Visualizing 3D geology in web browsers using X3DOM

Øystein Malt[a], Atle Geitung[a], Harald Soleim[a], Daniel Patel[ab]

[a] Western Norway University of Applied Sciences (HVL) [b] Christian Michelsen Research (CMR)

Abstract

This work presents an application for visualizing subsurface geological data in 3D in web browsers, using the X3DOM framework. The data supported is 3D terrain, vertical subsurface cross sections and subsurface measurements from wells. Data is visualized for the area of Svalbard. To avoid low-level development, we use X3DOM, which hides the details of graphics rendering in high-level, declarative XML syntax. The resulting application is cross-platform and runs on computers, tablets and mobile phones with adequate graphics capabilities. The work is a summarization of the first author's master's thesis.

Introduction

The visualization of subsurface geological data is important in many fields such as ground-water mapping, oil and gas exploration and CO_2 storage. In particular, the motivation for this work comes from the Virtual CO_2 Laboratory project (VIRCOLA) [1], initiated in 2012. The purpose of VIRCOLA was to improve data sharing and cross-disciplinary collaboration to facilitate the Norwegian National Center for Environmental-friendly Energy's research on subsurface CO_2 storage [2]. The National Center is a large consortium consisting of 11 partners from academia and industry, constituting over 100 researcher. To achieve this, an efficient tool for researchers to discover produced data and publications of other researchers, and identify fruitful collaboration constellations was needed. Such a tool would need to spatially covisualize data so researchers could identify other relevant data in proximity to their own data, and display owner and publication info about the data, so collaborations could be identified. VIRCOLA tried out several existing industrial-grade geoscientific 3D visualization applications such as SKUA [3] and Petrel [4], for covisualizing the data. However, it was experienced that these tools were built with other goals in mind than easy data and collaboration discovery. They were more suited for detailed interpretation and low-level domain specific tasks. Also, such software typically required costly licenses and had large user manuals with steep learning curves due respectively to the revenues in their typical domains of use (hydrocarbon and mineral extraction), and to the complexity of geological interpretation. What the researchers wanted was an easy-to-use visualization tool with the possibility to navigate to, and turn on and off visualization of individual datasets, and display additional information about the data. In addition, the tool should allow users to upload their own data and collaboratively enrich the database. We were not able to identify existing applications with such functionality. A similar observation has been done by Nimtz et al. [5] and Reumont et al. [6]. This paper describes a proof-of-concept application demonstrating that such a solution most likely can be implemented with X3DOM (with JavaScript). To encourage use and adoption, we have created a cost-free web application with inherent cross device,- and platform support (Figure 1, left), eliminating the need for payment, computer access rights management and installment. We provide all source code [7] so others can use it freely. The subsurface data supported in our solution is vertical cross sections of seismic data or any other measurements represented as 2D images, and subsurface measurements along the path of drilled wells. Several other types of subsurface data exist which would be useful to visualize, such as reservoir grids, fault and horizon surfaces and seismic volumes. However, we focused on cross sections and wells only as successfully visualizing them gives a good indication of the strength of X3DOM.

This paper was presented at the NIK 200x conference. For more information see http://www.nik.no/

We have created a client-server solution where all data is stored in a database on the server and the client runs in a web browser using JavaScript and X3DOM.


```
<html>
...
  <x3d width="800px" height="640px">
    <scene>
      <transform translation="0 0 0">
        <shape>
          <appearance>
            <material diffuseColor="0 0 1"/>
          </appearance>
          <sphere></sphere>
        </shape>
       <transform>
    </scene>
  </x3d>
...
</html>
```

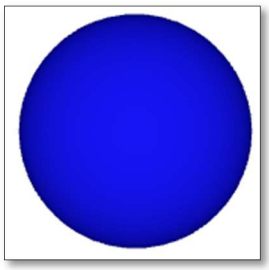

Figure 1 Left: Photo of a mobile phone running our app. Right: X3DOM code and the resulting rendering.

Related work

This article is based on the master thesis [8] by the first author, which describes the work in higher detail. The central technology we build on is the ISO standard X3D and its implementation X3DOM [9]. X3D is a standard for representing 3D computer graphics in a declarative manner. It describes a file format, a scene- and an event-graph. X3D scenes are encoded using XML syntax derived from its predecessor VRML97 [10]. X3DOM is an implementation of the X3D specification, along with support for regular DOM (Document Object Model) [11] interaction with the X3D elements declared in HTML5 content (see Figure 1 right for an example). The X3DOM framework has support for most major browsers on desktop and mobile [12]. Thus, most platforms and operating systems are supported inherently by the framework. X3DOM is open source and dual-licensed under the MIT and GPL license [13].

X3DOM uses the low level graphics API WebGL which interfaces with the underlying computer's hardware to render graphics. The WebGL specification is maintained by the Khronos Group [14] and has become the standard API for 3D graphics in browsers. [15].

