VRML to WebGL Web-based Converter Application

Fatma Ziwar and Rimon Elias
The German University in Cairo, Egypt
{fatma.ziwar, rimon.elias @guc.edu.eg}

***Abstract*—In this paper we propose a method to convert a VRML file into a WebGL rendered scene through an online application that is capable of rendering this uploaded VRML file within the Web browser itself without any plug-ins being installed. The conversion process between the input VRML file and the WebGL using our Web-based converter application is done by analyzing the uploaded VRML file content represented in nodes. These nodes are then converted into scene graph nodes used for the scene creation. A scene tree is then initialized and populated using these nodes containing their info and sent to the proposed dynamic WebGL template. Finally a new HTML5 file is constructed and displayed on the user's Web browser. An extensive set of experiments are conducted to show the performance of the online converter application.**

***Keywords*—VRML, WebGL, Web technologies, data structures, HTML5, Web-based converter, 3D vector graphics, Web-based graphics library**

I. INTRODUCTION

Until recently, Web browsers were not able to directly draw graphics on the Web without using additional plug-ins like Flash, or file viewers. Rendering rich computer 3D graphics directly on a Web page was a dream of many developers. But, as the Hyper Text Markup Language fifth major version (HTML5) arises it came up with the help for solving this problem, by having the canvas element embedded into its context. Thus graphics can be rendered without any helper programs, Java applet or even the usage of an additional plug-in, only by using the Canvas and Scalable Vector Graphics (SVG) elements.

The usage of HTML5 and additional scripts like JavaScript containing WebGL running on an Internet-connected computer and a WebGL enabled browser allows for faster drawing graphics onto the Web permitting the generation of 3D graphics and for automated memory management [1]. Currently there are many modern Web browsers that support WebGL, for example: Google Chrome, Mozilla Firefox, Safari, Opera and recently Internet Explorer IE11 [2].

Software converter applications can be Web-based applications in which the user or the developer can have its access through the Web browser being used directly, or as a stand-alone computer program. An existing stand-alone VRML converter applications example: Ayam which is a free open source 3D modeling environment [3] used to export and import files written in a way that can be read by human easily. These files are known for the RenderMan Interface Bytestream (RIB) files. To render the scenes modeled by Ayam, a RenderMan compliant renderer is required to be installed. Example of Web-based converter applications include VRMLMerge that runs as a Java applet inside the Web browser, a command line tool, or as a desktop application [4] and it is used to convert VRML'97 files to X3D files. Other examples include Instantreality X3D encoding converter that allows the developer to produce new applications and virtual worlds by modeling and not just programming [5].

In this paper, the first-ever VRML-WebGL converter is created. It performs the conversion between 3D virtual world graphics written in a VRML file and WebGL, thus allowing these 3D graphics to be drawn and rendered directly on the Web without using additional plug-in or file viewer. We succeeded in developing a conversion technique that is capable of successfully editing a WebGL template by sending the parameters and objects of the requested graphics to be included in the HTML5 context of the output Web page.

The application applies the different requested operations onto the requested objects. It does data acquisition, processing and conversion process between VRML and WebGL contexts. Also a file viewer is called to view the VRML input file graphical output on the Web browser to compare the two graphical outputs.

The paper is organized as follows. In the next section the problem is formally defined. Our approach is explained in Section III while experimental results and performance analysis are presented in Section IV. Finally, the paper concludes in Section V.

II. PROBLEM DEFINITION AND MODELING

VRML is an easy to write modeling language used to represent SVG. These graphics could be then viewed on a standalone application like a player or a viewer or onto a Web browser by using an additional plug-in. On the contrary, the main functionality for using WebGL is to represent SVG directly onto the Web browser. Unfortunately, there is no tool to convert from VRML to WebGL. Thus, we created our converter to help in filling the gap of the conversion between VRML and WebGL.

In this paper, the first-ever VRML-WebGL converter is presented and discussed (Section III). Not only the WebGL written inside the JavaScript context helped us in this conversion but also the new features available in HTML5 and the new open source modern Web browsers that support WebGL. We succeeded in developing a conversion technique that is capable of successfully editing a WebGL template by sending the extracted parameters and objects of the requested graphics to be included in the HTML5 context of the output Web page. Being a Web-based application, it can run directly onto the Web browser page, and

978-1-4799-5807-8/14/$31.00 @2014 IEEE

could run on wider range of hardware and operating systems platforms; so, it can be used by more developers and/or users.

