2019

Medical Data Visual Synchronization and Information interaction Using Internet-based Graphics Rendering and Message-oriented Streaming

Qi Zhang
Illinois State University, qzhan10@ilstu.edu

Informatics in Medicine Unlocked 17 (2019) 100253

Contents lists available at ScienceDirect

Informatics in Medicine Unlocked

journal homepage: http://www.elsevier.com/locate/imu

Medical data visual synchronization and information interaction using Internet-based graphics rendering and message-oriented streaming

Qi Zhang

School of Information Technology, Illinois State University, 100 North University Street, Normal, IL, 61761, United States

ARTICLE INFO

Keywords:
Medical information
Graphics rendering
Visual synchronization
Collaborative diagnosis
Bidirectional internet connection
Message streaming

ABSTRACT

The rapid technology advances in medical devices make possible the generation of vast amounts of data, which contain massive quantities of diagnostic information. Interactively accessing and sharing the acquired data on the Internet is critically important in telemedicine. However, due to the lack of efficient algorithms and high computational cost, collaborative medical data exploration on the Internet is still a challenging task in clinical settings. Therefore, we develop a web-based medical image rendering and visual synchronization software platform, in which novel algorithms are created for parallel data computing and image feature enhancement, where Node.js and Socket.IO libraries are utilized to establish bidirectional connections between server and clients in real time. In addition, we design a new methodology to stream medical information among all connected users, whose identities and input messages can be automatically stored in database and extracted in web browsers. The presented software framework will provide multiple medical practitioners with immediate visual feedback and interactive information in applications such as collaborative therapy planning, distributed treatment, and remote clinical health care.

1. Introduction

Medical data and information visualization has wide clinical applications [1], and software platforms that reside on web engines present advantages over local applications [2]. When compared with utilizing stand-alone software platforms, the use of Internet-based solutions can lead to better performance in diagnostic information collaborative analysis and treatment decision-making [3]. The advancement of Internet technologies has made it possible to establish a "common ground" for medical practitioners to collaboratively explore medical data and share diagnostic information across the globe [4,5]. When a software application is developed for the web, it can be directly accessed and executed by any users around the world with Internet connection and web browser [6]. Internet-based medical information systems at all levels can expand doctors' capacities and capabilities to meet the growing demand for delivering clinical services across wide geographic areas [7,8], and can also allow medical practitioners to interact each other at remote locations and effectively share medical resources [9].

Some interesting network-based medical information platforms have been developed, such as streaming medical data [10] or images [11] on Internet, transporting data between web server and clients for medical image sharing and analysis [12]. A medical data viewer is reported in Ref. [13] to process and visualize body surface potential maps in web browsers, where Description Extensible Markup Language (XML) is used to process data storage. For the purpose of providing web-based hereditary disease information to medical personals, the authors in Ref. [14] present an Android mobile platform, where Java and XML are integrated into the web framework to handle data rendering. To display and analyze 3D microscopy data on Internet, a web-based software package is developed in Ref. [15], where a Java applet and an interface that interacts with the applet inside HTML browsers are utilized for rendering medical data at a desired contour level. A similar Java applet technology is used by Salavert-Torres et al. [16] to interactively display 2D slice images extracted from 3D microscopy data in web browsers. The following development of this technology used WebGL [17] and HTML5 to build a platform for rendering medical images in web browsers, where VTK [18] is employed for slicing volumetric medical data [19]. All the outlined software platforms are constructed on a client-server structure, which is based on a request-response scheme to access web page and medical data information [20].

Furthermore, web technologies, third-party libraries and software frameworks have been developed for applications of interactive browser-based custom visualization [21] or presenting 3D graphics on the web [22]. Kitware releases a JavaScript library, ParaViewWeb [23],

E-mail address: qzhan10@ilstu.edu.

https://doi.org/10.1016/j.imu.2019.100253
Received 6 August 2019; Received in revised form 20 September 2019; Accepted 25 September 2019
Available online 8 October 2019

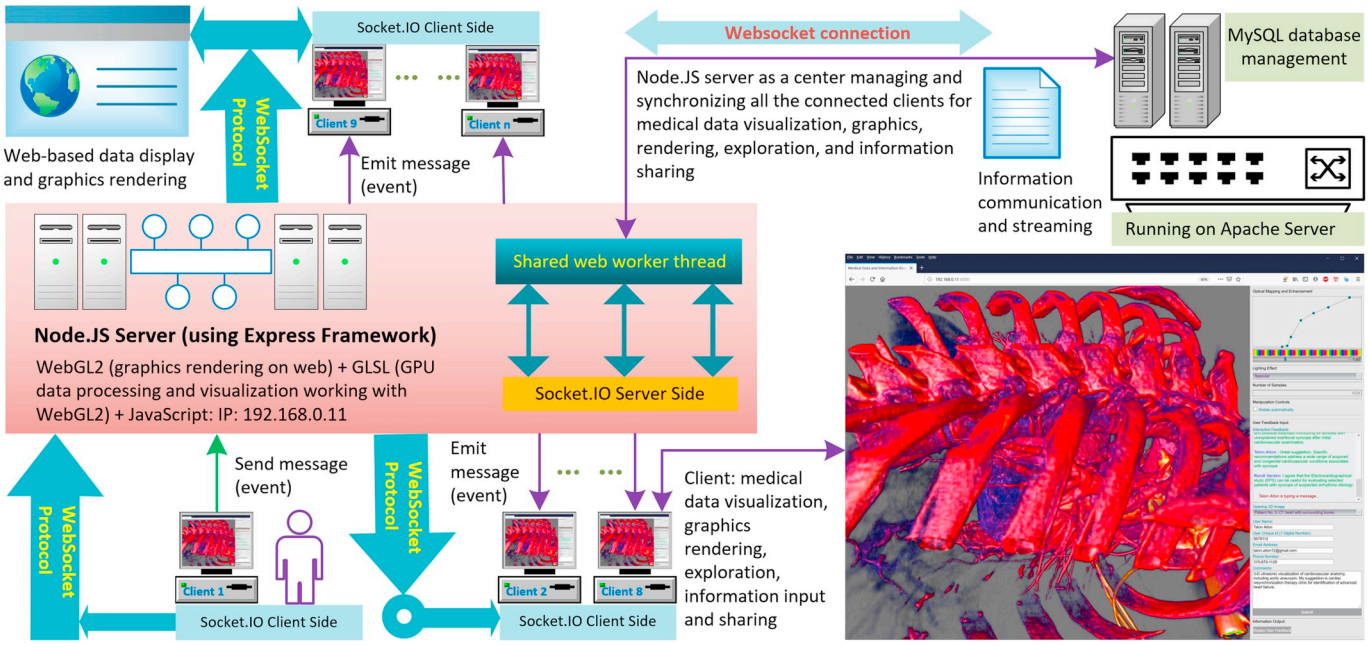

Fig. 1. The flowchart of our system architecture, whose components include Internet-based message streaming, data management, information sharing, 3D medical data real-time rendering and enhancement, as well as bi-directional network connection and medical data visual synchronization. The shared web worker is employed to optimize multi-thread based data loading and medical data exploration.

which is used as a web tool for scientific data visualization inside web browsers. At the same time, a number of researchers explore medical images on the Internet using graphics rendering in web browsers [24]. For example, the authors in Ref. [25] deploy a single-pass data rendering pipeline on mobile devices, where 2D texture in WebGL is used to simulate 3D texture for volume rendering. The same rendering technique is applied to interactively process 2D/3D medical images [26] and is combined with WebSocket for remote collaborative data exploration [27]. Later, Jiménez et al. further developed the presented algorithm and put forward an advanced platform to interactively display and analyze 3D fractal dimension of Magnetic resonance imaging (MRI) data in web browsers [28]. Additional applications include developing open source libraries and software packages to show neurological data [29], protein structure [30] and molecular data [31,32] on the web. Some authors also designed web-based rendering interfaces for advanced medical applications such as personalizing the management of atherosclerotic patients and assisting the diagnostic process by retrieving similar past cases through the Internet [33], exploring multivariate vascular data by visualizing spatial and temporal relationships among the hemodynamic parameters of interest [34].

Recently, to share medical data and research results among clinical and research institutions, the authors in Ref. [35] present a GIFT-Cloud Server to store anonymized data and use a Representational State Transfer Application Programming Interface (REST API) to integrate with external software on Internet. Instead of building server on cloud, a master-slave mode is introduced in Ref. [36] to allow users to access and display remote medical images using volume and surface rendering, where a message-responding mechanism is designed on server to listen to interactive requests from clients. To facilitate radiological practitioners manipulate and interpret cross-sectional volumetric medical data, Borgbjerg [37] integrates a browser-based Picture Archiving and Communication System (PACS) and a imaging viewer into the master-slave based Internet connection structure, and the authors in Ref. [38] create their master computer using an open-source web server software, i.e. Apache HTTP Server [39], then connect the master server with slave clients using the reported networking technology [36,37], where PHP Programming Language [40] is employed to link the server to a MySQL [41] database management system for data storage and extraction. Later, a similar strategy of using PHP to communicate with MySQL database server is reported in Ref. [42], where the authors use a server-client connection protocol to collect data of electronic patient-reported outcome registry.

2. Current work

To the best of our knowledge, current published software packages are generally created on isolated server and clients, which lack the capabilities of streaming messages in the network and data exploration cannot be synchronized among multiple users [2–9]. In addition, Java applets or other external plug-ins are usually required in client computers to handle data transmission and rendering in web browsers [14–16], and the applet connections relay on HTTP Protocol for data transmission, which lacks the capability of full-duplex communication. In addition, most reported software platforms are based on event request-response pipeline instead of automatic bidirectional connection [10–12,19,20]. Some web applications are centralized in cloud [23,35], while others are created on an Apache HTTP Server [39] to maintain data communication [21,22,36–38], both of which cannot stream data in the network and cannot link the explored data with database automatically [41,42].

