types (possibly fix NRRD)

4 Progress

Some effort has already gone into planning and implementing the project. The initial plan was to create a working prototype in a relatively short time. This was however foiled by the complexity of the task of creating a functioning web-app. At first, tests were done building on tutorials that are provided by the XTK toolkit site. This included a simple HTML front end as well as some javascript files to handle the XTK objects.

Loading local files in a web browser proved to be one of the early issues. Due to security issues, Chrome and other browsers do not allow local files to be loaded onto the HTML canvas element, tying into the same origin policy that browsers try to uphold. A way around this was found by analysing the SliceDrop code and noticing that it uses the HTML FileReader to convert the local file into a byteArray which can then be loaded into the XTK viewers.

Having some concerns that modifications will have to be made to the XTK toolkit, the author managed to successfully clone and fork the project from github. Since things are never as easy as hoped for, editing and rebuilding the toolkit proved tricky, since it employs yet another unknown JavaScript library, called GoogleClosure.js. After learning how to use this module and other little issues mainly due XTK being developed on Linux and not Windows, the OS the project is currently being developed on, the toolkit managed to recompile successfully.

Once some basic tests with the XTK toolkit proved that it could be used to display local scan files, it was time to start adding some functionality to the website. This quickly showed that the previous implementation of the HTML website was highly naive, and that it would require a restructuring from the ground up. Almost as an example of why web programming can be tricky, the HTML site was mixing elements from HTML, CSS and JavaScript without much structure. When separating out the JavaScript parts into separate files, the issue of module loading and dependency became apparent.

From the group project earlier in the term, the author was aware of JavaScript libraries that aid in structuring a web-app such as Backbone.js, Require.js and JQuery.js. So the next few days were spent trying to get to grips with these and understanding generally how web apps are built. This seems to be a rich topic in its own right, and will undoubtedly take more time still to become comfortable with. However it helps that Backbone tends to enforce a somewhat object-oriented structure, which ties development back to principles learned in the Masters course over the year. Even when starting to implement the Backbone it becomes apparent how easy it is to fall back into mixing different language in the files and to enforce that every component just does the one task it is supposed to do. Talk about templates...

When realising that an object-orientated structure can and should be applied to the project, it prompted the author to create a UML diagram with possible class design (see Appendix). This will prove useful when implementing the Backbone components and helps to visualise the flow of control of the web-app. Additionally, visual UI design have been made to show how the front-end would look like to the user and what controls would available to him or her (see Appendix). After it some research it also became apparent that yet another JavaScript library called twitter-bootstrap is very popular among the web community to create UI designs for buttons and menus, so time was spent learning how to tie this into the project. Currently, the plan is still to create a working prototype that has all the essential requirements fulfilled.

4.1 Concerns

4.1.1 XTK

So far, testing the XTK toolkit has revealed some concerns. In general the library is not so modularised as hoped for, as it does not seem possible to add a custom number of label maps. There is at present also no way to paint a custom label map. However, as already discussed, it is possible to fork the project from github and introduce changes, so this will have to be explored when adding additional features.

Other useful functionality is hard-coded into the library, which may need suppressing or changing of the input triggers. For example currently, the are no display lines for all viewers when one clicks or moves the mouse over one view. Furthermore the user has to hold down the shift key and move the mouse over a view, to affect the slice indices of the other views. This can hopefully be tweaked in the module.

There appears to be a bug with loading NRRD files, as they as displayed with an offset in the 2D and 3D viewers. This behaviour is visible in SliceDrop as well as in the XTK tutorials. The XTK developers have been made aware of the problem.

It appears that there is only a very small and inactive community for the XTK library. There is

a group on the popular tech forum StackOverflow but questions are being asked infrequently and get answered even more rarely.

4.1.2 Cross Browser Compatibility

Judging from limited previous experience and source online, cross browser compatibility can be a difficult issue, as different browsers on different operating systems are implemented differently. Therefore what might work very well on Chrome could not work at all on Internet Explorer, as some functions are implemented differently or not even available in certain browsers. The use of WebGL already imposes a restriction on which browsers are usable, so it would be desirable to have at least other restrictions as possible. This is something that will have to be paid close attention to.