Similar to our work and driven by the same motivations is the work by Reumont et al. [6] from 2011 who investigate X3D for geoscientific visualization. However, both X3D support and web development libraries have improved much since then, making our solution in comparison to theirs work directly in a browser without the need for plugins, and support smooth loading and interaction with terrains using level of detail support. Also whereas they seem to present a monolithic and static X3D document describing the whole application, we have developed a three-tier application that dynamically creates X3D for graphics rendering, using separate libraries for GUI rendering and using a database residing on a server.

Ziolkowska and Reyes' paper [16] from 2016 presents interactive visualization models for geo-temporal and geospatial data in virtual globes. The authors use KML (Keyhole Markup Language) to visualize data on a virtual globe. There are several virtual globes available that supports KML, notably Google Earth, NASA WorldWind and Cesium [16]. The article discusses the support for subsurface visualization in virtual globes, noting that these "were not originally developed with the purpose of representing data below the Earth's surface". From our own experiences with using different virtual globes, navigating below the surface of the globe is hard, as support for this is not made. Adding geometry below the surface can be done but this is subsequently overdrawn, and solving this problem would require access to low-level rendering details which are not clearly exposed. The models presented in the paper [16] circumvents these limitations by introducing transparent earth layers around the globe, created using KML, treating the top layer as the surface of the earth, and thus enabling the camera to seemingly move subsurface to view the geospatial visualizations. Due to this workaround, the real globe rendered by the respective framework can be seen as an artifact inside the transparent layers.

Krämer and Gutbell's paper [17] from 2015 investigates the open-source frameworks Cesium, Three.js and X3DOM for use in geospatial applications in web browsers. However, the paper only explores visualization of surface geometry such as houses, roads and trees, whereas we demonstrate visualization of subsurface geometry. The authors give a qualitative comparison of the frameworks based on several software prototypes developed to assess them. They find that each framework has different approaches, goals and target groups. Cesium is targeted specifically at geospatial applications, and is well suited for such. Three.js offers direct access to WebGL, making it suitable for a wide range of use-cases due to this flexibility, although developers might have to implement desired features. X3DOM is declarative, and based on the standardized X3D file format. This has the advantage of developers not having to learn how to use the low-level WebGL API, and since it is built on a standard it is also suitable for applications required to be supported for a long time.

3D Geology in a web browser

Our solution demonstrates that it is possible to interactively visualize and dynamically upload subsurface geological data in web browsers using X3DOM. Users can access the web application from a variety of platforms and devices, and quickly navigate to interesting data points using the application's viewpoint functionality. This, along with the rendered topography surrounding the data, allows users to get a quick overview of available data in a geographic area, see Figure 2.

Specifically, the application lets users visualize subsurface geological data of wells and of vertical cross sections, with accompanying information of e.g. owner and related publications. The data resides in a database stored on a server which users can upload data to directly from the client web app. The wells and slices are visualized in a scene along with a topography of Svalbard. The user can navigate around in the 3D topography and go underground to view the subsurface data. Wells consist of physical measurements, such as temperature, taken along the 1D path of a drilled well

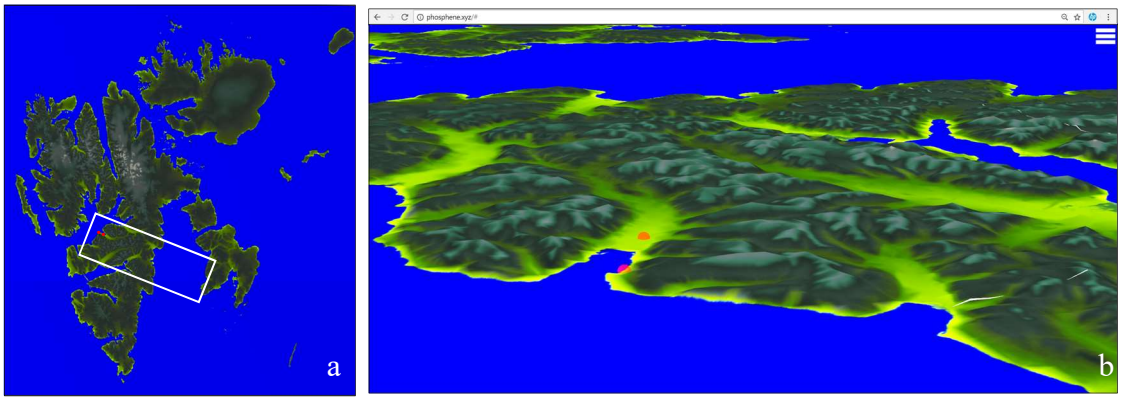

Figure 2 a) Overview screenshot of Svalbard heightmap data. b) Screenshot of the entire browser window showing a 3D view of the white rectangle in a) with view eastwards. b) The two red markers show the top of two wells situated in Longyearbyen.