III. SOLUTION APPROACH

Our solution approach includes both the client-side and server-side of this application in terms of implementation and usage.

A. *Solution overview*

To convert the VRML file to WebGL scene, the file has to be uploaded to the application while being opened at the client-side using a WebGL enabled Web browser. The server-side component then checks if it is a real VRML file, parses it, and gets its content. Then it creates the required objects according to the requested parameters and creates the file scene tree according to the discovered relationships. Then it restructures the objects into scene graph tree nodes, links the created nodes to the tree, and extracts the required information needed to draw the scene in WebGL using tree nodes depth-first in-order traversal. It creates the required string object to be sent to the proposed generic WebGL template, modifies it and finally opens two new tabs representing the two graphical outputs for VRML uploaded file using a plug-in and the final WebGL scene.

The converter client-side and server-side steps discussed above could be summarized as shown in Fig. 1.

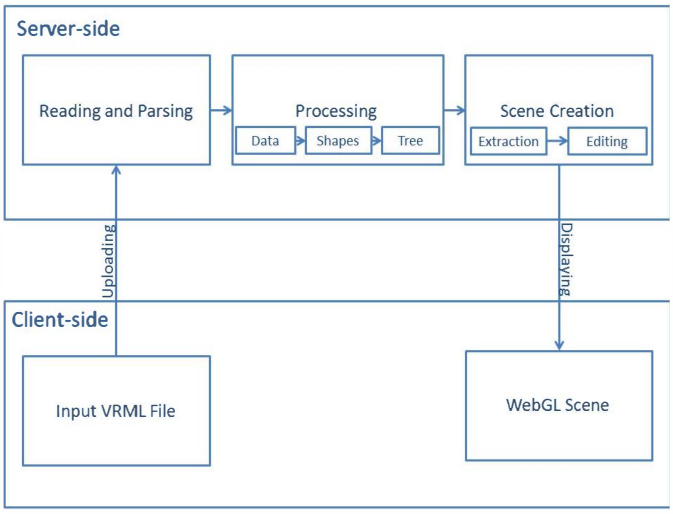

Figure 1. Solution overview

The five main blocks and their process steps representing the idea behind the converter usage and functionality over the VRML input file content (shown in Fig. 1) are discussed in the following sections.

B. *VRML File Uploading, Reading and Parsing*

After checking the file content as being written in a correct VRML format, the server then saves it and proceeds with parsing its content. The parsed content is then put into one string format to be processed.

C. *Content Processing*

At this process step, the scene tree is created. To extract the VRML nodes information, the string format resulted in the last step while parsing the file (Section III B) is restructured to an array of words to allow the server-side component to extract the VRML nodes. The words of curly braces character positions are then searched for in the words array. These parent curly braces are kept track of their character positions, then the nested curly braces pairs' character positions are being fetched and saved.

Data collection is then done through using these nested curly braces pairs; the VRML objects and their corresponding fields' values are extracted, and the required parameters for WebGL are to be determined accordingly.

To create the objects determined from the string of the VRML input file, the server-side implemented classes are instantiated to represent the requested scene objects.

The scene object is the base object for all the other objects. The transform and shape objects are its direct children. Box, sphere, cone and are the direct children objects to shape. Also appearance is a direct child object to shape, and its direct child is material.

Tree population is then done using the data collected.

These steps could be summarized as shown in Fig. 2.

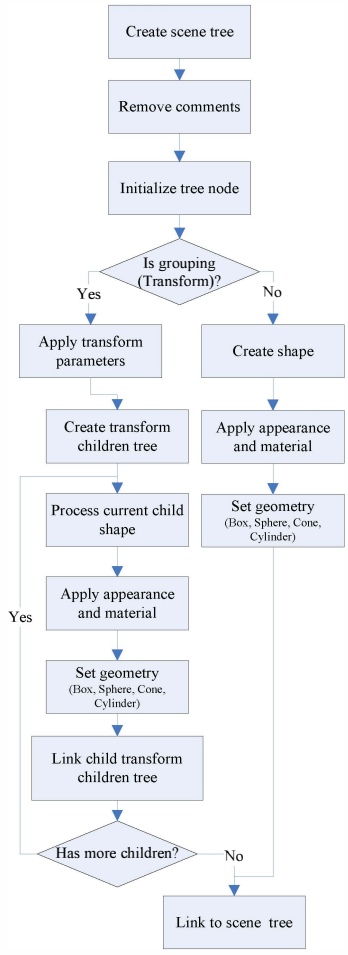

Figure 2. Content processing

D. *Scene Creation in WebGL*

The constructed scene tree (Section III C) is then in-order traversed to get the needed parameters to be sent to the WebGL template. Then parsed and the requested shapes are created accordingly. These steps could be summarized as shown in Fig. 3.