Master-slave interaction mode is also commonly presented, which uses server as master to organize network connections [36–38] or employs clients to manage data transformation [27]. However, both of the two schemes cannot synchronize data exploration between server and clients. Furthermore, the cross-browser interface three.js [43] is generally employed to manage WebGL-based data visualization, which lacks the ability of flexibly manipulating images and integrating diagnostic information into medical data exploration [27–29]. In addition, surface rendering is also used to display medical data at a contour level, resulting in loss of inner data information [13,31,32]. Finally, due to the limited graphics support of WebGL, 2D texture is usually exploited to simulate 3D texture for trilinear interpolation in volume rendering process, which delivers inferior image quality and limited data visualization speed [24–26].

To address the above described limitations, we build a web-based software platform [44], in which Apache HTTP Server is applied to

construct server-client connections, MySQL and PHP are exploited for data storage management, and a WebGL2 [45] based algorithm is developed to display 3D medical data in real time. This paper presents our further efforts on technology development of our previous work, whose major contributions include developing new algorithms that use Node.js [46] and Socket.IO [47] to build a server and create real-time bidirectional connections with clients, designing a novel method for information sharing and message streaming, and linking input information with MySQL database automatically. Furthermore, we implement a shared web worker and include it into our software framework to optimize medical data multi-thread handling and web connections. Finally, new data lighting and enhancement algorithms are designed and integrated into our volume rendering pipeline to emphasize medical data visualization and image features of interest.

The paper is organized as follows. First, we describe the full-duplex web communication architecture of our application. Then some concrete implementation details regarding data and information synchronization are illustrated. Afterwards, we present the working pipeline of the shared web worker [48]. In Section 5, algorithms of medical data visualization and image feature enhancement are outlined. The results of the experiments are explained and analyzed in Section 6. Finally, our work is summarized in the conclusions section.

3. System architecture

The developed software system is a web-based framework, which is based on a client/server architecture. One of the main features of the web-based application presented in this paper is the synchronization of feature-enhanced medical data rendering and user input message streaming among server, clients and database. Thus, in this section we describe the system architecture, basic algorithms and the theoretical basis of our software platform.

Fig. 1 outlines the system architecture of the server side programming and the opening of a two-way interactive communication between server and client computer's browser. Our software platform is based on an event-driven scheme that is able to synchronize web-based medical data visualization and message streaming, where all the users can share the same dynamically updated medical data exploration view and information over the Internet.

3.1. Node.js server

In this project, Node.js [46] is employed to build server for synchronizing medical data visualization and information communication, while JavaScript [49] is used to write programs for server-side scripting. As described in the Listing 1, first, Express.js (Express) [50] is required and used to build Node.js server. Express is a minimal and flexible Node.js web application framework that provides a set of features for web and mobile applications, while Node.js is an open-source, cross-platform run-time environment that executes JavaScript code outside a browser. The node package manager (npm) is applied to install the Express and create a new file called package JSON [51] to store information of the developed project. The Node.js server hosts the HTML and JavaScript files for data visualization and exploration, which is set to listen to all message events on port 4000.

The left side of Fig. 1 shows the main graphics rendering program, which is hosted on the Node.js server running on Express framework. We develop algorithms using WebGL2 [45] to visualize medical data in the web browsers, where OpenGL Shading Language (GLSL) [52] is utilized for graphics processing unit (GPU) based parallel data computing, rendering and image feature enhancement, which is illustrated in Section 5. All the connected clients are granted privilege to acquire and launch the web-based program hosted on the Node.js server using algorithms built on WebSocket protocol and Socket.IO library.

3.2. Socket.IO connection

To build bidirectional connections between Node.js server and all of the connected clients, JavaScript library Socket.IO [47] is required and utilized for real-time event-based communication: one component is running on Node.js server and the other is based on JavaScript client library that is executing on client web browsers. The Listing 1 shows the server side configuration, where *io* is employed to build socket connections using event callback function *io.sockets.on*.

In our software platform, all clients are connected to the server using algorithms that are based on WebSocket protocol and Socket.IO library [53], which can provide a mechanism for fast, secure, two-way message communication and data exchange between client and server over Internet without need of pulling or making request/response for each message. WebSocket protocol uses an HTTP like handshake to offer a persistent connection between server and client while keeping the connection open. The working pipeline of the described Socket.IO based two-way connection algorithm is demonstrated on the left side of Fig. 1.

```
var express = require('express', 4.17);
var app = express();
var server = app.listen(4000, listen);
var socket = require('socket.io', 2.2)
var io = socket(server);

const mysql = require('mysql', 2.17);
const parser = require('body-parser', 1.19);
app.use(express.static('public'));
io.sockets.on('connection', newConnection);

function newConnection(socket) {
    socket.on('mousedown_ws', mouseDownMsg);
    function mouseDownMsg(data) {
        socket.broadcast.emit('mousedown_ws',
        data);
    }
    socket.on('mousemove_ws', mouseMoveMsg);
    function mouseMoveMsg(data) {
        socket.broadcast.emit('mousemove_ws',
        data);
    }
    socket.on('submit', btClickMsg);
    function btClickMsg(data) {
        socket.broadcast.emit('submit', data);
    }
    ...
}
app.post('/user_create', (req, res) => {
    // Stream information to database
    ...
}
app.get("/users", (req, res) => {
    // Extract information from database
    ...
}
```

Listing 1: Algorithm for creating Node.js server using Express.js, communicating with MySQL database, linking to public directory for HTML and JavaScript code integration, and building connections between Node.js server and clients.

4. Data and information synchronization

In this section some implementation details regarding the development and optimization of the multi-user collaborative platform are outlined. In addition, shared web worker is presented for improving data transformation and web loading. As illustrated in Fig. 1, we develop algorithms to run Socket.IO on both server and clients at the same time. All of the data exploration and visualization have been synchronized on client computers that are connected with the Node.js server.

4.1. Server side programming

As described in Listing 1, on the server, we develop code in lines 1–3 to listen to all new client connections. When a client connects to the server and emits a message event with name *message_name*, the server will send the received message to all connected clients using code in lines 13–17, in which callback function *functionMsg* will broadcast the message using Socket.IO function in lines 15–16. The functions of generated events are mainly for medical data rendering and exploration, such as updating viewport and mouse, menu selection, optical mapping, data navigation, and rendering parameter adjustment. The server is listening to all events on port 4000 using code in lines 1–3. We access the Socket.IO library using the variable *io* created with inputting an object of the Express API *server* as illustrated in lines 4–5, which will be used to keep track of the server's input and output messages.

Listings 1 outlines two examples of listening mouse down and movement events in the function *newConnection*(*socket*), from which we can see that after having received either of these two events, the server will emit the received one to all of the connected clients in the framework using methods described in lines 13–22. In addition, our program creates a link to the "public" folder using a method call in line 9. Inside the "public" directory, there are JavaScript programs, whose main functions are managing clients, rendering medical data, enhancing image features of interest, exploring displayed medical data, as well as sending and receiving messages to and from the server. Fig. 1 describes the flowchart of the server side architecture and the message communication with connected clients, where Node.js and its web application framework Express are exploited to host the actual graphics rendering and information management programs stored in the "public" folder.

4.2. Client side programming

As illustrated in Listing 2, we develop code on the client side to connect the server. The *socket.io* library is first integrated into our client code through linking to the JavaScript library *socket.io.js* in the index. html file, and then the client side Socket.IO program is connected to the server whose IP address is 192.168.0.11 using code in line 1. The client is listening to events coming from the server on port 4000, and a unique client socket id is created for each of the new connections to the server, so that the server can track the connections over time.

```
1 var socket = io.connect('192.168.0.11:4000');
2 document.addEventListener('mousemove',
3 function (e) {
4     ...
5     sendMouseMove(e.screenX, e.screenY);});
6
7 function sendMouseMove(xpos, ypos) {
8     var data = {
9        a: xpos,
10       b: ypos
11    };
12    socket.emit('mousemove_ws', data);
13 }
14 socket.on('mousemove_ws',
15    function (data) {...});
16
17 submitButton.addEventListener('click',
18    function (e) {
19       sendButtonClick();});
20
21 function sendButtonClick() {
22    ...
23    var data = {
24       username: username.value,
25        winmessage: message.value
26    };
27    socket.emit('submit', data);
28 }
```

Listing 2: Algorithm for creating client side network connection using Socket.IO and event-based synchronization. The mouse movement event is used as an example.

For the client side programming, first, the event listening functions are added to the client side Socket.IO component, which will monitor and react to all events such as operations of mouse, keyboard, menu, color mapping, data loading, and medical data rendering parameter adjustment, etc. When the event is user input operation, they will be packaged in a message and emitted to the server, including the name of the message and the data inside the message as described in line 12 with message name *mousemove_ws*. If the event comes from the connected server, the client will conduct the operations programmed in the event using the Socket.IO callback function and the transferred data described in lines 14–15, in which the mouse movement event is listed as an example.

Listings 2 also describes an example of client side programming for handling a mouse movement event. Add an event listener to monitor mouse movement, if the received event comes from a local user input, the client sends the triggered event and corresponding mouse position data to the server using code in lines 14–15. On the other hand, if the event is broadcasted by the connected server, the client will accept event using the same code as the event that is received from user input. The client that has received the event will conduct operations according to the callback function, i.e., *function*(*data*) and the transferred data inside the callback function.