They are visualized as colored curves along the well path where the color represents one of the physical measurement, see Figure 3. Cross sections are 2D images representing vertical slices into the earth. Typical such 2D images are seismic slices produced by sending sound waves into the earth and processing the echoes. See Figure 4.

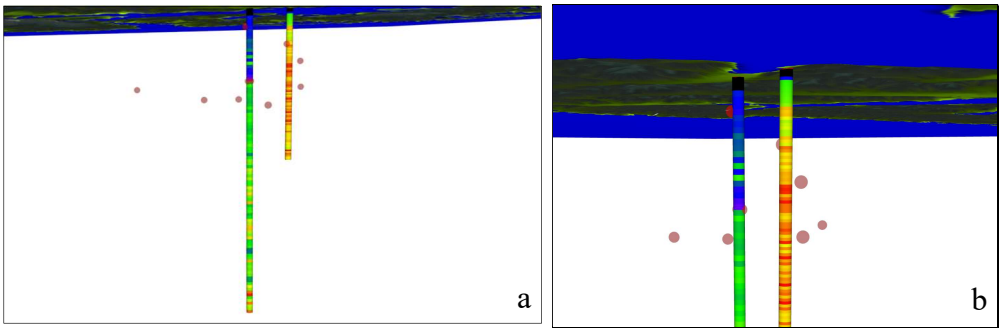

Figure 3. a) This shows Figure 2b) seen from below the horizon revealing two wells. b) Zoom-in on a). Left well shows "Self Potential" measurements while right well shows "Gamma radiation" measurements mapped to a blue-green-yellow-red rainbow colormap. Black color represents the lack of data. Red spheres show places where data visualization has been toggled off.

As data often overlap each other or obstruct view to data behind, functionality to easily toggle visualizations on and off is implemented by left clicking on the visualization. When toggled off, a red translucent sphere is shown instead indicating that there is data at this point. Such spheres are clearly visible in Figure 3 and Figure 4.

Interacting with slices and wells

The slices and wells can be right clicked for getting further information. Then an information dialog box (Figure 5 left) shows up containing a header with the name of the object, a textual description of the data ("Description") and clickable links to relevant documents ("Related articles").

The values of the selected attribute are then mapped to colors along the well according to the color legend and the user defined min and max value. For more accuracy, the user can read out an exact value on a well using the "Value at click point" information.

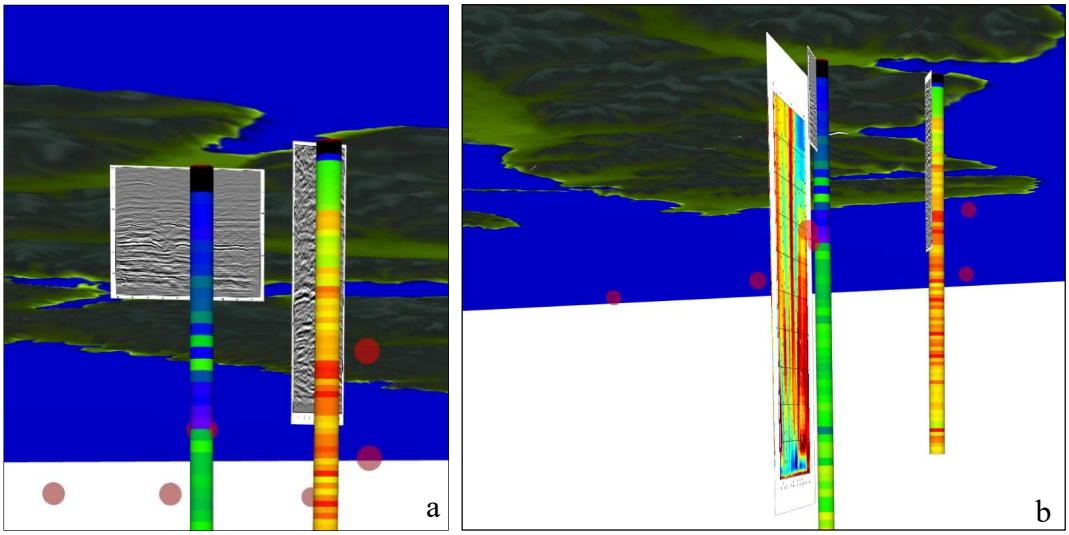

Figure 4 a) Two seismic slices intersecting the wells. b) Slice with magnetotelluric data shown to the left.

Whereas a slice shows a single measurement as an image, a well typically has several measurements, called attributes, taken along its path, such as gamma radiation and temperature. Which attribute to visualize is chosen with a drop-down menu (see Figure 5 right).