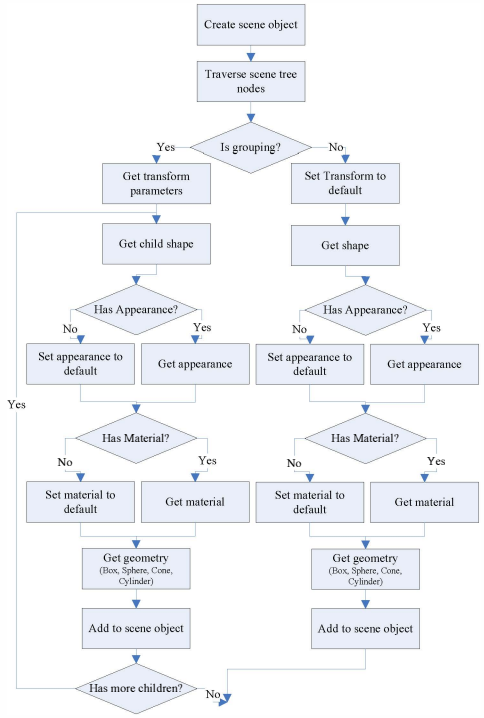

Figure 3. Scene creation

E. *Parameters parsing and passing*

The latest HTML5 technology capabilities were utilized in the process of generating the dynamic WebGL template document and in having the required corresponding WebGL as an embedded code inside the JavaScript in its header. This is done through using the constructed scene string (Section III D) for rendering the final WebGL graphical output scene on the client's Web browser as shown in Fig. 4.

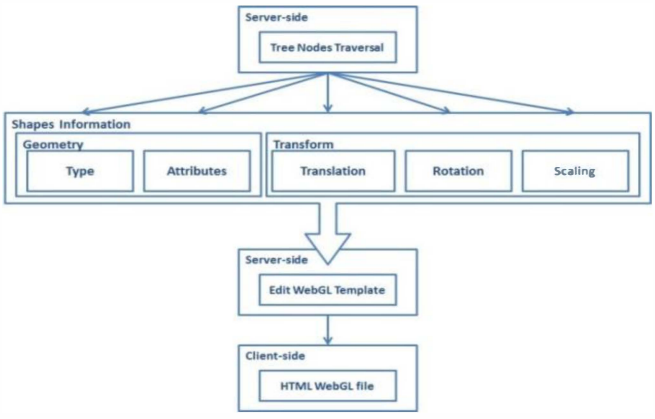

Figure 4. Sending tree content to WebGL template

In this step the server sends the whole scene objects constructed string to the WebGL template. The WebGL Template structure is shown in Fig. 5.

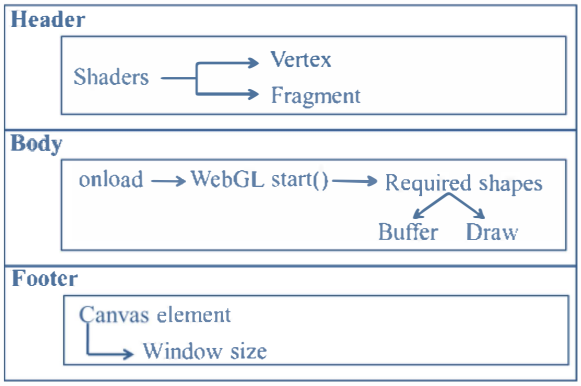

Figure 5. WebGL Template structure

Then the scene shapes and their info are buffered in a big array. The shape info cells includes: string shape type, array of fields' values, translation attributes, rotation vector, and rotation angle in radians, scaling factors, and color in RGBA. Each buffer row obtained through tree traversal is then sent to the WebGL template as shown in Fig. 6.

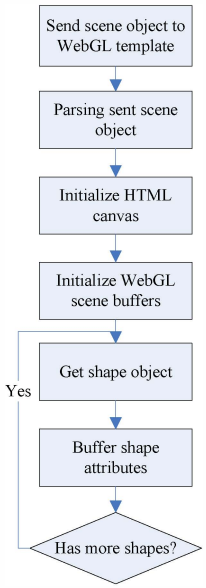

Figure 6. Sending scene objects and buffering

F. *Scene Display*

To display the scene on the client's Web page the following process is done inside the WebGL template.