4.3. Message streaming

In our software platform, we develop a message streaming scheme for processing information real-time sharing. As demonstrated by the right side of Fig. 1 and Listings 1, Node.js server uses body-parser and

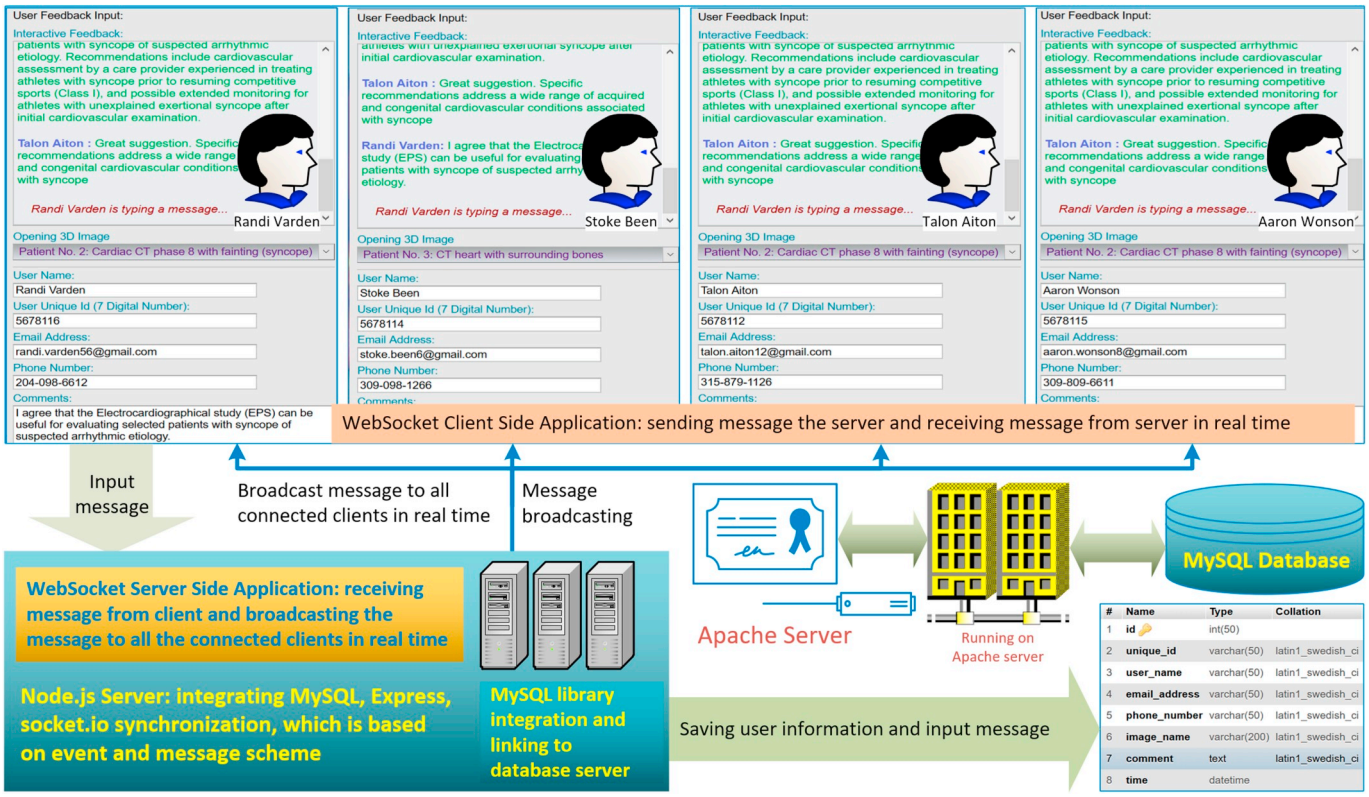

Fig. 2. The working pipeline of our software platform's message streaming and information sharing component. In the graphics user interface, users can input their unique identities and comments regarding medical data exploration that can be shared among all the connected clients. All the information and input messages can be stored in a database table and can be extracted and displayed in web browsers.

mysql libraries to connect MySQL database and post/get data information to/from the database table. On the client side, as illustrated in Fig. 2 and Listings 2, users can input information such as name, unique id, email address and phone number in our platform's graphics user interface. The users can also input diagnostic comments and data exploration messages that are streamed with all the connected clients in real time.

As shown in lines [17–19], a "submit" button is designed to monitor "clicking" event. Inside the callback function in lines [21–28], user information and input messages are sent to the server using Socket.IO emit function in line 28. As outlined in Listings 1, on the server side, a Socket.IO function is designed to listen client events using code in line 23. When the server has received the user input messages, it broadcasts them with the integrated data to all the connected clients in real time using code in line 25.

Our web-based software platform uses the MySQL database server to store and process user messages and information, which are automatically stored in database tables and can be extracted by any connected clients and displayed in web browsers, where Fig. 3 describes the message streaming working pipeline. The MySQL database is running on an Apache HTTP server, which shares the same hardware computer as the Node.js server. Data input and output operations are controlled by Node.js, mysql and body-parser libraries. The Express API is employed to handle data communication between Node.js server and MySQL database using functions in lines [29–36] of Listings 1, while the HTML user interface is the top layer of the message streaming operations. The manager of the system can access and visualize all the stored data and information for the purpose of information mining and data analysis.

4.4. Shared web worker

A WebSocket connection can only persist as long as the page is open, and cannot conduct across loading of new pages from the same domain. Therefore, when a thread spends a long time trying to complete a certain task, such as data loading and information transmission, the application will appear to be frozen. To address this issue, we design a new algorithm that takes advantage of shared web worker to enhance web connection, in which the main thread and the worker communicate via messages and the shared web workers are not reliant on any particular page. Therefore, our JavaScript web connection code can be executed in a separate thread from the page's main thread.

In our application, the shared web worker uses WebSocket protocol to communicate with the server, and once it gets data from the server, it posts them to the main thread for rendering. One shared web worker thread can be used by multiple pages from the same domain, and the web worker is spawned to load in data from the server progressively while the main thread deals with the user interaction, so the data loading will not freeze the applications such as medical data rendering, exploration, and information sharing. The right part of the center rectangle in Fig. 1 illustrates the procedure of using shared web worker to optimize data loading and establish the WebSocket connection with the Apache HTTP Server. The following items describe the working pipeline:

- Initialize the shared web worker and establish connections with Apache server using WebSocket protocol.
- The initialized web worker loads data from server for volume visualization. Every time the worker receives a data file, it parses the file and then sends it to the main thread.
- When the main thread receives the data from the worker, it adds them to a volume and creates a 3D texture.
- Data rendering engine loads the 3D texture for volume rendering and visual exploration.

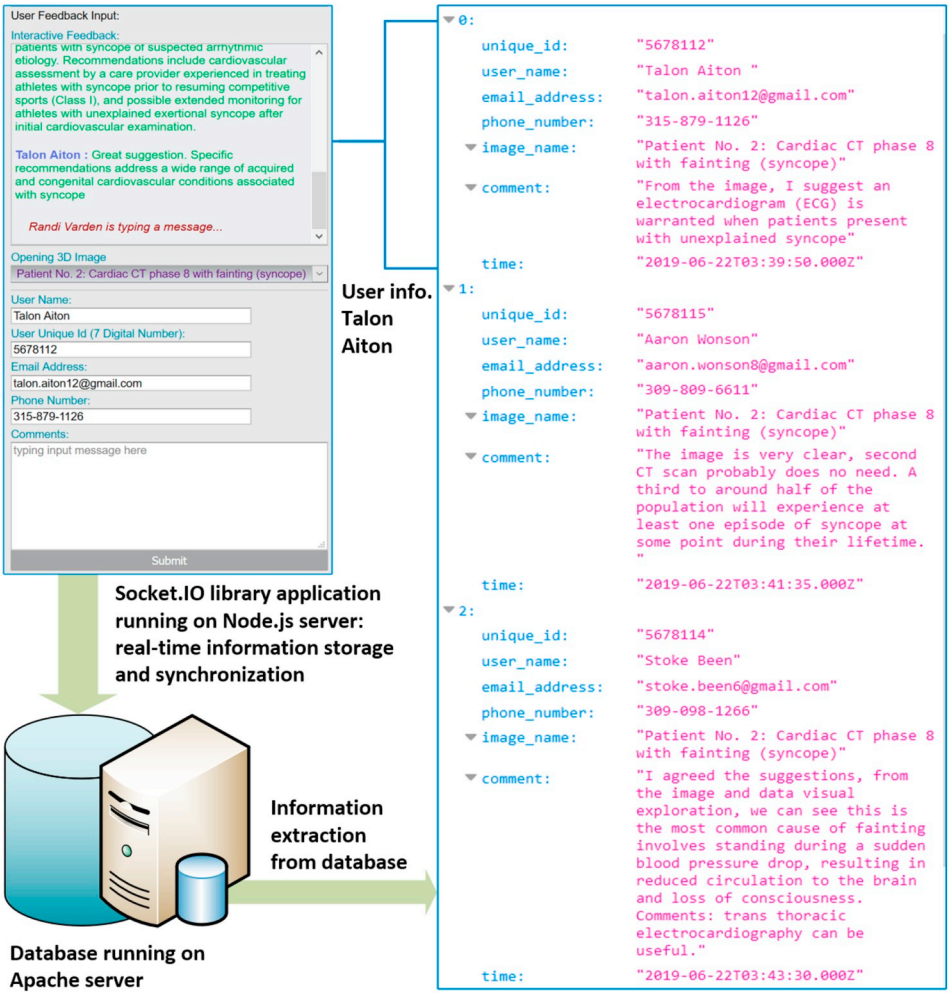

Fig. 3. The workflow of inputting information and messages from graphics user interface, which are automatically streamed into a database and synchronized among all the connected clients. As demonstrated by the right side web-based table, all the connected clients can extract the stored data from the database and display them in web browsers.

5. Data rendering and enhancement

In this section we describe the major technologies used and algorithms developed to visualize and enhance 3D medical data acquired from medical devices such as magnetic resonance imaging (MRI) and computed tomography (CT). In addition to the general structure of the rendering engine, we highlight the aspects and novelty that are useful for increasing algorithm performance and medical data rendering result, such as using graphics hardware to calculate voxel normal, trilinear interpolation, optical value dynamic adjustment, and integrating lighting into the rendered medical image in real time.

5.1. WebGl2 and GPU pipeline

WebGL (Web-based Graphics Library) [17] is a cross-platform web standard developed by the Khronos Group. WebGL programs consist of control code written in JavaScript and special effects code, i.e., shader code written in OpenGL Shading Language (GLSL) [52], that is executed on a computer's Graphics Processing Unit (GPU) to access lower level graphics hardware. WebGL 2.0 (WebGL2) [45] enables web content to use an API based on OpenGL ES 3.0 to perform 3D rendering in HTML *<canvas>* without using plug-ins. Thanks to the HTML5 canvas element, WebGL2 objects are shown in web browsers and their related data are accessible through the Document Object Model (DOM) interface. Therefore, WebGL elements can be mixed with other HTML elements and composited with other parts of the page background in web browsers.

