DICOM SIVR: A web architecture and platform for seamless DICOM image and volume rendering

Jozić, Krešimir; Frid, Nikolina; Jović, Alan; Mihajlović, Željka

SoftwareX 18 (2022) 101063

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

DICOM SIVR: A web architecture and platform for seamless DICOM image and volume rendering

Krešimir Jozić [a], Nikolina Frid [b,*], Alan Jović [b], Željka Mihajlović [b]

[a] Industrial Applications, Siemens Energy d.o.o., Heinzelova ul. 70a, 10000 Zagreb, Croatia
[b] University of Zagreb, Faculty of Electrical Engineering and Computing, Unska 3, 10000 Zagreb, Croatia

ARTICLE INFO

Article history:
Received 14 February 2022
Received in revised form 18 March 2022
Accepted 23 March 2022

Keywords:
DICOM
Medical imaging
Web application
Go

ABSTRACT

Quick access to radiological images is important for timely diagnosis and effective patient treatment. In this paper, we present a web-based client–server system for seamless image and volume rendering of DICOM images that provides fast access to the data needed for diagnosis without placing a heavy load on computer resources on the client side. DICOM images are rendered on the server, and the resulting 2D images are sent to physicians who can view and analyze them via web browser. Security of patient medical data is ensured by encryption during storage and transfer. The system's communication model hides the latency of remote rendering to ensure a seamless experience for the user.

Code metadata

Current code version	*v0.3*
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-22-00045
Permanent link to reproducible capsule	https://sivr.info:5000
Legal code license	*MIT License*
Code versioning system used	*git*
Software code languages, tools and services used	*Go, Angular framework with Material components, CockroachDB Core edition database*
Compilation requirements, operating environments and dependencies	*Go, Node.js*
If available, link to developer documentation/manual	https://gitlab.com/kjozic/sivr/-/blob/main/README.txt
Support email for questions	kjozic@gmail.com

1. Motivation and significance

The application of computer visualization in medicine in its numerous modalities (CT, MR, PET, etc.) is essential for diagnosis, education, practicing operative procedures, pre-operative planning, and telemedicine [1]. DICOM (Digital Imaging and Communications in Medicine) is a standard for storage, management, and transmission of medical images developed in 1985, and continuously republished since [2,3]. It is used to store different modalities and provides a good foundation for building PACS systems (Picture Archiving and Communication System) [4,5].

The major issue in this field is that medical images are still often stored and transferred using disposable optical media or flash drives [6,7]. Besides the possibility of media damage and loss of records after prolonged storage, the main drawback of this approach is the time it takes for the image to reach the physician's office when it is dislocated from the scanning facilities. In addition, the large amount of discarded disposable media negatively impacts the environment.

The solution is to enable access to medical images over the Internet. The development of remote imaging was initiated by the needs of small or rural hospitals [8], and gained significant importance during the COVID crisis [9], when physicians' contact with patients, transmission, and handling of optical or USB media became more complicated. The key requirements for such a system are: (1) DICOM image processing must be done on the backend, as it requires more complex hardware and software

* Corresponding author.
E-mail addresses: kresimir.jozic@siemens-energy.com (Krešimir Jozić), nikolina.frid@fer.hr (Nikolina Frid), alan.jovic@fer.hr (Alan Jović), zeljka.mihajlovic@fer.hr (Željka Mihajlović).

https://doi.org/10.1016/j.softx.2022.101063

than typically found in medical offices, and (2) transport of rendered images must be optimized for low-to-medium throughput internet connections affordable for small GP's.

Several client–server platforms for medical image visualization have been developed over the past decade. The authors in [10] present a custom implementation that relies on VRML or X3D as data transfer and storage formats. Although this solution is well suited for modest resources on the client side, these technologies have reached the end of their life. Newer solutions are based on the DICOM standard. In [11], volume rendering is performed using GDCM VTK, and the Python NumPy library. However, using an interpreted language like Python usually results in lower performance, and the authors show only the graphical renderings, but no timings. In [12], DICOM images are transferred to and rendered in a web browser, which requires a high-throughput internet connection, large RAM capacity and hardware capability for volume rendering. The Studierfenster platform [13] offers visualization and image analysis with advanced features like AR/VR, cranial implant design, and facial reconstruction. DICOM images are converted to NRRD format on the client side to avoid transferring patient data to the server. While advanced functionalities are executed on the backend, some image processing is still done client-side. This relies on WebGL and requires GPU capabilities that are not always available on PCs in physicians' offices.

