A Standalone Interface for Web-Based Virtual Reality of Calculated Fields

M. Jüttner, N. Zhao, and S. Grabmaier

University of Stuttgart - Institute for Theory of Electrical Engineering,
Pfaffenwaldring 47, 70569 Stuttgart, Germany

Introduction

The fifth-generation of the web standard HTML in combination with Cascading Style Sheets (CSS) and JavaScript (JS) enables a standalone cross-platform web application for the visualization of COMSOL Multiphysics® results [1]. Using it, teaching, product or research presentations as well as modern simulation engineering profit from the provided accessible and lightweight web-based visualization system without any dependencies on plugins and other external libraries. The applied web standard enhances the support for multimedia and graphics for every modern device and every actual web browser. An example is the Web Graphics Library (WebGL). It was introduced in HTML5 and enables the GPU acceleration for web applications. Using WebGL, even the realization of Virtual Reality (VR) within a web application gets possible. VR provides an even more immersive experience of three-dimensional (3D) simulation results and supports a better understanding of the numerical results. As an intermediate step, also the visualization at 3D-TVs gets possible.

In this paper, a modern and improved user friendly Graphic User Interface (GUI) is presented. Using upcoming web technologies, it provides an interactive VR interface to visualize COMSOL Multiphysics® simulation results. Thereby, it extends the existing standalone web-based visualization system for simulation results [1]. Its web-based implementation enables a usage across multiple platforms like mobile phones, tablet computer and 3D-TVs. Since no deployment is required, it is also applicable for head mounted VR displays. Here, details about its implementation and the updated visualization system are given. This includes details for rendering elements like the legend within the visualization and its adaption to hardware depended specifications.

The paper is organized in six chapters. While chapter 1 gives the introduction, chapter 2 describes the updated architecture and the components of the corresponding visualization system including the data preparation. Chapter 3 give details about the design of the web application. How stereo displays are supported is described in chapter 4. Chapter 5 shows evaluations of the implemented system. A conclusion and outlines to further work are given in chapter 6.

Visualization System

To visualize COMSOL Multiphysics® diagrams in an independent web application the system architecture shown in fig. 1 is used.

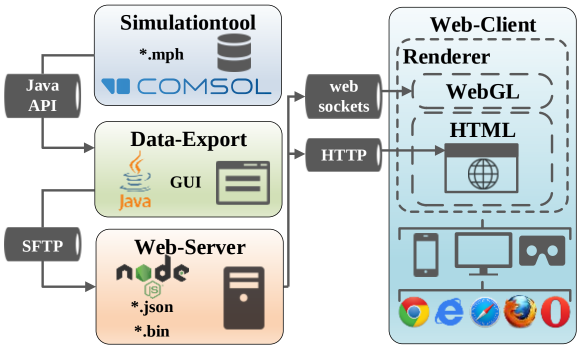
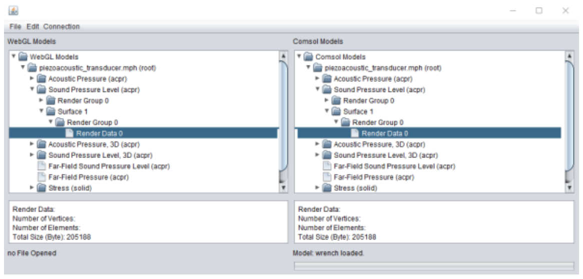