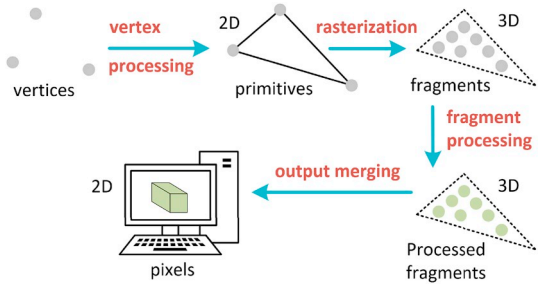

Fig. 4. Web-based graphics rendering pipeline of 3D texture loading and processing in vertex and fragment shaders inside graphics processing unit (GPU) using WebGL2 and OpenGL Shading Language (GLSL).

In the described software platform, we have developed medical data parallel processing and rendering programs using WebGL2's special effects code, i.e., GLSL shaders running on GPU. Vertex shader is first designed and applied to create a 3D data graphics rendering engine. The WebGL2-based medical data rendering pipeline is a process in which 3D images are prepared and output onto the 2D screen: first, take the 3D objects built from primitives using vertices, next, apply vertex processing and calculation in the fragment shaders, finally, render the

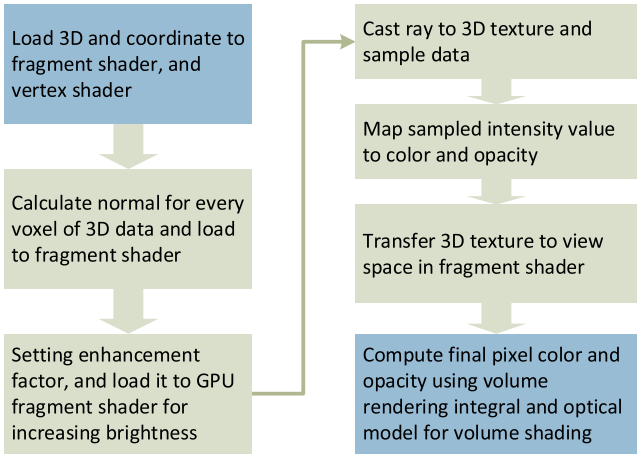

Fig. 5. The outline of the new web-based medical data visualization and enhancement algorithm: data loading, texture translation, texture space transformation, casting ray into texture volume, texture sampling, optical mapping, lighting calculation, and image generation and shading.

calculated results on the 2D screen as image pixels.

As described in Fig. 4, vertex processing is a process of integrating individual vertex information into primitives and setting their coordinates in the 3D space for display. Projection transformation then defines the camera settings and sets up what can be seen by the camera, which includes the field of view, aspect ratio and the near and far clipping planes. Rasterization is a procedure of converting primitives into a set of fragments. Fragment processing focuses on textures and lighting, which is a process of calculating the final colors based on the given parameters. During the output merging stage, all the fragments of the primitives from the 3D space are transformed into a 2D grid of pixels that are then printed out on the display screen.

5.2. Voxel normal calculation

Shading is an important technique to add realistic lighting effect to the volume rendered 3D medical data. In our application, we design an algorithm to integrate a real-time lighting model into the ray casting computing process on graphics hardware unit.

The following Eq. (1) is the calculation of a voxel normal at the location (x, y, z) inside a 3D volume with dimension $M \times N \times K$, where $0 \le x \le M$, $0 \le y \le N$, and $0 \le z \le K$, and we assume that the voxel value at the location (x, y, z) is $I(x, y, z)$.

$$\nabla I(x,y,z) = (\partial I(x,y,z)/\partial x, \partial I(x,y,z)/\partial y, \partial I(x,y,z)/\partial z) \quad (1)$$

where for each element factor of $\partial I(x, y, z)/\partial x$, $\partial I(x, y, z)/\partial y$, and $\partial I(x,y,z)/\partial z$ is calculated by the following Eqs. (2)–(4):

$$\partial I(x,y,z)/\partial x = [I(x+1,y,z) - I(x-1,y,z)]/2 + \lambda \quad (2)$$

$$\partial I(x,y,z)/\partial y = [I(x,y+1,z) - I(x,y-1,z)]/2 + \lambda \quad (3)$$

$$\partial I(x,y,z)/\partial z = [I(x,y,z+1) - I(x,y,z-1)]/2 + \lambda \quad (4)$$

The variable λ is used for boundary adjustment. We can set different values to λ to get a little bit different shading result. Using experimental evaluation, we find that choosing 128 can generate better shading result than using other values, so λ is set to 128 in the above Eqs. (2)–(4).

At the volume boundary, for example $x = M$, the voxel $I(x+1, y, z)$ is invalid, we set it to a large number Ω, i.e., $I(x+1, y, z) = \Omega$. The same $I(x-1, y, z) = \Omega$ when $x = 0$. In the above two cases, we assume $\partial I/\partial x = \lambda$. The same rule is used for normal calculations in the y and z directions. Then load the calculated normal $\vec{n}(x, y, z) = \nabla I(x, y, z)$ into the GPU fragment shader as a global shader variable *uniform sampler3D*.

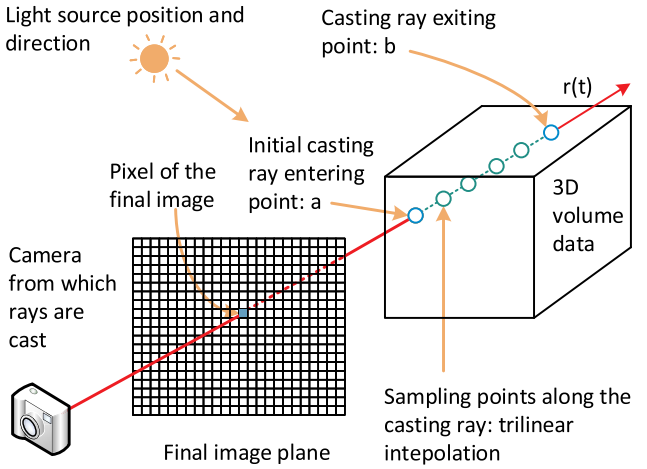

Fig. 6. Pipeline of raycasting calculation on GPU graphics hardware: setting camera and image plane, casting ray into 3D texture volume, adding light source, and conducting trilinear interpolation based texture sampling.

5.3. Data visualization and enhancement

Taking advantage of the new features of WebGL2, we develop a new algorithm for 3D texture based raycasting calculation on GPU with real-time lighting and image feature enhancement. The following items describe our algorithm's procedure of rendering 3D scalar fields into 2D images, which is outlined in Fig. 5.

- Load the volumetric medical data into GPU fragment shader as a 3D texture, at the same time, texture coordinates are loaded to vertex shader for processing, such as adjusting aspect ratio to fix the various z direction sampling distance during 3D texture generation process, then the adjusted 3D texture coordinates are transformed to the fragment shader for texture sampling.
- The normal for every voxel in the 3D texture volume calculated in subsection 5.2 is loaded to fragment shader as a 3D texture for lighting computation using the variable *uniform sampler3D*. In addition, the lighting enhancement factor F_b is also loaded to the fragment shader as an uniform float variable.
- Use the model view matrix to transform 3D texture from texture space to world space and then to view space, which can be processed in the fragment shader for final display that can be seen by viewers.
- As shown in Fig. 6, inside the view space, based on the viewpoint (camera position from which rays are cast) $\vec{p_v}$ and view direction $\vec{v}_d$, from each pixel p on the final image plane, cast a ray $r(t)$ into the data volume in 3D texture space. At each regular sampling point $r(t_i)$ $(i = 0, 1, \cdots, K$, we assume that there are in total K sampling points along the casting ray $r(t))$, acquire the voxel's intensity value $v(r(t_i))$ using trilinear interpolation [54].
- The post color attenuated voxel classification algorithm [55] is used to create a transfer function, which is employed to map each sampled intensity value $p_i = r(t_i)$ $(i = 0, 1, \cdots, K)$ along the casting ray $r(t)$ in the previous step into color $C_i = C(p_i) = C(r(t_i))$ and opacity $\alpha_i = \alpha(p_i) = \alpha(r(t_i))$.
- The ambient light is $\vec{l}_a$, light vector, i.e., we assume that there is far away light with the same direction casting rays, $\vec{l}_v$, and the diffuse light is $\vec{l}_d$. At each sampling point $r(t_i)$, the extracted voxel normal is $\vec{n}_i$ calculated with trilinear-based interpolation on GPU fragment shader. The reflection light is computed with Eq. (5):

$$\vec{l}_r^{\,i} = max(min(\vec{l}_d \cdot \vec{n}_i, 0.0), 1.0) \quad (5)$$

Table 1
Medical data used in the experiment, including data number, image illustration, number of slices, original raw data size in megabyte (MB), and the extracted data size in MB, which can be loaded and displayed in HTML web browsers.