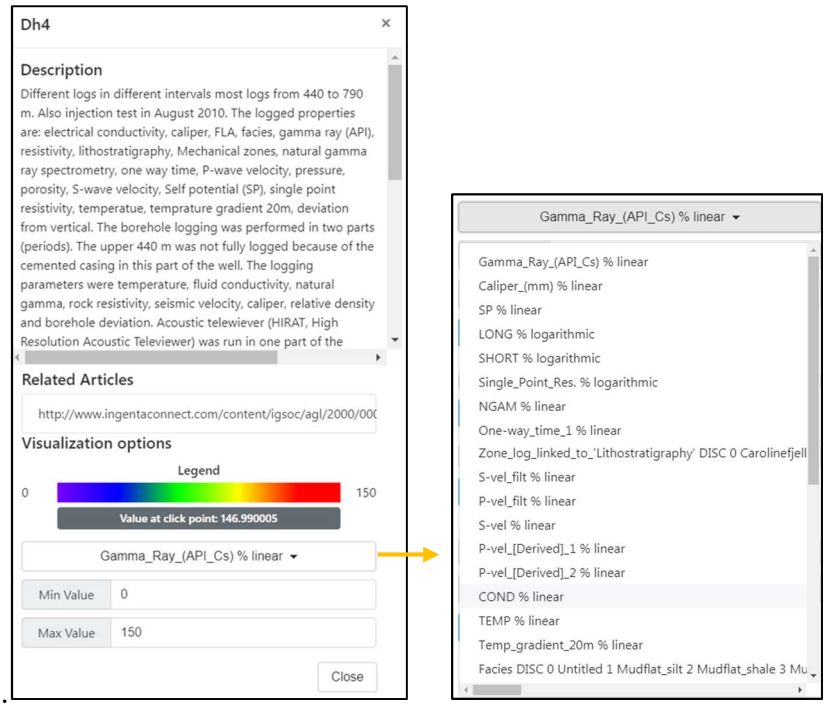

Figure 5. Left: Right clicking on a visualization opens up a window giving a description of the data and links to related articles. For well objects also several visualization options are presented including a drop-down menu (right) for selecting which well attribute to visualize.

The system offers an alternative way to navigate to a slice by finding its name in a list of all slices, see Figure 6 bottom left. This is practical when the user wants to navigate to a slice based on its name, without knowing where it is. From the list, it is also possible to quickly toggle the visualization of a selection of slices, or to open the data description window. The list is part of the sidebar, which is turned on by clicking the three horizontal lines shown in the upper right corner of Figure 2 b). The sidebar has settings for the terrain and for uploading data, which are described in the two following chapters.

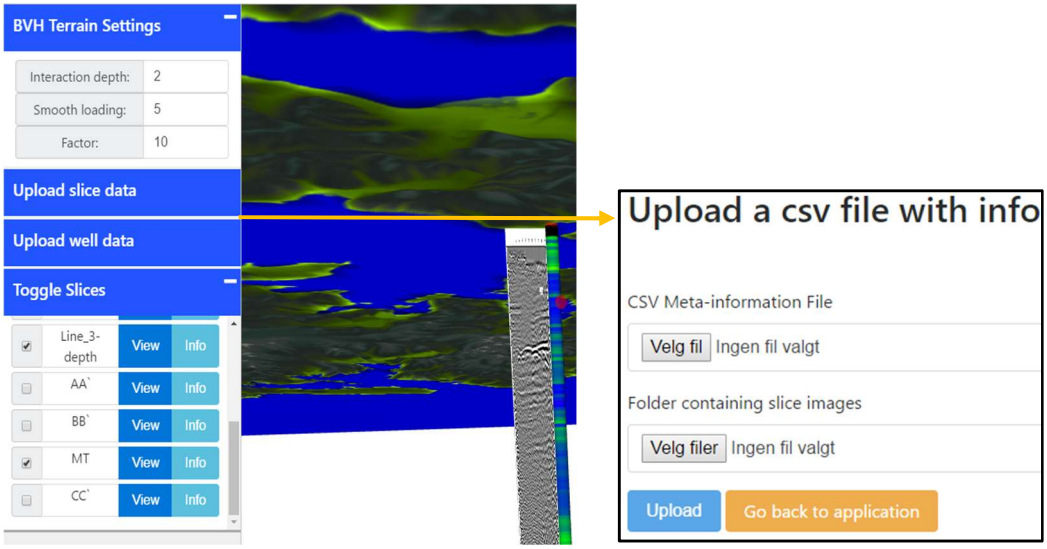

Figure 6. Left. The sidebar menu. Right: The upload data window.

Uploading slice and well data to server

From the sidebar in Figure 6 left, the user can upload slice and well data into the database residing on the server. As our goal was to demonstrate 3D capabilities, functionality for staging the uploads so that an administrator can verify the data before making it accessible to all users, and functionality for assigning read/write rights to users, has not been implemented. Also, we have not implemented user authentication with login and password.