Figure 1: System architecture

To provide visualization at the web client, an administrator selects meaningful diagrams from multiple existing models and provide related data by the COMSOL Java API (CJAPI). To use the CJAPI, a Java application is created. The Java application accesses the model object and extracts meta information and raw rending data from the diagrams. Therefore, selected *plotGroup* objects and their rendering groups are evaluated. They contain the required numerical rendering data and additional meta information. Examples are vertices and point data as well as derived information like the bounding box calculated from the minimum and maximum values of the coordinates. Also, names, tags and data types defined within the COMSOL model are used based on the existing model structure. This enables the compatible handling of multiple models and different diagrams by this framework. The hierarchical model structure and the extracted meta information for the model "piezoacoustic transducer" from the model library are shown in fig. 2 within the developed Java application for the administrator.



```
WebGL Models
  piezoacoustic_transducer.mph (root)
    Acoustic Pressure (acpr)
    Sound Pressure Level (acpr)
      Render Group 0
      Surface 1
        Render Group 0
          Render Data 0
    Acoustic Pressure, 3D (acpr)
    Sound Pressure Level, 3D (acpr)
    Far-Field Sound Pressure Level (acpr)
    Far-Field Pressure (acpr)
    Stress (solid)

Render Data:
Number of Vertices:
Number of Elements:
Total Size (Byte): 205188

no File Opened
```

```
Comsol Models
  piezoacoustic_transducer.mph (root)
    Acoustic Pressure (acpr)
    Sound Pressure Level (acpr)
      Render Group 0
      Surface 1
        Render Group 0
          Render Data 0
    Acoustic Pressure, 3D (acpr)
    Sound Pressure Level, 3D (acpr)
    Far-Field Sound Pressure Level (acpr)
    Far-Field Pressure (acpr)
    Stress (solid)

Render Data:
Number of Vertices:
Number of Elements:
Total Size (Byte): 205188

Model: wrench loaded.
```

Figure 2: Java application

Enabling WebGL to render a diagram, the numerical data is converted as described in [1]. While the data type of the numerical values is adapted and stored as binary data, meta information including their connection to the binary data is handled separately in a JSON file. JSON is used as file format for data exchange because it is easily read- and writeable for humans as well as easily generate- and parseable for machines. Since JSON is a subset of JS [2], JS provides native support for receiving and parsing JSON format. Via the Secure File Transfer Protocol (SFTP), the files are uploaded to a common web server. As backend of the visualization system, it provides a web application and all necessary data for visualization. Node.js is used as web server since it is built on Google's JS runtime environment. Its event-driven architecture and asynchronous I/O capability provide good performance even for handling large data [3]. Initially, the webserver transmits the single page web application. The transmission of the binary visualization data is realized by WebSockets. Interactions are enabled based on requests handling.

Web Application

The visualization application is supposed to run on all modern devices that provide a nowadays web browser. HyperText Markup Language (HTML), Cascading Style Sheets (CSS) and JavaScript (JS) are considered as the primary technologies to build a web application. A Document Object Model (DOM) is used to describe logical relation between the applications structure elements. While loading such a web site, the browser parses the HTML file into a hierarchical DOM. This DOM is accessed and manipulated through CSS and JS. This enhance web sites to be more dynamic and interactive. To support various platforms and devices, a Responsive Web Design (RWD) [4] is done. As example for RWD, media queries introduced in CSS3 are used to check the capability of a device and define different style rules for each media type. They include blocks of CSS code within the website only, if a certain condition is true. Their syntax is

@media not | only media type and (media feature) {
CSS−Code ;},

where media types and features are predefined expressions. They are applied e.g. on the logo elements in the navigation bar. The logos appear in full size, when the application is accessed from a desktop computer with a maximized window. Only the icons are shown when the application is used at a mobile device. In fig. 3 the different layouts of the logos are given.

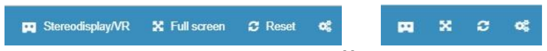


Figure 3: Logo element under different display mode

Another example is the font size that needs to be scaled in relation to the screen size to achieve a consistent experience from the GUI. Therefore, the units *rem* are used that were also introduced in CSS3. They specify the size of a CSS property relative to the size of a root element. In addition to CSS the layout is also altered by JS. For instance, when the size of the visualization window is reduced, menus are collapsed and replaced by a smaller navigation bar. During the process of collapsing and expanding the size of related elements are also recalculated and resized. In the same way, different display sizes are handled. This dynamic behavior is also applied for other components such as a switch for different display modes. For simplicity, no horizontal or vertical scrolling is featured except in such components like the property menus.

When loading the application, first a wire frame is created that contains menus to setup the visualization. Since the web server offers access to multiple models, the user selects one model and then the diagram to visualize. Now features like a reset of the visualization window, color table modifications, light effects and other user interactions are provided. Advanced options like a switch of the display to full screen or stereo display mode are always present. Rendering the COMSOLs data within the web application is done using WebGL. It is used in HTML5 *canvas* elements. So, interactive 3D graphics within compatible web browsers without any plugins are enabled. 2D diagrams are also possible and we decide to disable the rotation interaction. In fig. 4 the design elements of a web application using WebGL are shown. Elements added in comparison to a common website are highlighted.