Data number	Image description	Number of slices	Original data size	Extracted data size
1	MR brain with strokes (a part of the brain is interrupted due to a blocked blood vessel)	172	30.45	2.03
2	Cardiac CT phase 8 with fainting (syncope, associated with high rates of morbidity)	84	30.26	1.78
3	MR heart data with surrounding bones with issues of heart pumping oxygen and blood	280	21.26	1.52
4	CT Heart with coronary atherosclerosis, evaluation is necessary for the risk of heart attack	100	24.32	3.43
5	Normal MR brain image for demonstration purpose	176	23.41	1.70
6	MR brain review to check concussions	122	21.42	2.80
7	Skull tooth data with impacted wisdom teeth that is incompletely embedded in the jawbone	100	18.27	0.87
8	MR brain with problems with concentration	109	1.84	0.08
9	Head phantom data cut the brain into half for teaching anatomical demonstration	320	33.41	2.57
10	MR brain has problems of concentration with vessel high cholesterol	160	39.27	2.31
11	Lang data with chronic bronchitis and emphysema with a decline in lung function	42	1.62	0.06
12	Brain data with vessel high cholesterol, which is a building block of steroid hormones	60	23.76	1.32
13	CT data for heart with shortness of breath	49	21.4	4.67
14	Wrist CT data with carpal tunnel syndrome from sudden injuries with scaphoid fractures	160	60.18	3.54

$$C_i = (\vec{l}_a + \vec{l}_{ri} \times \vec{l}_d) \times C_i \tag{6}$$

- The updated C_i using Eq. (6) is used with the flowing ray casting integral Eq. (10) for adding shading to the rendered 3D data without specular lighting.
- To add specular shading, we set specular color C_s and casting ray direction is $\vec{d}$, the specular item at the voxel $r(t_i)$ is computed with Eqs. (7) and (8).

$$\vec{l}_{ri} = -\vec{l}_v + 2 \times \vec{n}_i \times [\vec{n}_i \cdot \vec{l}_v] \tag{7}$$

$$\vec{s}_i = max(\vec{d} \cdot \vec{l}_{ri}, 0.0) \tag{8}$$

$$C_i = (\vec{l}_a + \vec{l}_{ri} \times \vec{l}_d) \times C_i + \alpha_i \times \vec{s}_i \times C_s^{\varphi} \tag{9}$$

where $\vec{l}_{ri}$ is the reflection of $\vec{l}_v$ using normal $\vec{n}_i$. The updated C_i using Eq. (9) is employed with the flowing ray casting integral Eq. (10) for adding shading to the rendered 3D data with specular effect with power $\varphi = 5$.

- Volume rendering integral Eq. (10) is exploited to compute the accumulated color and opacity $I(t_i) = I(a, b_i)$, i.e., $I(t_i) = I(r(t_i))$ is the ray casting integral value at the i^{th} sampling point along each casting ray starting from $p_0 = a$, and stop the ray casting calculation when the calculated opacity is greater than the preset threshold λ ($\lambda = 0.95$ is used in our algorithm) or the ray passed the volume, i.e., $r(t_n) = b$.

$$\begin{aligned} I(a, b_i) &= F_b \int_a^{b_i} C(s)\tau(s)\rho(s)ds \\ &= F_b \left[\sum_{i=1}^{n} C_i \prod_{j=i=1}^{n} (1 - a_j) \right] \end{aligned} \tag{10}$$

where α_j is opacity of the j^{th} segment, and the samples are evaluated from the sampling point $r(t_i)$ to the eye on the casting ray $r(t)$. Loaded normals at each voxel's position is used with light source position $\vec{l}_p$ and direction $\vec{l}_d$ to add shading effect to the final rendered image using Phong optical model [56]. F_b is lighting enhancement factor and can be interactively adjusted through graphics user interface to change the brightness of the rendered 3D medical image.

- When the viewpoint and view direction changes, the casting ray direction and the texture transformation matrix will be updated automatically, and the algorithm will resample the casting ray and recompute the volume rendering integral Eq. (10) to get the updated image.

6. Results and evaluation

In this section, we describe the experimental results using the medical data sets listed in Table 1. We illustrate the user interface (UI) and major functions of our software platform. We also show the medical data visualization results using our new web-based volume rendering, specular lighting and feature enhancement algorithms. Finally, we evaluate the performance of the developed web-based software platform.

6.1. User interface and functionality

The user interface is the communication point between the user and the software platform. The bottom right of Fig. 1 shows a snapshot of the major interface of our web-based platform, which includes a major display area showing the volume rendered 3D medical data and a control panel for medical image exploration and sharing information. The main function of our graphics user interface is collaborative medical data rendering, information visual synchronization, and message streaming. It allows all users to share the same medical data rendering view and to efficiently communicate during the data exploration process.

Fig. 7 shows the console panel for adjusting color, opacity, shading and image enhancement. Our previous work [44] includes only color and opacity adjustment. Using our new volume rendering algorithm, we can effectively enhance the brightness and feature of interest of the visualized image in real time, and we can also add specular lighting to the displayed 3D medical data. In the data rendering control panel, data exploration can be synchronized among all the connected clients in real time, including data set loading and update, zooming in and out, lighting selection, feature enhancement, rotating and panning.

Fig. 3 illustrates the message streaming interface, where the user can first input name, unique id, email address and phone number, then inputting comments on the displayed 3D medical data, and the message is streamed and shared with all the connected clients. As shown in the

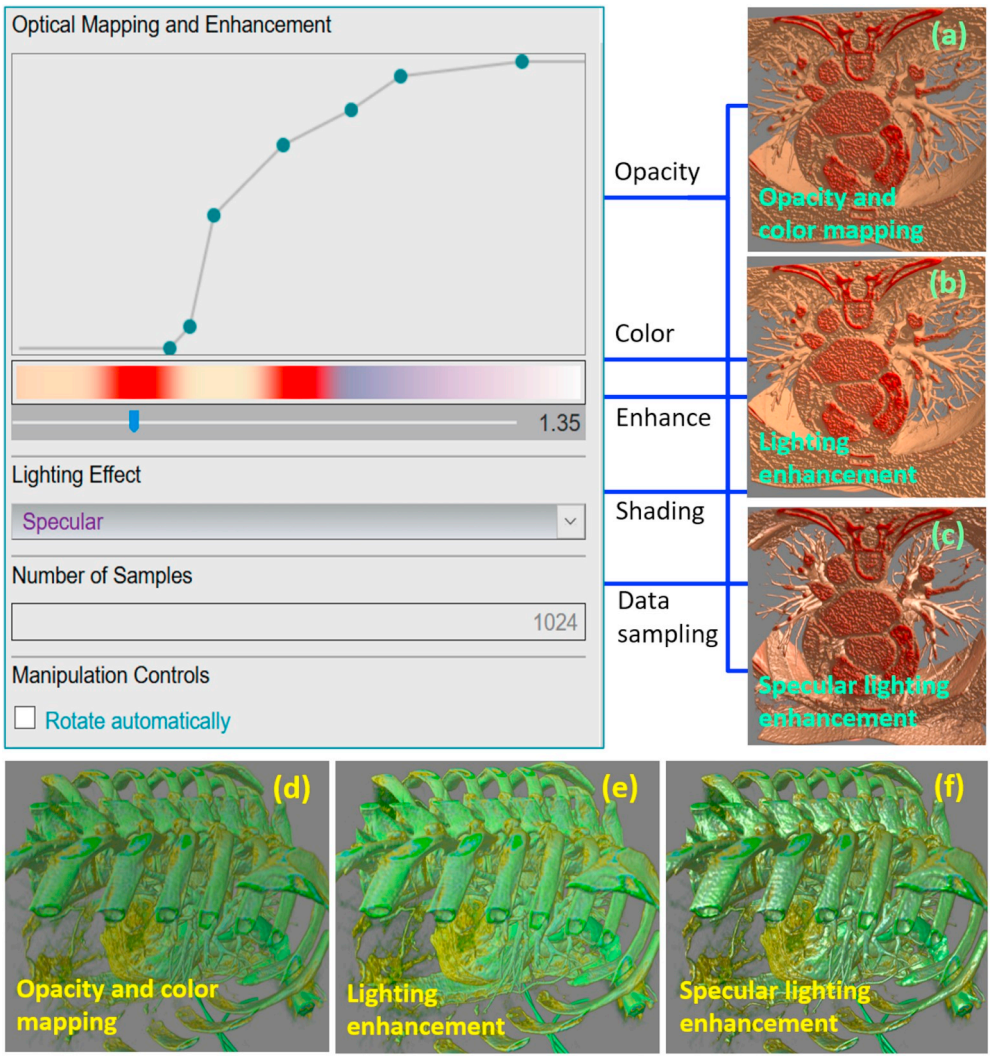

Fig. 7. Demonstration of medical data visual synchronization, including data rendering, opacity adjustment, feature enhancement, and specular lighting. (a)–(c) and (d)–(f) are generated using data number 12 and 3 in Table 1 respectively.

top of Fig. 2, there are four clients that are using the message streaming interface, all the clients share the same dynamically updated information panel and know the user who is typing comments. All the user information and input messages are automatically stored in MySQL database, and can be extracted and displayed in the web page as illustrated in the right table of Fig. 3. In the user interface, all the connected users share the same view, data navigation and message display panel in real time using our new information and data visual synchronization methodologies.

6.2. Data visualization

In this subsection, we will demonstrate medical data visualization results using our new data volume rendering and enhancement algorithm. Fig. 7 (a) and (d) show the original volume rendered images without lighting and enhancement, (b) and (e) are images with brightness enhancement, while (c) and (f) demonstrate images rendered with both specular lighting and enhancement. Data sets listed in Table 1 are used to generate the described images: (a)-(c) are rendered using data number 12, while (d)-(f) are generated with data number 3.

More rendering examples are shown in Fig. 8, where the top row are images generated with our basic visualization algorithm. Eqs. (5) and (6) are used to calculate ambient lighting value in the raycasting computing process, which work with the adjustment factor F_b in the volume rendering function Eq. (10) to integrate lighting enhancement to images (b) and (e), and the second row of Fig. 8. To add specular shading to the rendered images, Eqs. (7)–(9) are used to calculate the specular shading factors and add shading to the final visualized images in (c) and (f) and the bottom row. Fig. 9 shows the enlarged part of the images in Fig. 8, from which we can see the details of the rendered images.

When compared with the medical images generated with our visualization algorithm running on a stand-alone computer with graphics hardware [57], the 3D medical images rendered with our new web-based raycasting algorithm can deliver the same or even better image quality and additional dynamical brightness and enhancing features. As demonstrated in Figs. 7, 8 and 8, our algorithms can generate high-quality volumetric images with realistic shading effects and various colors, which means that the developed raycasting algorithm with the integrated techniques such as lighting, image feature increase, trilinear interpolation and post color attention classification can effectively eliminate image noise and improve the image rendering result.