First the WebGL template body (shown in Fig. 5) which contains the canvas element, the display window size, and the onload initiates the WebGL scene rendering process. Then the scene shaders and lights required for each shape object are formed by binding the shape parameters to the corresponding buffers and initialized. Finally the vertex attribute pointer is initialized, and the scene is drawn as shown in Fig. 7.

In addition to the original Web application tab and VRML scene tab, a new tab will be opened by the server to display the WebGL scene in it.

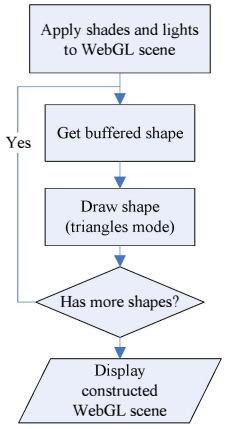

Figure 7. Scene display

IV. EXPERIMENTAL RESULTS

In this Section, experimental results are discussed by observing the conversion of an input VRML file representing a room scene in (Section IV A). Then, five experiments were held to examine the application's performance in Section IV B.

A. *VRML Room Scene*

The original VRML file uploaded is composed of a scaled and transformed clock, chair, desk, and a small car toy on it. The room sides are colored as: dark blue floor, white ceiling, yellow front wall, while the two side walls are red. As for the furniture: the chair is colored in blue, the desk's upper side in green, while its legs are in blue. The VRML file was converted and the WebGL scene was displayed within a total time of about 600 milliseconds. The WebGL scene output is shown in Fig. 8.

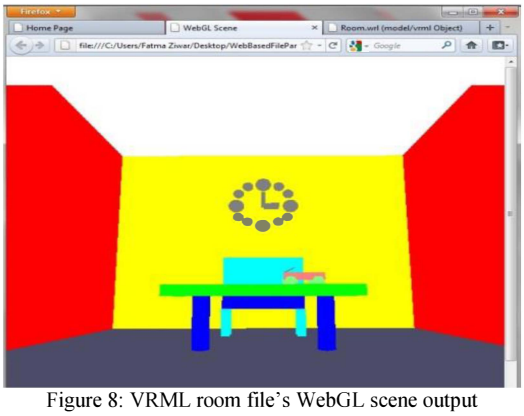
Figure 8: VRML room file's WebGL scene output

B. *Performance Analysis*

The following Experiments were held on a computer with Intel Core i5 2.67GHz processor, running on 4GB RAM, with ATI Radeon Graphics card.

1) *Experiment 1*

This experiment was held to discover the conversion processing time changes while incrementing the input VRML file size. While testing, ten VRML input files were created to have file size of scale range between 5 and 50 KB on a scale difference of 5 KB.

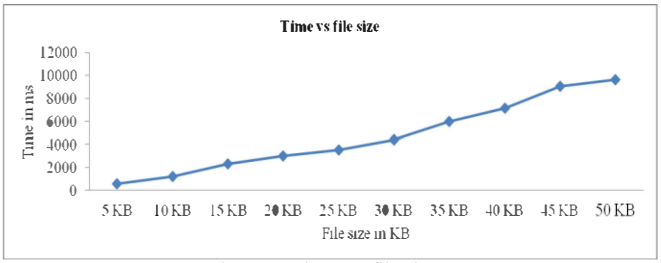

Figure 9. Time vs. file size

By observing the time line (shown in Fig. 9), it was found that the total conversion process time increases almost linearly while incrementing the input file size by a fixed interval. This is due to enlarging the amount of input data in the VRML input file.

2) *Experiment 2*

This experiment was held to show each conversion step contribution in the total file conversion time. The main conversion algorithm's six steps are as follows: reading VRML file content, removing comments, constructing scene tree, getting required shapes pairs, populating tree, and traversing and sending data. Information records were collected as a result of converting four VRML files each containing one shape: box, sphere, cone, and cylinder. Results were obtained by recording their cumulative conversion time per conversion process step and observing the results for each shape.