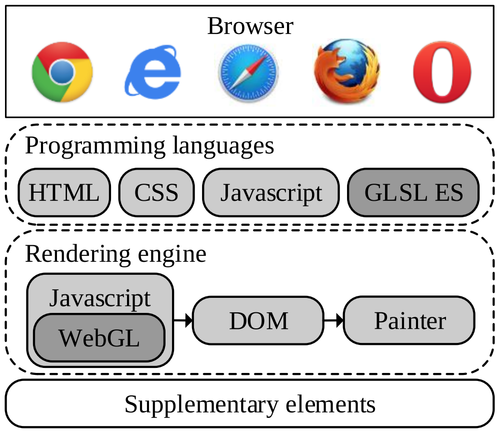


Figure 4: Web application elements

Rendering is done according to the pipeline defined by WebGL [5]. Within the pipeline, programmable vertex and fragment shader inherited from the Open Graphics Library for Embedded Systems (OpenGL ES) and programmed in the OpenGL shading language (GLSL ES) are executed on a graphic processing unit (GPU). Since several steps of the rendering pipeline are modified in chapter 4 to provide a stereoscopic visualization, a short introduction is given.

After initializing WebGL, the vertex shader is executed to process vertex arrays and buffer objects. It performs general operations like transformations and rotations on vertices. While 3D objects are handled with a perspective projection, an orthographic projection is applied for 2D objects. Examples for constantly present 2D objects are the coordinate indicator object and the diagrams legend. In addition, the vertex shader calculates the coefficient for the shading model per vertex based on the Phong reflection model. The primitive assembly converts the vertex stream into a sequence of base primitives. The base primitives are points, lines or triangles. If a primitive is not completely within the frustum of sight, clipping is needed. The frustum of sight is represented by a virtual camera with its point of view, its near and a far clipping plane, its screen ratio and its four borders at the sides. The rasterization step converts the base primitives to fragments that represent pixels and can be displayed on screen. It also blends assigned colors to the fragments. The following fragment shader process fragments. This includes several tests like the scissor test, the stencil test and the depth test to ensure a correct visualization. Afterwards per fragment operations are performed before the final pixel array is stored in the framebuffer.

To obtain a diagram, the prepared binary data and meta information are provided by the web server. Since WebGL 1.0 supports only 16 bit indexing, the data segments are split in in 65k blocks at the web client. The actual draft of the WebGL 2.0 specification currently supports 32 bit indexes. This splitting holds for values e.g. vertices and their attributes. Once the raw WebGL data is prepared, the rendering process is executed. During the rendering process, the performed transformation also changes the coordination system. Like after clipping the coordination system is normalized. Data is finally projected to the screen by performing the viewport transformation. To do so, a viewport is defined by two coordinates and the screen size. In case of multiple viewports, the data in the buffer objects is reused for rendering in one canvas element to reduce the calculation effort. To generate text content within WebGL, texture mapping is applied. The related 2D canvas is mapped to a target point. This billboard or point sprite technique faces the text to always point toward the viewer. Based on the same principle and in combination with an orthographic projection the color legend is created. For creating it labels meta information from the JSON file are used. Fig. 5 shows the resulting GUI for the wrench of the model library.

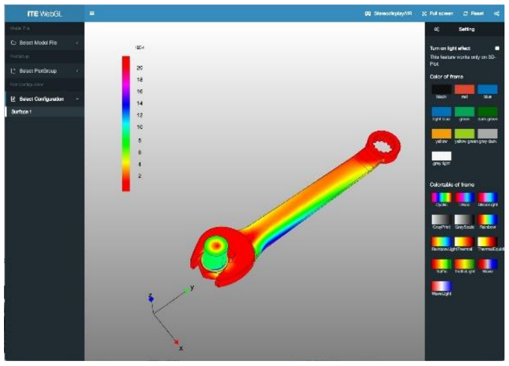


Figure 5: Visualization of a wrench within the web application on a desktop computer

Stereo Displays

Using 2D images to create a 3D experience for a user is the idea of stereo displays. In addition to the 2D information also depth information is provided. Interaction get possible with controllers like the mouse, the keyboard, touch inputs, sensors like an accelerometer and a gyroscope or techniques like motion capturing, eye tracking and speech recognition. In the following the fundamentals for stereo displays are given.