Uploading slices

The data that must be uploaded to define a slice has three parts: textual metainformation about the data, a two-dimensional image of the geological data in "png" format, and the positioning of the data. The textual metainformation consists of the dataset name, a dataset description, and a list of URI links to related information (Figure 5 left). The positioning is defined by specifying start and end coordinates using the UTM [17b] coordinate system. This defines a line on the globe that the slice is vertically situated in. To specify at which depth interval the data resides, a start and end depth must be given.

To upload the data (Figure 6), the user selects a csv (comma-separated values) file and a list of images. The csv data is originally defined in, and exported from, a Microsoft Excel spreadsheet document. Excel was used to provide a user interface for manually editing and constructing the data during the VIRCOLA project. Each row in the spreadsheet has information for one slice. Three cells are used to describe the metainformation (name, description and URI's) where multiple URI's are stored in a single cell using a semicolon separator. Additional cells are used to store the coordinates and the image file name.

Uploading wells

The data that must be uploaded to define a slice has two parts: textual metainformation about the data as for slices, and a well log. Each well log is stored in a file which essentially contains a list of points in 3D space (UTM and depth coordinates) describing the well path and a list of measurements (attributes) for each point. The header of the file

contains a list of strings describing which measurements the log contains and is used to populate the dropdown shown in Figure 5 right. The well log is stored in a file following the Log Ascii standard [18].

To upload a set of wells, the user selects a csv file describing them and then selects the well log files. The upload window is identical to that for slices shown in Figure 6 right. As with the geological slices, the csv data for the wells are originally defined in an Excel spreadsheet (the same that stores the slices, but in a separate tab) where each row describes a well. The first cells in each row contains the metainformation and the last cell contains the name of the well log file. When the server receives the well log files, it parses them using our own JavaScript reader and stores the data in the database.

Preprocessing of terrain

Terrains can cover large areas and thus contain a high number of triangles. To achieve interactivity, a level-of-detail (LOD) scheme [19] is applied which calculates several versions of the terrain from the original terrain, each with an increasing degree of coarseness containing a decreasing number of triangles. A LOD scheme achieves interactivity by showing coarser geometry for terrain far away from the viewer and finer geometry for terrain close to the viewer. Based on a user-defined upper bound on how many triangles to be rendered, the system can choose appropriate levels of details during rendering. Terrain LOD algorithms can be complex to implement, however there is a ready implementation in X3DOM called the BVHRefiner node [20], which we use. The terrain is constructed from a heightmap image of the area. In a preprocessing step, using a software called the BVHRefiner Dataset Converter [20], the heightmap is divided into tiles and each tile is subsampled into new tiles representing different levels of detail for the same area. The tiles are stored as png files which the BVHRefiner selectively loads during runtime depending on the viewpoint. See Figure 7 for LOD example.

The height-map image is extracted from an UTM referenced GeoTIFF file, of Svalbard where each pixel represents a 50x50 meter area, downloaded from the Norwegian Polar Institute web page [21]. The GeoTiff to PNG conversion is accomplished using GDAL (Geospatial Data Abstraction Library), which is a set of programs and libraries providing access to geospatial data in raster or vector formats [22]. In addition to extracting the heightmap, GDAL was used to generate the colors of the terrain from the terrain height using a standard GIS elevation palette going from blue at

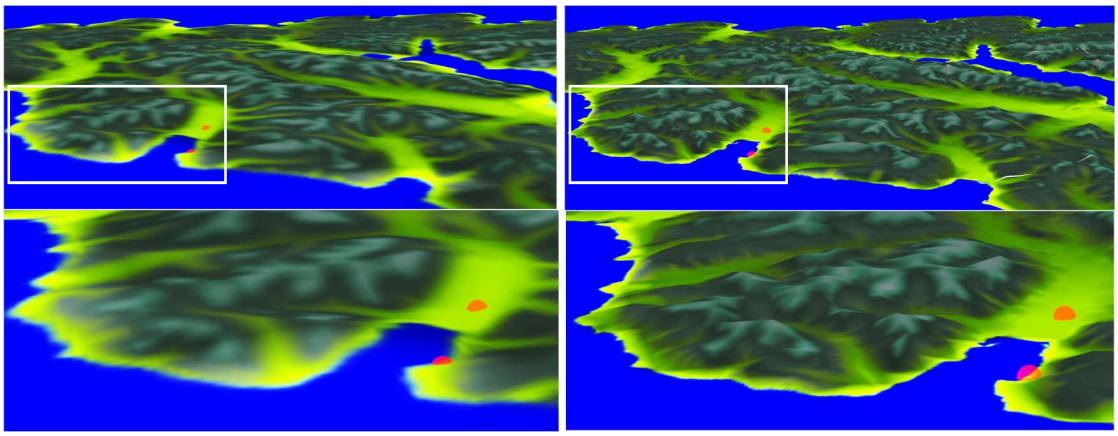

Figure 7 LOD Left: Coarse LOD with zoom in of white square below. Right: Fine LOD with zoom in of white square below. The coarse LOD to the left is shown during interaction such as rotation to achieve responsiveness.

sea level through light green at plain level, green at lower slopes to white at high elevations. However other terrain textures could have been used such as satellite photo or geological maps which have colors according to the rock type of the terrain.