The visual demonstration in our experiment shows that the new data rendering algorithm can effectively deliver high-quality feature-enhanced medical images. The developed algorithm can show the required details and keep all the data information during the whole image exploration procedure. The imaging enhancement implemented in our web-based platform can increase the reality of the images displayed in web browsers, which is useful in the applications such as

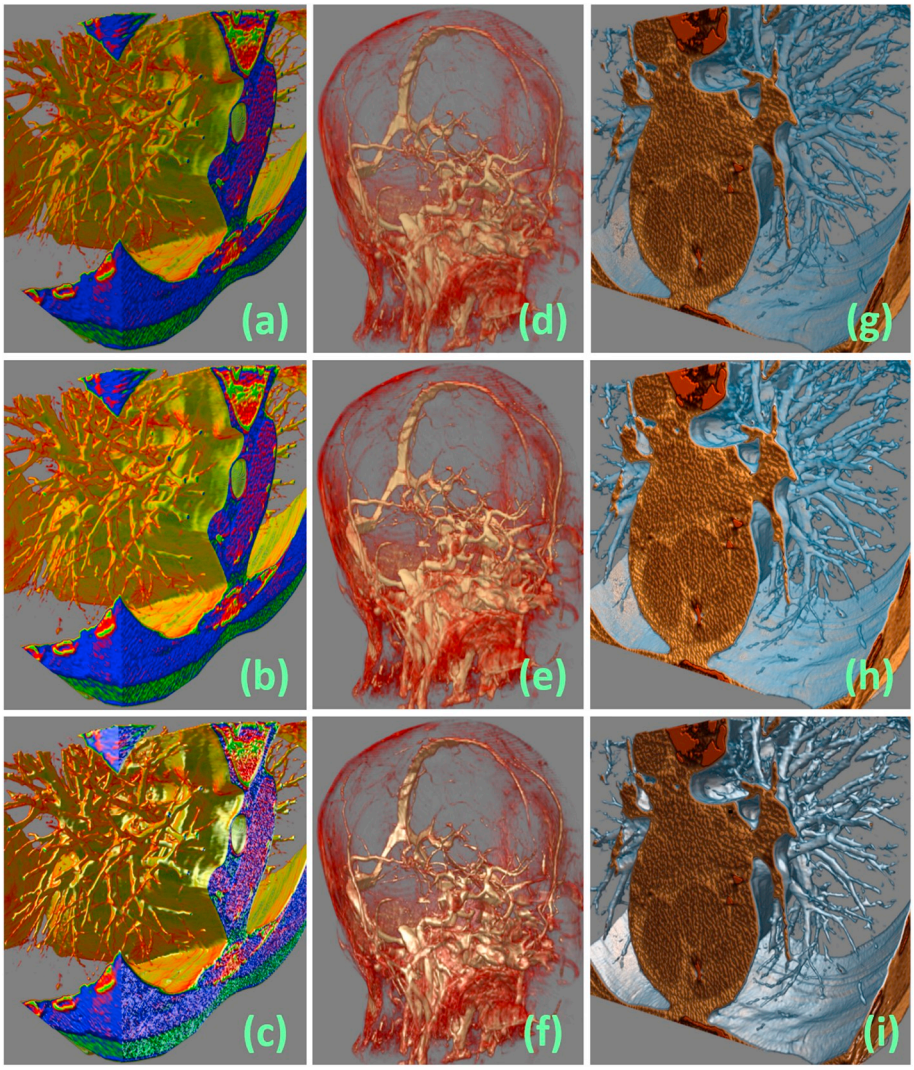

Fig. 8. The rendered 3D medical images using our new WebGL2-based raycasting, image enhancement and specular lighting algorithms. The top row images are rendered without lighting or enhancement, the second row are images visualized with ambient lighting and enhancement, while the images in the bottom row are generated with feature enhancement and specular lighting. (a)–(c) are visualized with data number 4 in Table 1, while (d)–(f) and (g)–(i) are generated with data number 10 and 13 respectively.

virtual reality based imaging guidance and medical teaching demos on the Internet.

6.3. Performance evaluation

Three hardware systems listed in Table 2 are used to evaluate the performance of our web-based software platform. Table 1 describes the data set used in the experiment. Node.js and Apache servers are currently running on a hardware system 1, and we run the clients on all of these three hardware systems. The servers and clients are connected with Internet and WebSocket protocol.

Five persons with basic medical imaging training and networking knowledge evaluate our software platform using data listed in Table 1. All of the participants give positive or very positive feedback on the system's performance regarding the rendering quality and enhanced features. They report that the rendering is smooth and they cannot detect synchronization delay in the process of medical data exploration and information sharing. The evaluators can communicate using our message panel as illustrated in Fig. 3. From the collected feedback, we can confirm that all the messages are streamed in real time and can be extracted from database, and the same medical data view and dynamic message information are shared by all participants in various locations with Internet connection as described in Fig. 2.

As demonstrated in Table 3 and Fig. 10, the first 5 data sets listed in Table 1 are used in the rendering performance experiment. When using Firefox version 68 web browser and system 1, we can detect ~140 frame per second (fps) rendering speed with standard deviation (SD) 5.1 on a high-frequency monitor. The platform's rendering speed on the other hardware systems can achieve ~115±7.8 fps and ~76±7.5 fps respectively. When using Chrome version 75 web browser to visualize the same data sets on the same hardware systems, the display speed is around 5% lower than that of using Firefox. The calculated SD of the rendering speed on Firefox is around 10% smaller than the corresponding rendering speed's SD on Chrome, which means that when rendering 3D data on the Firefox browser, the system's performance is more uniform than that of Chrome.

The top bar image in Fig. 10 visually shows our system's performance on rendering and enhancing the above described medical data sets using the hardware configurations and web browsers listed in Table 3. When using Firefox or Chrome to show the above illustrated five data sets, the average speed is faster than 68 fps with SD less than 10, which means the performance of both browsers are uniform and our algorithm can achieve real time volume rendering for normal size medical data.

We have tested the bidirectional connection time between Node.js server and clients using an Internet with ~25 Mbps download and ~3 Mbps upload speed, including initial connection for server response and shader connection for loading shaders to visualize medical data in web browsers. Table 3 shows the time consumed when using the listed three hardware configurations and two web browsers. Thanks to the newest

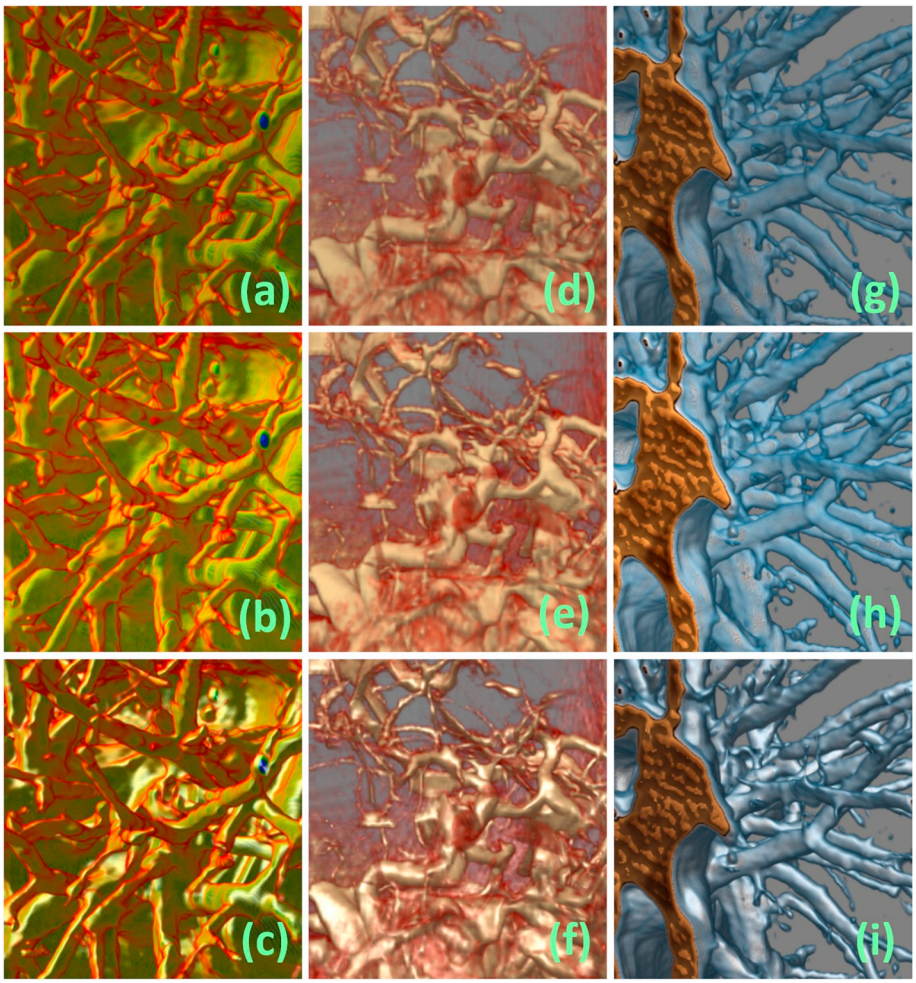

Fig. 9. Zoomed in part of the displayed 3D medical images in Fig. 8. The corresponding images in both figures share the same image label (a)–(i).

Table 2
Hardware configurations used in the experiment: system number, central processing unit (CPU), memory size in Megabytes (MB) and graphics processing unit (GPU). Three systems, i.e., sys. i (i = 1, 2, 3), are employed to evaluate the platform's performance.

Sys.	CPU	Memory	GPU
1	Intel i9-9900KF	64	GTX 1080
2	Intel i7-3770K	32	GTX 680
3	Intel i7-8750H	32	GTX 1070Q

hardware technologies, the connection time with system 1 is shortest in the three systems, which is around 30% faster than the other two hardware configurations. When using Firefox, the initial connection time is 0.21 ms (ms) - 0.42 ms, while the corresponding time needed is 0.35 ms–0.53 ms when using the Chrome web browser. The shader connection (both vertex and fragment shaders) is around 5 times the corresponding initial connection time. The SD for all these connections is less than 0.62, which means that the connection speed is relative uniform and the users can get smooth system performance. The bottom bars in Fig. 10 visually show the connection speed of our software platform including the average connection time needed with corresponding SD, from which we can see that the connection speed is millisecond level, so the final users cannot detect any delay when using our system for Internet based medical data exploration and visual synchronization.