The binocular vision describes the physiology to see with two eyes. Because of the interpupillary distance between the eyes we simultaneously see two slightly different perspectives of one scene. The disparity of two views is fused to generate the 3D stereopsis impression [6]. The eyes accommodation and convergence enables us to automatically see near and far objects clear and without diplopia (double visions). Diplopia obviously depend on the eyes focus. When the eyes converge at the same point, it has zero parallax. Objects located behind this point have positive parallax and vice versa [7]. The amount of parallax also helps our brain perceive the depth and distance information [6]. Supplementary depth information is extracted from the monocular impressions e.g. curvilinear perspective, known details, relative scale, occlusion, aerial perspective, texture gradients, lighting, shadows and relative motion [7].

The considered high definition 3D-TVs display video streams that were recorded side by side (SBS). For displaying, each side is straightened to the full size and interlace with a shared frequency. Using polarized or shutter glasses the two displayed images are alternatingly conveyed to the viewers eyes. Displaying them on a 16: 9 TV screen with 1920×1080 pixels, the interlaced representation requires two figures with 960×1080 pixels (8: 9) each. To create a SBS stereo image multiple methods exist. In the web application, an adapted parallel view method is applied [8]. A second viewport is set up to render the second image based on the shared content. Each viewport contains a virtual camera perspective facing parallel to each other. The location of the rendered objects in each view frustum shift relative to the shared centerline of the virtual camera in the opposite direction. Due to the parallel camera views the keystone effect is avoided that is cause by trapezoid projections like in the case of two cameras with non identical projection planes focusing on a single point (toe-in method). Considering that 3D-TVs straighten the images their width is reduced to a half. For static objects no relative motion and scaling is needed. Objects in front of the projection plane will be experienced as they pop out of the screen and vice versa. The cameras separation is set equal to the separation between the two presented images. Since this separation and the distance between the two cameras and the projection plane majorly influences the impression of depth their relation is estimated as $\leq 1/20$. For a comfortable view, the negative parallax is supposed to be smaller than the interpupillary distance. Otherwise eye strain and diplopia occurs. The angle of parallax

$$\Theta = 2\,\mathrm{atan}\left(D_{Obj}/D_{view}\right), \tag{1}$$

with D_{Obj} as horizontal separation between the objects and D_{view} as their distance to the screen. It is advised to be $\Theta \leq \pm 1.6°$ [7]. Therefore, the user is enabled to adjust the separation with keyboard keys $\boxed{\leftarrow}$ and $\boxed{\rightarrow}$. Further interaction like a zoom or a shift of the model are also provided by using the mouse. An additional layer with different parallaxes is added to supports the viewer to distinguish between 3D and 2D contents like a legend or subtitles. Fig. 6 shows the 3D visualization of the wrench from the model library within full screen mode at a LG 65UB950V including its color legend.

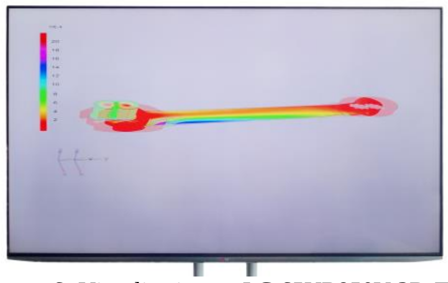


Figure 6: Visualization on LG 65UB950V 3D-TV

Extending the 3D experience to be more realistic and enhance is the idea of Virtual Reality (VR) environments [9]. To do so, classical input techniques are replaced by various technologies such as head tracking, gesture recognition and motion capturing. Here, the abilities of the visualization system are extended to response to the user's interaction by supporting VR head mounted displays (HMDs). This is of special interest, since HMDs enable a low-cost experience of VR within areas of limited space like an office for a product developer or a simulation engineers. For visualization the basic principle is similar to 3D-TVs. The SBS projection divide the screen. For a screen with 1920×1080 pixels this results in two areas with 960×1080 pixels each. A higher resolution enables more impressive and realistic VR experience. Within HMDs convex lenses are used to break the restriction of close object. The virtual image forms according to thin lens formula