5 References

$http://www.frontiersin.org/10.3389/conf.fninf.2014.08.00101/event_abstract$
$http://users.loni.ucla.edu/\ pipeline/paint/$
$http://backbonejs.org/$
$http://requirejs.org/$
$http://getbootstrap.com/$

6 Appendix

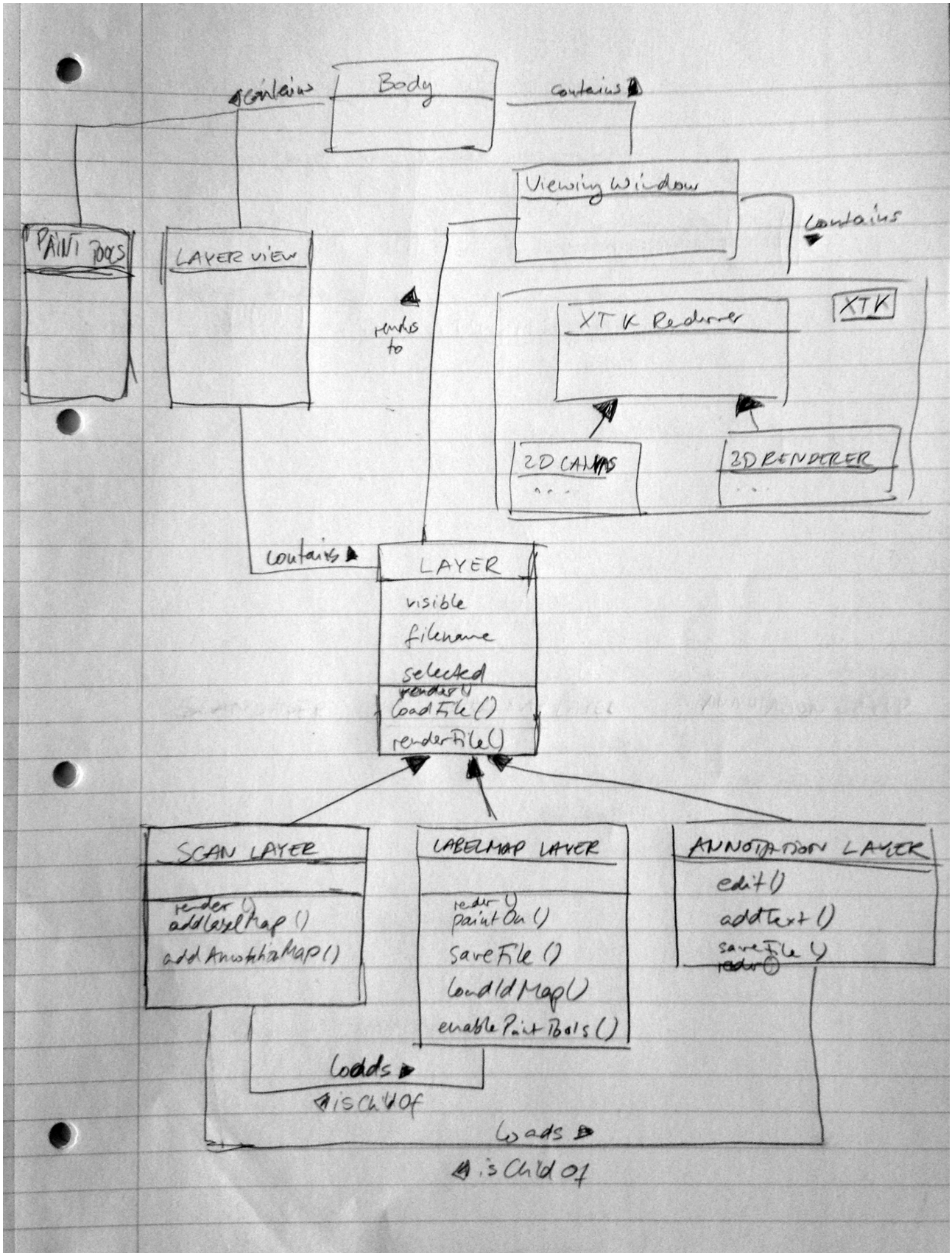

Figure 1: Early UML design

Figure 2: Early UI design