In our web-based software platform, the extension in shared web workers allows data upload from web workers, which frees the main thread and we can use it to handle data rendering and user interactions. Thanks to the efficient use of web workers, data loading in our platform can be conducted smoothly without disturbing other operations on client computers. We also notice that using different hardware systems as Node.js server does not affect the performance of duplex information communication and data rendering speed on client computers. However, the web workers can only solve the problem partly, given their memory overhead and communication model, the use of many workers in a single application may offset the performance gain.

Our experiment demonstrates that all the client computers and the Node.js server can share the same volume rendering and data navigation view in the image display window and the related messages can be streamed in real time using our system's information panel, making the developed web-based software platform useful in collaborative diagnosis, virtual reality based medical training and treatment planning through Internet.

7. Conclusion

In this paper, we present a web-based software platform to visually synchronize medical data exploration on Internet. Taking advantage of Node.js and Socket.IO, we develop algorithms to bidirectionally connect server and clients, where parameter settings for volumetric medical image rendering and manipulation can be interactively updated and shared among all connected clients. We also design an information sharing pipeline for streaming messages among all the connected clients and linking their input to MySQL database directly, where authorized

Table 3
Performance evaluation of data rendering and server-client connection using five medical datasets in Table 1, three hardware configurations listed in Table 2 and Firefox (F) and Chrome (C) web browsers (Br.). The rendering speed is frames per second (fps), while the time cost in both the initial and shader connections is millisecond (ms). The listed numbers are the average of ten tests and standard deviation (SD) is also calculated.

Sys.	Br.	Data Num.	Rendering Speed±SD	Initial Conn.± SD	Shader Conn.± SD
1	F	1	142±5.1	0.21±0.02	1.35±0.25
		2	132±4.8		
		3	145±4.5		
		4	142±4.2		
		5	131±3.8		
	C	1	136±5.8	0.35±0.04	1.56±0.38
		2	116±5.2		
		3	143±5.6		
		4	140±4.7		
		5	128±4.5		
2	F	1	115±7.8	0.32±0.06	1.62±0.38
		2	102±6.9		
		3	118±6.6		
		4	112±7.3		
		5	108±7.5		
	C	1	113±8.4	0.39±0.07	1.85±0.41
		2	98 ±7.2		
		3	115±7.5		
		4	107±8.1		
		5	100±8.8		
3	F	1	76±7.5	0.42±0.08	1.95±0.46
		2	74±6.6		
		3	82±5.4		
		4	78±7.3		
		5	73±5.9		
	C	1	74±8.5	0.53±0.09	2.12±0.62
		2	72±7.3		
		3	78±6.4		
		4	73±8.1		
		5	68±6.2		

users are able to store and retrieve medical data and their associated information.

In the developed system, shared web workers are implemented and integrated into our data synchronization algorithm to optimize multi-thread network connections and web page interactions, and novel graphics lighting and specular shading algorithms are designed to enhance features of interest of the 3D medical images rendered in web browsers. Our software platform can display normal size medical data in real time and deliver high-quality tissue structure enhanced images, which allows clinical users and medical researchers at arbitrary locations to concurrently visualize and analyze the same medical data view in real time. The presented algorithms and web-based software platform will benefit medical applications such as distributed diagnosis, medical collaboration and training on Internet, telemedicine, and remote treatment.

In the future work, we will consider designing some advanced interpolation techniques to handle the missing voxels outside the volume as described in subsection 5.2, and then use the interpolation result to calculate the boundary voxel derivatives. We also plan to improve user interactions, allowing users to dynamically change φ in Eq. (9) from user interface, so the shading effect of the specular volume lighting can be updated accordingly in real time. Furthermore, we will integrate new web technologies such as WebCL (Web Computing Language) [58] into our current WebGL based data rendering pipeline to take advantage of

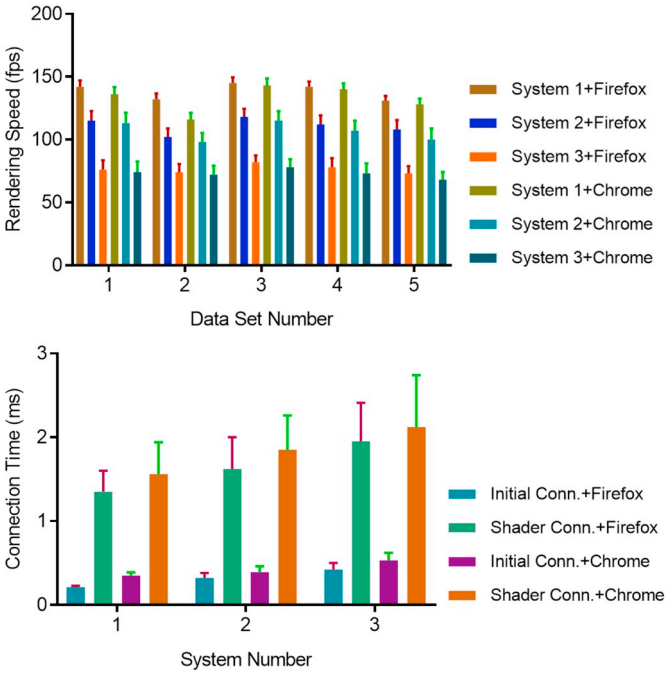

Fig. 10. Demonstration of volume rendering and system connection speed, i.e., time used to launch the required connection, using the listed three systems and five data sets in Table 3. Top: data visualization speed in frames per second. Bottom: connection speed between clients and Node.js server in millisecond (ms).

the combined computing features of both central processing unit (CPU) and graphics processing unit (GPU) for high-performance parallel medical data computing and visualization in web browsers. Finally, we plan to apply our software to a large cloud framework in hospitals or medical institutes to validate its real-world performance in Internet-based clinical diagnosis, treatment planning, and collaborative therapy.

Ethical statement

The author has no ethical conflicts, financial or personal or otherwise, related to the presented research.

Declaration of competing interest

The author declares no conflicts of interest.

Acknowledgment

The author would like to thank the Faculty Startup Grant of the School of Information Technology at Illinois State University. The author also thanks the support of the College of Arts and Sciences (CAST) Publication Incentive Program Award and the CAST University Research Grant (URG). As an adjunct faculty, the author would appreciate the support of the Department of Medical Biophysics at the Western University for providing digital library access and research collaborations. Finally, the author would like to extend his appreciation to the reviewer's suggestions and comments as well as Dr. Ciaccio's recommendation and proofreading this paper.

References

[1] Carroll LN, Au AP, Detwiler LT, chieh Fu T, Painter IS, Abernethy NF. Visualization and analytics tools for infectious disease epidemiology: a systematic review. J Biomed Inform 2014;51:287–98. https://doi.org/10.1016/j.jbi.2014.04.006.

[2] Glover T. Using web applications for data visualisation. In: Proceedings of the european conference on cognitive ergonomics, ECCE '16. New York, NY, USA: ACM; 2016. 31:1–31:2.

[3] Kooper R, Shirk A, Lee S-C, Lin A, Folberg R, Bajcsy P. 3d medical volume reconstruction using web services. Comput Biol Med 2008;38(4):490–500. https://doi.org/10.1016/j.compbiomed.2008.01.015.

[4] Min Q, Wang Z, Liu N. An evaluation of html5 and webgl for medical imaging applications. Journal of Healthcare Engineering 2018;2018:11. https://doi.org/10.1155/2018/1592821.

[5] Qiao L, Li Y, Chen X, Yang S, Gao P, Liu H, Feng Z, Nian Y, Qiu M. Medical high-resolution image sharing and electronic whiteboard system: a pure-web-based system for accessing and discussing lossless original images in telemedicine. Comput Methods Progr Biomed 2015;121(2):77–91. https://doi.org/10.1016/j.cmpb.2015.05.010.

[6] Ku W-Y, Nfor ON, Liu W-H, Tantoh DM, Hsu S-Y, Wang L, Chou T-Y, Liaw Y-P. Online community collaborative map: a geospatial and data visualization tool for cancer data. Medicine 2019;98:89. https://10.1097/MD.0000000000015521.

[7] Elhoseny M, Bian G-B, Lakshmanaprabu S, Shankar K, Singh AK, Wu W. Effective features to classify ovarian cancer data in internet of medical things. Comput Network 2019;159:147–56. https://doi.org/10.1016/j.comnet.2019.04.016.

[8] Woo J, Lee MJ, Ku Y, Chen H. Modeling the dynamics of medical information through web forums in medical industry. Technol Forecast Soc Chang 2015;97:77–90. https://doi.org/10.1016/j.techfore.2013.12.006.

[9] Fonzo GA, Fine NB, Wright RN, Achituv M, Zaiko YV, Merin O, Shalev AY, Etkin A. Internet-delivered computerized cognitive & affective remediation training for the treatment of acute and chronic posttraumatic stress disorder: two randomized clinical trials. J Psychiatr Res 2019;115:82–9. https://doi.org/10.1016/j.jpsychires.2019.05.007.

[10] Al-Shammari A, Zhou R, Naseriparsaa M, Liu C. An effective density-based clustering and dynamic maintenance framework for evolving medical data streams. Int J Med Inform 2019;126:176–86.

[11] Shen H, Ma D, Zhao Y, Sun H, Sun S, Ye R, Huang L, Lang B, Sun Y. Miaps: a web-based system for remotely accessing and presenting medical images. Comput Methods Progr Biomed 2014;113(1):266–83. https://doi.org/10.1016/j.cmpb.2013.09.008.

[12] Koulouzis S, Zudilova-Seinstra E, Belloum A. Data transport between visualization web services for medical image analysis. Procedia Computer Science 2010;1(1):1727–36. https://doi.org/10.1016/j.procs.2010.04.194. iCCS 2010. URL.

[13] Bond RR, Finlay DD, Nugent CD, Moore G. A web-based tool for processing and visualizing body surface potential maps. J Electrocardiol 2010;43(6):560–5. https://doi.org/10.1016/j.jelectrocard.2010.05.010.

[14] Oluwagbemi O, Oluwagbemi F, Ughamadu C. Android mobile informatics application for some hereditary diseases and disorders (amahd): a complementary framework for medical practitioners and patients. Informatics in Medicine Unlocked 2016;2:38–69. https://doi.org/10.1016/j.imu.2016.03.001.