$$1/d_{Foc} = 1/d_{Im} + 1/d_{Obj}, \tag{2}$$

with d_{Foc} as focal length, d_{Im} as distance from the lens to screen and d_{Obj} as distance from the object to the lens. For Googles Cardboard the focal length is denoted as $d_{Foc.} \approx 45$ mm, the distance from the lenses to the screen as $d_{Im} \approx 38$ mm and the resulting image distance as $d_{Obj} \approx 250$ mm [10]. This fits to the least distance of distinct vision (LLDV) of a young adult with normal vision. This also represents minimum comfortable distance between an object and the eyes. The related magnifying power

$$M = d_{Obj}/d_{Foc} \tag{3}$$

of the lens is $M \approx 5.5$, so objects appear in a certain distance instead of right before the eyes. Using a smartphone as screen within a HMD, its sensors get available to handle at least 3 degree of freedom (dof). This is feasible for orientation within the visualization by using head tracking. Using input devices with 6 dofs also translation and rotation can be handled. Combining head tracking with the rendered spherical visual scene an enhanced experience of VR is created. *DeviceOrientationEvents* are used for implementation within the web application. They define several DOM events that enable reactions on the motion and orientation of a device [11]. The introduced coordinate system is shown in fig. 7.

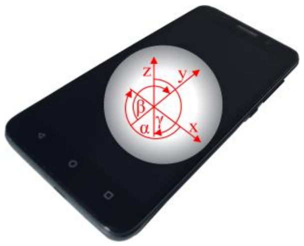


Figure 7: Coordinate System of a mobile phone

The x-axis is in the plane of the screen and points to the right. A rotation around x-axis is given as β with $-180° \leq \beta \leq 180°$. The y-axis is also in the plane of the screen and points to the top. A rotation around it is evaluated as γ with $-90° \leq \gamma \leq 90°$. The z-axis is perpendicular to the screen and completes the right-angled coordination system. The related rotation angle is α with $0 \leq \alpha \leq 360°$. Since a mobile device is used in the HMD and it is vertically mounted, an adequate input matching is required. This enable the user to see the visualization from different perspectives. Fig. 8 shows the visualization at a mobile phone with and without the stereoscopic mode for HMDs.

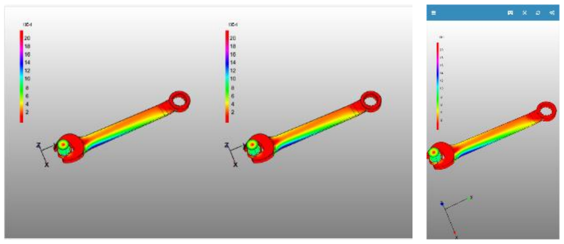


Figure 8: Mobile and stereoscopic visualization

Evaluation

Using hardware within a VR system, the latency and the frame rate are two critical factors for generating a realistic impression. For a comfortable usage of a HMD, the latency is limited to 20 ms [12]. Therefore, the refresh interval of the canvases and the sample interval of the gyroscope sensor is set to 17 ms. Depending on the complexity of the displayed scene and the performance of the smartphone, the frame rate can significantly decrease. By our experience, a discomfort usage occurs below 30 Hz or in case of a noticeable delay between motion and visualization. This can also lead to motion sickness. To achieve a good VR experiencing, a frame rate above 60 frames per second (FPS) and a low latency between motion and display updates is provided. Therefore, visualization updates are forced when the model is moved or its orientation changes. Results given in table 1 for $77 \cdot 10^3$ vertices prove this for the Feeder Clamp model at several ordinary mobile phones, different operating systems and various internet browsers.