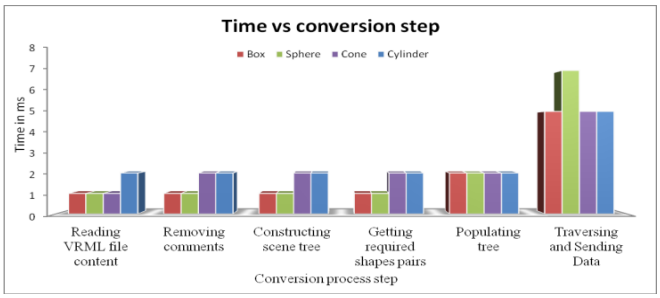

Figure 10. Time vs. conversion process step

Two observations (in Fig. 10) were found: The longest conversion time was recorded for the last conversion process step; traversing and sending data. This is due to the calculation time consumed in the WebGL constructed file for the requested polygons to be rendered on the screen. The longest input file's total conversion time was recorded for the sphere shape with a cumulative time of 7 milliseconds. This is due to the fact that creating a spherical surface requires the largest number of WebGL triangles among all shape primitives (i.e., box, cone, cylinder and sphere).

3) *Experiment 3*

This experiment was held to discover the relationship between the input VRML file shape type and the corresponding total

conversion time. The obtained results were recorded according to the average conversion processing time per requested shape in the VRML input file. It is done through converting four VRML files each containing ten shapes from the same type: box, sphere, cone, and cylinder. We used multiple shapes of the same type to magnify the consumed time differences between types, if any.

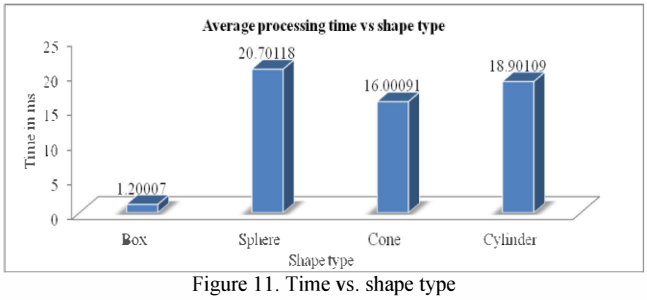

Figure 11. Time vs. shape type

By observing the time shown in Fig. 11, it was found that the longest total conversion time was for the sphere shape with average total time of 21 milliseconds, while the smallest record was for the box shape. Again, this is a result of the method by which faces are drawn in WebGL. A face or a polygon is drawn as a set of adjacent or neighboring triangles. In that sense, a box is composed of 6 faces where each face surface can be drawn as two triangles. On the contrary, to render the spherical shape surface, it should be divided into slices representing the triangles from top to bottom. These surface triangles depend on three parameters: latitude, longitude, and radius of the sphere to be drawn. All of the points of intersection between the longitude and latitude curves are to be determined on the surface. Those points are grouped and treated as quads, which can be further split as triangles as done with box faces. Hence, execution time differs clearly.

Conical and cylinder surfaces are divided into hollow rings represented as circles by calculating the point coordinates to be drawn onto the specified circle circumference. Those points are used to create the required triangles. Finally, the top and bottom circles of the cylinder and only the bottom of the cone surface are filled.

4) *Experiment 4*

Two experiments were held to discover the amount of conversion processing time change that will happen while using the grouping node transform. The results were recorded according to converting four VRML input files containing different shape types, while depending on two properties; keeping the file size constant at 30KB, and keeping the number of shapes constant at 200. For each of them there are two converted files one with transformation and the other without transformation.

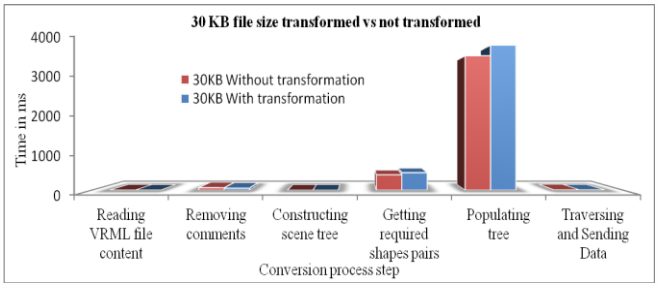

Figure 12. 30 KB transformed vs not transformed

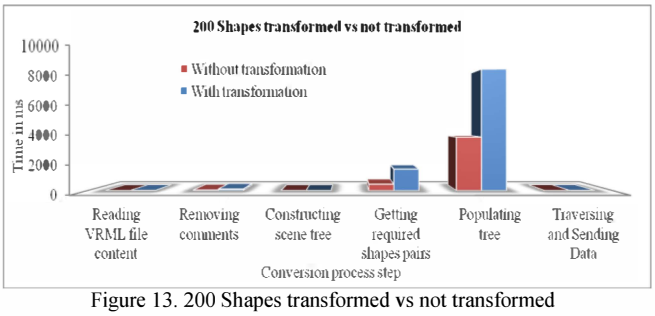

Figure 13. 200 Shapes transformed vs not transformed

By observing the time shown in Fig. 12 and Fig. 13, it was found that while having a larger file size and/or greater number of shapes, populating tree conversion process step consumed the largest amount of processing time. Using transform nodes result in more total amount of conversion time consumption than without using it.