[15] Lagerstedt I, Moore WJ, Patwardhan A, Sanz-García E, Best C, Swedlow JR, Kleywegt GJ. Web-based visualisation and analysis of 3d electron-microscopy data from emdb and pdb. J Struct Biol 2013;184(2):173–81. https://doi.org/10.1016/j.jsb.2013.09.021.

[16] Salavert-Torres J, Iudin A, Lagerstedt I, Sanz-García E, Kleywegt GJ, Patwardhan A. Web-based volume slicer for 3d electron-microscopy data from emdb. J Struct Biol 2016;194(2):164–70. https://doi.org/10.1016/j.jsb.2016.02.012.

[17] The Khronos Group, WebGL. OpenGL ES for the web. URL, https://www.khronos.org/webgl/; 2019.

[18] Schroeder W, Martin K, Lorensen B. The visualization toolkit–an object-oriented approach to 3D graphics. fourth ed. Kitware, Inc.; 2006.

[19] Tiwari M, Kumar P, Agrawal A. Web-based volume visualization of 3d medical data using slice streaming method. In: Proceedings of the sixth international conference on computer and communication technology 2015, ICCCT '15. New York, NY, USA: ACM; 2015. p. 194–8.

[20] Min Q, Liu N, Chen Y. A web-based medical image viewer for 2d and 3d visualization. In: Proceedings of the 2018 2Nd international conference on management engineering, software engineering and service Sciences, ICMSS 2018. New York, NY, USA: ACM; 2018. p. 261–4.

[21] Shahzad F, Sheltami TR, Shakshuki EM, Shaikh O. A review of latest web tools and libraries for state-of-the-art visualization. Procedia Computer Science 2016;98:100–6. https://doi.org/10.1016/j.procs.2016.09.017. the 7th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN 2016)/The 6th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH-2016)/Affiliated Workshops. URL.

[22] Evans A, Romeo M, Bahrehmand A, Agenjo J, Blat J. 3d graphics on the web: a survey. Comput Graph 2014;41:43–61. https://doi.org/10.1016/j.cag.2014.02.002.

[23] Jomier J, Jourdain S, Ayachit U, Marion C. Remote visualization of large datasets with midas and paraviewweb. In: Proceedings of the 16th international conference on 3D web technology, Web3D '11. New York, NY, USA: ACM; 2011. p. 147–50.

[24] Noguera JM, Jiménez JR. Mobile volume rendering: past, present and future. IEEE Trans Vis Comput Graph 2016;22(2):1164–78. https://doi.org/10.1109/TVCG.2015.2430343.

[25] Mobeen MM, Feng L. High-performance volume rendering on the ubiquitous webgl platform. In: IEEE 14th international conference on high performance computing and communication 2012 IEEE 9th international conference on embedded software and systems, 2012; 2012. p. 381–8. https://10.1109/HPCC.2012.58.

[26] Mahmoudi SE, Akhondi-Asl A, Rahmani R, Faghih-Roohi S, Taimouri V, Sabouri A, Soltanian-Zadeh H. Web-based interactive 2d/3d medical image processing and visualization software. Comput Methods Progr Biomed 2010;98(2):172–82. https://doi.org/10.1016/j.cmpb.2009.11.012.

[27] Marion C, Jomier J. Real-time collaborative scientific webgl visualization with websocket. In: Proceedings of the 17th international conference on 3D web technology, Web3D '12; 2012. p. 47–50.

[28] Jiménez J, López A, Cruz J, Esteban F, Navas J, Villoslada P, de Miras JR. A web platform for the interactive visualization and analysis of the 3d fractal dimension of mri data. J Biomed Inform 2014;51:176–90. https://doi.org/10.1016/j.jbi.2014.05.011.

[29] Sherif T, Kassis N, Rousseau M-t, Adalat R, Evans AC. Brainbrowser: distributed, web-based neurological data visualization. Front Neuroinf 2015;8:89. https://doi.org/10.3389/fninf.2014.00089. https://www.frontiersin.org/article/10.3389/fninf.2014.00089.

[30] Shi M, Gao J, Zhang MQ. Web3DMol: interactive protein structure visualization based on WebGL. Nucleic Acids Res 2017;45(W1):W523–7.

[31] Rego N, Koes D. 3Dmol.js: molecular visualization with WebGL. Bioinformatics 2014;31(8):1322–4.

[32] Marion C, Pouderoux J, Jomier J, Jourdain S, Hanwell M, Ayachit U. A hybrid visualization system for molecular models. In: Proceedings of the 18th international conference on 3D web technology, Web3D '13. New York, NY, USA: ACM; 2013. p. 117–20.

[33] Gastounioti A, Kolias V, Golemati S, Tsiaparas NN, Matsakou A, Stoitsis JS, Kadoglou NP, Gkekas C, Kakisis JD, Liapis CD, Karakitsos P, Sarafis I, Angelidis P, Nikita KS. Carotid – a web-based platform for optimal personalized management of atherosclerotic patients. Comput Methods Progr Biomed 2014;114(2):183–93. https://doi.org/10.1016/j.cmpb.2014.02.006.

[34] Tao J, Huang X, Qiu F, Wang C, Jiang J, Shene C-K, Zhao Y, Yu D. Vesselmap: a web interface to explore multivariate vascular data. Comput Graph 2016;59:79–92. https://doi.org/10.1016/j.cag.2016.05.024.

[35] Doel T, Shakir DI, Pratt R, Aertsen M, Moggridge J, Bellon E, David AL, Deprest J, Vercauteren T, Ourselin S. Gift-cloud: a data sharing and collaboration platform for medical imaging research. Comput Methods Progr Biomed 2017;139:181–90. https://doi.org/10.1016/j.cmpb.2016.11.004.

[36] Qiao L, Chen X, Zhang Y, Zhang J-N, Wu Y, Li Y, Mo X, Chen W, Xie B, Qiu M. An html5-based pure website solution for rapidly viewing and processing large-scale 3d medical volume reconstruction on mobile internet. International Journal of Telemedicine and Applications 2017;2017:1–13. https://doi.org/10.1155/2017/4074137.

[37] Borgbjerg J. Mulrecon: a web-based imaging viewer for visualization of volumetric images. Curr. Probl. Diagn. Radiol. 2019;48(6):531–4. https://doi.org/10.1067/j.cpradiol.2018.09.001.

[38] Huang Q, Huang X, Liu L, Lin Y, Long X, Li X. A case-oriented web-based training system for breast cancer diagnosis. Comput Methods Progr Biomed 2018;156:73–83. https://doi.org/10.1016/j.cmpb.2017.12.028.

[39] Bloom RB. Apache server 2.0: the complete reference. New York, NY, USA: McGraw-Hill, Inc.; 2002.

[40] Atkinson L, Suraski Z. Core PHP programming. third ed. Prentice Hall Professional Technical Reference; 2003. third ed.

[41] Widenius M, Axmark D. Mysql reference manual. first ed. Sebastopol, CA, USA: O'Reilly & Associates, Inc.; 2002.

[42] Lizzio VA, Gulledge CM, Meta F, Franovic S, Makhni EC. Using a web-based data collection platform to implement an effective electronic patient-reported outcome registry. Arthroscopy Techniques 2019;8(6):e535–9. https://doi.org/10.1016/j.eats.2019.01.012.

[43] Dirksen J. Learning Three.js – the JavaScript 3D library for WebGL. second ed. Packt Publishing - ebooks Account; 2015. second ed. https://threejs.org.

[44] Zhang Q. Web-based medical data visualization and information sharing towards application in distributed diagnosis. Informatics in Medicine Unlocked 2019;14:69–81. https://doi.org/10.1016/j.imu.2018.10.010.

[45] The Khronos Group. WebGL 2.0 specification. https://www.khronos.org/registry/webgl/specs/latest/2.0/; 2019.

[46] Teixeira P. Professional Node.Js: building javascript based scalable software. first ed. Birmingham, UK, UK: Wrox Press Ltd.; 2012.

[47] Rauch G. Socket.io 2.0 is here: featuring the fastest and most reliable real-time engine. https://socket.io; 2019.

[48] The Web Hypertext Application Technology Working Group (WHATWG). HTML living standard. https://html.spec.whatwg.org/multipage/workers.html; 2019.

[49] Ready to try javascript?. https://www.javascript.com/. [Accessed 16 July 2019].

[50] Holmes S. Getting MEAN with mongo, express, angular, and node. first ed. Greenwich, CT, USA: Manning Publications Co.; 2015.

[51] Contributing packages to the registry: creating a package.json file. https://docs.npmjs.com/creating-a-package-json-file. [Accessed 16 July 2019].

[52] Wolff D. OpenGL 4 Shading Language cookbook: build high-quality, real-time 3D graphics with OpenGL 4.6, GLSL 4.6 and C++17. third ed. Packt Publishing; 2018.

[53] Fette I, Melnikov A. Relationship to TCP and HTTP. In: RFC 6455 the WebSocket protocol. first ed. Chichester: IETF; 2011.

[54] Hill S. Graphics gems iv. Ch. Tri-linear Interpolation. San Diego, CA, USA: Academic Press Professional, Inc.; 1994. p. 521–5. http://dl.acm.org/citation.cfm?id=180895.180944.

[55] Zhang Q, Eagleson R, Peters TM. Rapid scalar value classification and volume clipping for interactive 3d medical image visualization. Vis Comput 2011;27(1):3–19. https://doi.org/10.1007/s00371-010-0509-z.

[56] Phong BT. Illumination for computer generated pictures, Commun. ACM 1975;18 (6):311–7. https://doi.org/10.1145/360825.360839.
[57] Zhang Q, Eagleson R, Peters TM. Dynamic real-time 4d cardiac mdct image display using gpu-accelerated volume rendering. Comput Med Imag Graph 2009;33(6): 461–76. https://doi.org/10.1016/j.compmedimag.2009.04.002.
[58] The Khronos Group. Webcl overview: heterogeneous parallel computing in html5 web browsers. https://www.khronos.org/webcl/; 2019.