Device	Browser	Screen (pixel)	FPS
Mate 9 Pro	Chrome	2560×1440	60
Mate 9 Pro	Firefox	2560×1440	60
iPhone 6 +	Chrome	1920×1080	60
iPhone 6 +	Safari	1920×1080	60
iPhone 6	Chrome	1334×750	60
iPhone 6	Safari	1334×750	60

Table 1: Web application performance at several mobile phones

Fig. 9 shows the evaluation of the calculation time within 17 ms on a mobile phone during visualization updates. Here, the two viewports refresh synchronously by invoking the draw function. Since the second viewports differ only in the horizontal position, the cost for its drawing is small. As shown, the website including its rendering performed well and the remaining idle time provide a buffer for additional tasks.

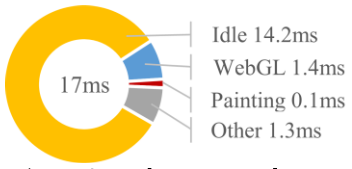


Figure 6: Performance evaluation

Conclusion

Here, a standalone interface for web-based virtual reality was presented and details were given for the underlying standalone visualization system. The developed user-friendly web application was implemented based on responsive web design rules and by applying nowadays web techniques. Details for the data handling and the rendering were given to visualize COMSOL Multiphysics® simulation results. The web application runs on multiple platforms like mobile phones, tablets, desktop computers and 3D-TVs in combination with a common browser as shown in fig. 7.

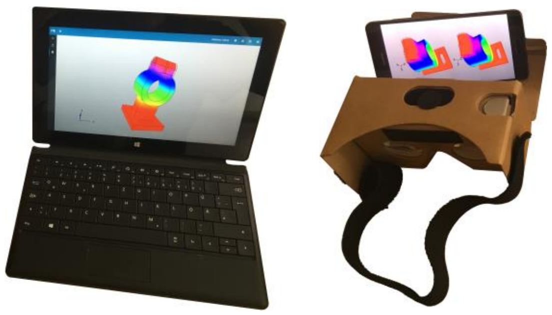


Figure 6: Application examples

No additional plugins or other external libraries are needed. The provided stereo display mode is used to display a stereoscopic image on a 3D-TV or on a head mounted display. This is of special interest, since these techniques enable a low cost and much more powerful experience of the simulation results, even in areas of limited space like an office. The presented figures and the evaluations proves the functionality of the developed interface and the visualization system. Further improvement of the VR experience is expected by following the pre-released *WebVR* specification by the World Wide Web Consortium (W3C). Simplified calls of the viewports and perspective manipulation are announced as well as optimized display effects like distortion correction. Also support for further and more intuitive input devices outline a better VR experience.

References

[1] M. Jüttner, S. Grabmaier and W. M. Rucker, Web Based 3D Visualization for COMSOL Multiphysics, Cambridge, UK: European COMSOL Conference, 2014.

[2] D. Crockford, "The application/json Media Type for JavaScript Object Notation (JSON)," *Request for Comments: 4627*, 2006.

[3] S. Tilkov and S. Vinoski , "Node.js: Using JavaScript to Build High-Performance Network Programs," *IEEE Internet Computing* , vol. 14, no. 6, pp. 80-83, 2010.

[4] E. Marcotte, Responsive Web Design, A Book Apart, 2011.

[5] D. Ginsburg, B. Purnomo and D. Shreiner, OpenGL ES 3.0 Programming Guide, Addison-Wesley Professional, 2014.

[6] I. P. Howard and B. J. Rogers, Binocular vision and stereopsis, Oxford University Press, 1995.

[7] D. A. Southard, "Transformations for stereoscopic visual simulation," *Computers & Graphics* , vol. 16, no. 4, pp. 401-410, 1993.

[8] R. R. Hainich and O. Bimber, "Displays: fundamentals & applications," CRC Press, 2016.

[9] G. Saggio and M. Ferrari, New Trends in VR Visualization of 3D Scenarios, Virtual Reality - Human Computer Interaction, InTech, 2012.

[10] Google Inc., 1600 Amphitheatre Parkway, Mountain View, CA 94043, USA, *Google Cardboard*, 2014.

[11] R. Tibbett, T. Volodine, S. Block and A. Popescu, "DeviceOrientation Event Specification," *W3C Working Group* , 2017.

[12] R. Yao , T. Heath , A. Davies , T. Forsyth , N. Mitchell and P. Hoberman , "Oculus VR Best Practices Guide," Oculus VR, LLC , 2014.