Representing slices in X3DOM

X3DOM provides predefined geometric primitives such as spheres (<sphere> tag in Figure 1), boxes (<box>) and cylinders (<cylinder>) which we use to build up our visualizations. Transformations (<transform> tag in Figure 1) are used to position them, and callback functions can be registered on any collection of primitives (<shape> tag in Figure 1) to e.g. call a JavaScript function when graphics is clicked on.

When the application is started, the first thing that happens is the loading of slice data from the server. Once this has been loaded, each slice is processed on the client-side with JavaScript for generating X3DOM code describing its visualization. The X3DOM code is inserted into the HTML page by using the DOM API.

The geometry of the slice shape is declared as a box, where the dimensions of the box are specified by the slice's size. Listeners for the *click* event are added to the <*shape*> X3DOM element in the slice using X3DOM's *onclick* attribute. The individual pieces of X3DOM code for each slice are then concatenated and inserted into the X3DOM <*scene*> in the DOM.

The toggle functionality requires a substitute visualization to be shown in the slice visualization's place when it is toggled off. Whether the slice shape or the substitute shape is displayed, depends on the value of the *visible* property of the slice data object. The event listener function of a slice differentiates between left and right clicks. If a left click is registered, the Boolean visibility state of the slice is inverted. The X3DOM code for the clicked slice or substitute shape is then removed from the DOM, and replaced with the X3DOM code for the "opposite" shape. X3DOM notices the change in the DOM, and updates the scene accordingly.

The sidebar menu is created on the server by querying the database for all slice data and constructing a list of GUI elements. GUI is created in our app using the Angular framework [23]. Clicking the toggle button triggers an Angular event that is propagated to all dependents, thereby updating the graphics. Clicking the view button for navigating to a slice, uses the database id of the slice to retrieve the slice position and orientation, calculates a viewpoint looking perpendicular on the slice, and moves the camera to this viewpoint using the X3DOM <*viewpoint*> element. This X3DOM element allows specifying a location and orientation that can be applied to the camera, so that the camera navigates to the viewpoint with a smooth animation. When the *Info* button is clicked, the information dialog box for a slice is created and displayed using Angular GUI functionality.

Representing wells in X3DOM

The shape of a well is described by an array of 3D positions, where each position represents the start of a cylinder segment of the well, and the next position represents the end of the current segment and the start of the next one. For each well attribute, there is an equally long array containing the measured values for that attribute. Using several cylinder segments instead of a single, allows us to use different colors on each segment to communicate attribute values, and to support curved wells, although curved wells do not exist in our Svalbard data.

To generate the X3DOM code for each cylinder shape, a height, position and color is required. An object with arrays for each of these attributes and for each well segment is created using the well data object fetched from the server. This function loops over the

positions array of the well, and uses the depth value of the current and next position to calculate the height of the well segment. The depth of the position for the cylinder shape is translated by half the height of the segment, so that the positions represent the top of the cylinder.

The color coding is calculated by using the visualization options found on the well data object. These options include the minimum and maximum value of the current attribute to be visualized. The attribute value is normalized within the min and max value, and this normalized value is used to look up the color in a color array corresponding to the color legend.

For each cylinder, a <*transform*> element is used to position it. After the X3DOM code for the wells has been inserted into the <*scene*>, event listeners for each individual cylinder shape is added. This is to enable right-clicking on any part of the well for presenting the information dialog box and for displaying the attribute value of the clicked point. Toggling and showing the information dialog box is done in the same way as for the slices.

The server

The server is responsible for providing the client with access to the database, where the data for slices and wells reside. For this server, we use Node.js [24], which is a runtime for JavaScript, letting us use the language outside the browser. The ecosystem around Node.js contains a large number of extensions/plugins organized in the Node Package Manager (npm), enabling rapid development. One package used in the server application is Express.js [25]. This package facilitates creating so called RESTful web applications [26] that abide by sound design principles. In addition to providing an API interface to the database, the server stores heightmap and texture images used by the BVHRefiner node. These are served statically from the server. A MongoDB database [27] is used for storing data. MongoDB is a NoSQL database, which means data is stored in documents instead of traditional SQL table structures. A document in MongoDB is stored in JavaScript Object Notation (JSON) [28], allowing us to easily map query results to JavaScript objects. Another npm package, Mongoose [27], is used for interacting with the database. This package provides simple ways for modeling data structures in MongoDB, and for querying the database. Figure 8. *The mongo shell application used to show the database representation of a slice* shows the database query for the magnetotelluric slice object shown in Figure 4 b) and the formatted database response, using the MongoDB shell application.

```
> db.slicemodels.find({_id: ObjectId("592df7a46223ca3d5eaecb37")}).pretty()
{
        "_id" : ObjectId("592df7a46223ca3d5eaecb37"),
        "name" : "MT",
        "imageUrl" : "/image_model/MT",
        "imageType" : "image/png",
        "description" : "Magnetotelluric measurement in Adventdalen, 2km deep",
        "position" : "-208.51926069100847,124.82386065742983,-27.347505608375172",
        "startPos" : "-219.24167055515267,130.29883210680472,-27.347505608375172",
        "endPos" : "-197.7968508268643,119.34888920805493,-27.347505608375172",
        "start_e" : 15.69257295,
        "start_n" : 78.21979138,
        "end_e" : 16.11838439,
        "end_n" : 78.17359123,
        "start_depth" : 0,
        "end_depth" : 54.695011216750345,
        "related_links" : [
                "http://www.earthdoc.org/publication/publicationdetails/?publication=80544"
        ],
        "__v" : 0
}
```

