An introduction to WebGL

Jing Guo

1. Introduction

Rendering 3D object in websites is becoming more and more popular as the number of web-based applications grows significantly. An interactive 3D object is especially important for online gaming and shopping, where the graphics in the web application needs to be realistic and fast-loading. However, the plugin-based solution for 3D objects rendering is not sufficient for the highly growing demands. The plugin-based solution usually requires additional installation and is developed for specific browsers, which provides poor flexibility and scalability. Hence, a light-weighted plugin-free solution to render 3D objects is greatly needed. In this paper, we focus on web graphics library (WebGL), a Javascript API for rendering interactive 3D and 2D graphics. We would describe the WebGL in session 2, and detail the implementation of WebGL through a simple example in session 3, and then discuss the advantages and limitations in session 4 and 5. Finally, we would draw our conclusion and talk about future works in session 6.

2. What is WebGL?

WebGL provides Javascript APIs for rendering interactive 2D and 3D graphics directly in the browser. The use of WebGL does not require installations of any plug-ins, however, it needs browser support. Currently, WebGL is supported by Firefox 4+, Google Chrome 9+, Opera 12+, Safari 5.1+ and Internet Explorer 11+. Through the `<canvas>` tag, WebGL enables the web content to call the provided APIs based on OpenGL ES 2.0 to render graphics in the web app. WebGL uses Javascript to control the invocations and pass the variables to computer's Graphics Processing Unit (GPU), where the code is executed. WebGL can access the hardware-level units to accelerate the rendering process and also has access to all DOM interface so that it can be easily mixed with other HTML elements.

WebGL is a low-level language which is built upon OpenGL ES 2.0. It passes the code to computer's GPU to process. To date, several frameworks and libraries have been developed on

top of WebGL to provide a simpler way to program WebGL applications [2, 3]. Usually, these libraries wrap the WebGL functions to create elements that are more straightforward and intuitive, such as modeling 3D graphics as a scene, a light source, etc [1]. The implementations of these libraries differ from each other, however, the fundamental idea of modeling 3D environment remains the same. Besides the easy-use feature, these top-level libraries also provide easy ways for event handling.

3. Implementation of WebGL

Like many other 3D graphics scheme, WebGL uses triangles as building blocks to render complicated graphics. Briefly, WebGL provides Javascript APIs to get information regarding these triangles, such as position, color, texture, etc. Then, WebGL feeds the collected information to GPU, where the information is actually processed. Finally, WebGL gets the computed results and renders the scene. The implementation of WebGL involves the direct connection with hardware-level architecture, so that WebGL is quite flexible for high-level frameworks, and it's highly scalable and provides competitive performance. In this paper, we would walk through a simple example to demonstrate the pipeline of how WebGL works.

```
1  <body onload="start()">
2    <canvas id="glcanvas" width="640" height="480">
3      Your browser doesn't appear to support the
4      <code>&lt;canvas&gt;</code> element.
5    </canvas>
6  </body>
```

Figure 1. Create a <canvas> tag in HTML for WebGL to render. The code is taken from reference [1].

Before initiating WebGL, a `<canvas>` tag needs to be created in HTML file. WebGL renders graphics to the web through this `<canvas>` tag. After creating the tag, WebGL would be initialized through the onload function, `start()`. During the initialization, we create a global variable to hold the reference of the canvas object that we want to render graphics to.

Next, shaders are initialized using OpenGL ES shading language. In WebGL, there are two kinds of shaders, the vertexShader and the fragmentShader. The vertexShader takes care of the

computations on the vertexes. It computes projected positions of the supplied vertex, and can also compute other attributes, such as color and texture. In WebGL, the individual pixel in the graphics is named as a fragment. The fragmentShader establishes the attributes for each fragment in the scene. Currently, in the higher-level frameworks and libraries wrapping WebGL provides shaders that are ready to use. However, users could also construct customer-tailored shaders.