5) *Experiment 5*

This experiment was held to get more information about the relationship between the conversion time and number of children in the VRML file and/or number of transform nodes in the file.

The results were recorded according to converting four VRML input files. Each of the first two files has one transform node while each of the last two files has 10 box shapes.

In addition to the transform node, the first file has one box child while the second has five box children. The third one has two transform nodes; each with five children. Finally, the fourth file contains ten transforms; each of them has one box child.

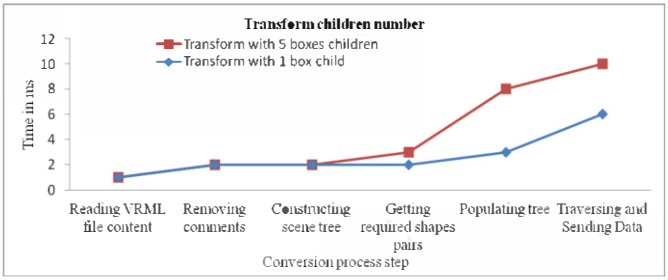

Figure 14. Transform children number

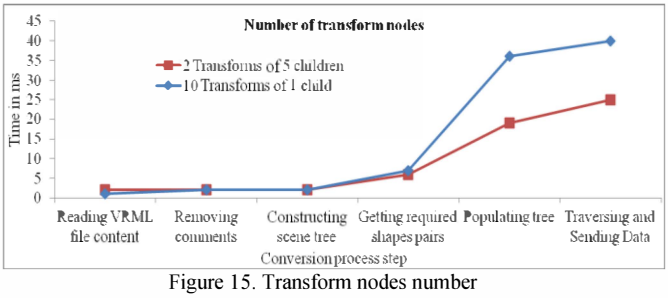

Figure 15. Transform nodes number

By observing the time shown in Fig. 14 and Fig. 15, it was found that the total amount of conversion time increases in both cases: increasing transform children number, and increasing the transform nodes number.

V. CONCLUSION

A. Problem

Many developers write VRML files to represent their graphical scenes. Unfortunately, these graphics should be displayed using a plug-in or a file viewer. With the appearance of modern Web browsers and their current evolution and continuous enhancements in having more advanced capabilities and HTML5 strong support, rendering rich computer graphics on the Web became easier.

As a result of introducing the recently developed JavaScript/OpenGL integration the new computer graphics library WebGL, proposed by Mozilla and Khronos, provides an API based on the popular and widely used 3D graphics standard OpenGL ES2.0 for the first time in March 2011. This is driven by the similarities between WebGL and OpenGL style of coding, also because WebGL is supported by many browsers which are members of the Khronos consortium's WebGL Working Group, and the ability to directly render 3D graphics onto the Web browser.

It is natural to think about the migration process from VRML to WebGL. But, unfortunately, there was no available converter to help in this migration.

B. Solution Summary

In this paper's presented work, we succeeded in implementing and developing a VRML to WebGL novel conversion tool. We created an easy-to-use Web application that the user is only requested to upload the VRML input file by clicking on two buttons. As a result the WebGL scene is sent to the Web browser and displayed immediately without having to install any plug-in to view it.

The step in between the uploading and the displaying of the final scene is the conversion process. This takes place on the server-side of the application by reading the file contents to process. This is followed by extracting the required shapes to be drawn along with their parameters, and finally sending this whole scene object to the newly proposed WebGL template. The requested scene is drawn on a new browser tab opened by the server.

REFERENCES

[1] Khronos Group. WebGL - OpenGL ES 2.0 for the Web. http://www.khronos.org/webgl/ Retrieved 15 December 2013.
[2] Microsoft Developer Network. WebGL. http://msdn.microsoft.com/en-us/library/ie/bg182648(v=vs.85).aspx Retrieved 15 December 2013.
[3] R. Schultz. Ayam. http://ayam.sourceforge.net/ayam.html. Retrieved 15 December 2013.
[4] VrmlMerge - Features. http://www.deem7.com/vrmlmerge/features.php Retrieved 15 December 2013.
[5] J. Behr, Fraunhofer Institute for Computer Graphics (IGD). Features and Benefits. http://www.instantreality.org/story/what-is-it/. Retrieved 15 December 2013.