Figure 8. *The mongo shell application used to show the database representation of a slice.*

Discussion

In this article, visualizations were built by combining existing X3DOM nodes. However X3DOM lets developers define their own X3DOM nodes with custom appearance and functionality. Therefore the well and slice objects could have been represented as two custom and reusable X3DOM nodes. Custom nodes are created using X3DOM's JavaScript API, which can make use of WebGL and GLSL shaders. However, one of the reasons for choosing X3DOM as a graphics framework in this work was to achieve a high level of abstraction through declarative programming and avoid low level WebGL programming. Low level WebGL programming can be avoided by using libraries such as Three.js [29], however such libraries still require the user to understand JavaScript and large imperative APIs instead of declaring and attaching high-level X3DOM nodes with well-defined and simple interfaces. Therefore, the approach of creating custom X3DOM nodes was not investigated. We believe that X3DOM is well suited for situations where the application can be built using already existing nodes. If this is not possible we believe that using an imperative framework with direct access to the underlying technologies (such as WebGL or Three.js) requires less work than creating custom X3DOM nodes. This is because the latter requires first implementing the visualization with an imperative framework and then encapsulating the solutions into an X3DOM node. On the other hand, if the custom nodes that need to be created have a high level of reusability, and the project consists of several programmers, it might still be better to invest the extra time into creating these nodes. This would require a small selection of programmers to perform, possibly time consuming, low-level coding and create a high-level library of custom X3DOM nodes. Then the rest of the team would be users of the library and would not need knowledge of low-level APIs such as WebGL. Our impression is that X3DOM is an excellent choice of framework, that requires very little knowledge outside of standard DOM manipulation.

Due to space constraints, we have not discussed our experiences using the Angular framework. We originally tried to express all of the web logic in Angular's declarative syntax as this would fit well with X3DOM which is also declarative (as opposed to imperative languages). However, we were not able to make these technologies work together the way we wished. For details, see Malt's thesis [8].

Conclusions and future work

In conclusion, the application created in this work shows that it is possible to interactively visualize 3D subsurface geological data using X3DOM. Users can access the web application from a variety of platforms and devices, upload data, quickly navigate to interesting data using the application's viewpoint functionality and get background information about the data. This, along with the rendered topography allows users to get overview of available data in an area, thus fulfilling the requirements of the application as described in the introduction. The source code can be downloaded here: [7].

As this was a proof-of-concept, much functionality is lacking for it to become a fully usable software. A sidebar for toggling wells was not implemented in our version. Useful terrain functionality is lacking, such as being able to toggle the terrain on and off; changing the texture that is projected to the terrain for e.g. showing a geological map of rock types (lithology); to support dynamic uploading of new terrain areas and to improve the navigation control for moving around in the terrain. Other subsurface visualizations of e.g. 3D surfaces representing faults, horizons and outcrops, as well as reservoir grids and volumetric data should be supported. 3D surfaces are supported in X3DOM, and for large surfaces, the LOD approach we used for terrain can be applied. Large reservoir

geometries could be a challenge to support, but sizes that fit in the GPU memory could be represented using triangles in X3DOM. Volumes could be visualized by extending our slice visualization so that the user can move a slice through the volume, requiring only the visible slice data to be resident on the client. But even true volumetric rendering supporting translucency is supported in X3DOM [30], however, this approach is taxing on the client as all slices of the volume must be resident in client memory. Finally, the management of users with logins and different access rights as described earlier including also a more granular way of uploading and removing data should be in place. We see no limitations in the technology we have used in this work, for realizing the aforementioned functionality.

References

[1] S. Tavakoli and T. Langeland. VIRCOLA - A Tool for Visualization and Cross-discipline Collaboration Related to CO2 Storage. Fourth EAGE CO2 Geological Storage Workshop. 2014

[2] "SUCCESS centre" http://www.fme-success.no/. [Online] [Accessed: 31- May- 2017].

[3] "SKUA® Software Suite by Paradigm®", *Pdgm.com*, 2017. [Online]. http://www.pdgm.com/products/skua-gocad/. [Accessed: 31- May- 2017].