After initializing the shaders, WebGL starts rendering the environment. The rendering process starts with creating vertex arrays, which contains information of the vertices, such as position, color, texture, etc. The vertex arrays are created in Javascript and provided from the web. After receiving the vertex arrays, the GPU would create a vertex buffer to store the vertices. Then GPU reads each vertex and feeds it to the defined vertexShader function. Next, GPU collects the computed attributes and connects the projected vertices to form triangles, which are the fundamental building blocks in graphics. Then, GPU feeds each triangle to the rasterizer for post-processing. The rasterizer breaks each triangle into pixel-sized fragments and then passes them through the fragmentShader. The fragmentShader outputs attribute for each pixel and the result is sent to the framebuffer. The graphics are then finally rendered from the framebuffer.

The rendering process in WebGL follows the graphics pipeline in OpenGL, shown in Figure 2. To rendering complicated graphics, including color and texture, users can modify the vertexShader and the fragmentShader to achieve it. Rendering the 3D graphics follows the same steps, however, WebGL needs more vertices to define the 3D objects and more complicated matrix calculations.

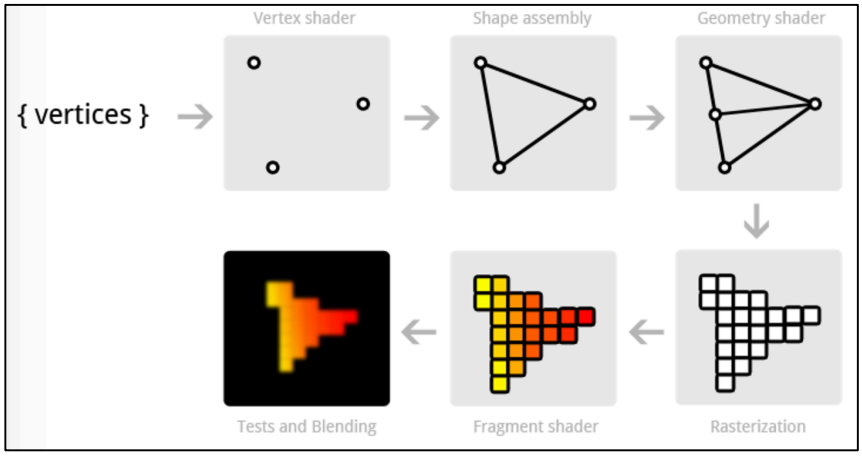

Figure 2. The pipeline of rendering graphics in OpenGL. Figure is taken from reference [5].

4. Advantages of WebGL

WebGL has several advantages over other 3D graphics rendering solutions.

First, WebGL requires no plug-ins. WebGL provides APIs that can directly talk with computer's GPU. Other 3D rendering solutions, such as Silverlight, Unity3D, are dependent on third-party plugins, which is especially limited on mobile devices [4].

Second, WebGL is flexible and scalable. Given the access to the computer's GPU, WebGL is highly scalable as the user's graphics unit is becoming more powerful. Because of the direct access to lower-level hardware, WebGL is very flexible. Currently, there are many frameworks and libraries built upon WebGL, which greatly facilitate the development.

Third, WebGL provides an easy way to integrate into web applications. WebGL renders the graphics through the HTML's <canvas> tag, and it also has the access to all DOM elements. Users can easily plugin a WebGL program into an existing web project.

5. Drawbacks of WebGL

When rendering very complicated graphics, the limitation from the computer's graphics unit is hard to compensate by other means. In this point, WebGL's feature of largely relying on GPU is a double-edged sword. On one hand, it provides scalability and flexibility. On the other hand, if the computer's GPU is poor, it's hard to compensate such limitation.

6. Conclusion

WebGL is a powerful solution to render 3D graphics in web development. It allows developers to integrate real-time interactive 3D graphics in the browser, which greatly facilitate the development of 3D online modeling, online gaming, etc.

Reference:

[1] https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API

[2] http://www.senchalabs.org/philogl/

[3] https://github.com/mrdoob/three.js#readme

[4] https://www.linkedin.com/pulse/why-webgl-awesome-you-should-start-considering-luigi-mannoni

[5] https://open.gl/drawing