[4] "Petrel E&P Software Platform", *Software.slb.com*, 2017. [Online]. http://www.software.slb.com/products/petrel. [Accessed: 31- May- 2017].

[5] Nimtz, M., Klatt, M., Wiese, B., Kühn, M., Krautz, H.-J., 2010. Modelling of the CO2 process- and transport chain in CCS system-examination of transport and storage processes. Chemie der Erde –Geochemistry, 70, Suppl. 3, pp. 185-192.

[6] F. Reumont, J. Arsanjani and A. Riedl. Visualization of geologic geospatial datasets through X3D in the frame of WebGIS. International Journal of Digital Earth. January 2011

[7] Ø. Malt. Source code. https://bitbucket.org/oysmal/ [Online] [Accessed: 31- May- 2017].

[8] Ø. Malt. Using 3D functionality available in current web-browsers to create and visualize geological models. 2017. Master's thesis. http://bora.uib.no/handle/1956/16264

[9] "X3DOM at a Glance", *X3DOM*, 2017. [Online]. https://www.x3dom.org/. [Accessed: 31- May- 2017].

[10] J. Behr, P. Eschler, Y. Jung and M. Zöllner, "X3DOM: a DOM-based HTML5/X3D integration model", *Web3D '09 Proceedings of the 14th International Conference on 3D Web Technology*, 2009.

[11] "W3C Document Object Model", *W3.org*, 2017. [Online]. https://www.w3.org/DOM/#what. [Accessed: 31- May- 2017].

[12] "Browser support - x3dom.org", *X3DOM*, 2017. [Online]. https://www.x3dom.org/contact/. [Accessed: 31- May- 2017].

[13] "Getting Started with X3D | Web3D Consortium", *Web3d.org*, 2017. [Online]. http://www.web3d.org/getting-started-x3d. [Accessed: 31- May- 2017].

[14] "WebGL - OpenGL ES for the Web" https://www.khronos.org/webgl/ [Online] [Accessed: 31- May- 2017].

[15] T. Parisi, *Programming 3D applications with HTML5 and Webgl*, O'Reilly, 2014, pp. 17-28.

[16] J. Ziolkowska and R. Reyes, "Geological and hydrological visualization models for Digital Earth representation", *Computers & Geosciences*, vol. 94, pp. 31-39, 2016.

[17] Krämer and Gutbell, "A case study on 3D geospatial applications in the Web using state-of-the-art WebGL frameworks", *Web3D '15 International Conference on 3D Web Technology*, pp. 189-197.

[17b] "Universal Transverse Mercator coordinate system" https://en.wikipedia.org/wiki/Universal_Transverse_Mercator_coordinate_system

[18]" Log ASCII Standard" http://www.cwls.org/las/ Online] [Accessed: 31- May- 2017].

[19] H. Hoppe. Smooth view-dependent level-of-detail control and its application to terrain rendering. Proceedings of the conference on Visualization 1998.

[20] "BVHRefiner", *X3DOM Documentation* https://x3dom.org/docs-old/tutorial/bvh.html. [Online] [Accessed: 31- May- 2017].

[21] "Terrengmodell Svalbard (S0 Terrengmodell)", *Data.npolar.no*, 2017. [Online]. https://data.npolar.no/dataset/dce53a47-c726-4845-85c3-a65b46fe2fea. [Accessed: 31- May- 2017].

[22] "GDAL - Geospatial Data Abstraction Library" http://www.gdal.org/ [Online] [Accessed: 31- May- 2017].

[23] "Angular Documentation, Architecture", *Angular.io*, 2017. [Online]. Available: https://angular.io/docs/ts/latest/guide/architecture.html, [Accessed: 31- May- 2017].

[24] "About | Node.js", *Nodejs.org*, 2017. [Online]. Available: https://nodejs.org/en/about/. [Accessed: 31- May- 2017].

[25] "Express - Node.js web application framework", *Expressjs.com*, 2017. [Online]. https://expressjs.com/. [Accessed: 31- May- 2017].

[26] "RESTful Web services: The basics", *IBM developerWorks*, 2017. [Online]. Available: https://www.ibm.com/developerworks/library/ws-restful/. [Accessed: 31- May- 2017].

[27] "Mongoose ODM v4.10.4", *Mongoosejs.com*, 2017. [Online]. http://mongoosejs.com/. [Accessed: 31- May- 2017].

[28] "Working with JSON data", *Mozilla JavaScript Manual*. [Online]. https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON. [Accessed: 31- May- 2017].

[29] J. Dirksen, *Learning Three.js: The JavaScript 3D Library for WebGL,*, 1st ed. Birmingham: Packt Publishing, 2015, pp. 7-35, 37-64.

[30] "X3DOM Volume Rendering" https://www.x3dom.org/volumerendering/ [Online] [Accessed: 31- May- 2017].