In this paper, we present a client–server web-based system for Seamless Image and Volume Rendering (SIVR) of DICOM images. DICOM images are rendered in the backend, and the resulting 2D images are sent to physicians who can view and analyze them in a web browser. The key advantage of the proposed system is that it provides fast access to the data needed for diagnosis without placing a heavy demand on computing resources on the client side, thus requiring minimal or no investment on the part of the institution, i.e., making optimal use of the available hardware. The system architecture and communication model are designed to support load distribution across multiple servers, allowing easy scaling. Image pre-processing is performed in advance to mask latency so that users do not feel any delay in transmission. This work is a continuation of the work previously published in [14], now extended to include the ability to render the entire volume. Security and overall performance have also been improved by adding compression and encryption of stored data.

In Section 2, we describe system architecture and main functionalities. In Section 3, we present several use cases, while in Section 4, we elaborate on the overall impact of the software and future challenges. Section 5 concludes the paper.

2. Software description

The proposed system is based on a distributed client–server web architecture. The frontend is a thin client connected to the backend that consists of four layers: web server, two intermediate application servers responsible for managing administrational and meta-data, and the final layer responsible for rendering DICOM images. The details of the architecture implementation, software functionalities and key benefits are described in the rest of this section.

2.1. Software architecture

The frontend acts as a user interface through which the backend functionalities are accessed. It is implemented in the TypeScript programming language using Angular framework [15] with Material components [16], and translated by the Angular CLI tool into JavaScript code that can run in a web browser.

The backend is divided into four layers. The first layer is the web server, which provides the frontend files (HTML5, CSS, JavaScript). The toplevel server is the second layer, responsible for storing and managing data about physicians, patients, and various metadata (IP addresses and server ports at the institution layer, supported types of DICOM records) common to a group of medical institutions, e.g., the entire county. The institutional server is the third layer, responsible for storing and managing data at the medical institution level: radiological findings, rendering servers' metadata (IP address, port), and metadata about DICOM images stored on servers within the institution (location, MD5 sum). Radiological findings contain only the link to the DICOM record, not the record itself. The rendering node is the fourth layer of the backend, responsible for rendering slices of DICOM records and volume rendering. It relies on the image characteristics metadata in the DICOM header for image processing. Other metadata from the DICOM header (about the patient, device, imaging process, etc.) are not analyzed since the entries are very different in each case and multiple usage options are available [17]. Instead, we rely on the data manually entered into the patient record stored in the database. The CockroachDB Core edition database [18] is used to store the data. Its main advantages are that the executable does not need to be installed, and it provides NoSQL features like easy replication, geo-distribution, and cluster creation [19]. Finally, DICOM records do not necessarily have to be stored on these servers but can also be stored on SAN or NAS [20] devices accessible from these servers.

Horizontal and vertical communication independence is proposed to minimize latency in remote rendering and ensure a seamless user experience. Vertical independence means that the frontend can connect directly to any of the backend layers depending on the functionality requested by the user, which is completely obfuscated from the user who does not know which layer of the backend they are interacting with. For example, radiological findings are loaded directly from the institution layer servers, and rendered images of DICOM records are obtained directly from the rendering. Communication between the frontend and each layer of the backend is done using RESTful architectural style by sending JSON messages. By avoiding an intermediate layer, the overall latency and load on the toplevel server are reduced. Horizontal independence means that there is no direct communication between backend servers. Consequently, the number of servers at each layer is not predetermined, and it is possible to set up additional servers in some layers if needed (i.e., upscale). Also, multiple toplevel servers can use the same database, or a database cluster can be created and connected to multiple toplevel servers. If DICOM records are stored on SAN or NAS devices, multiple nodes can be used. Fig. 1 illustrates the system architecture.

The entire backend is written in the Go programming language using libraries for OpenGL, linear algebra, parsing DICOM records, database access, etc. Everything is compiled and statically linked into a single self-contained binary. All static resources required to run each server and serve the frontend are also part of the final binary. In addition to the backend executable file, the following components of the architecture are required for the system to function properly: the configuration file, the database engine, and the database data files, as illustrated in Fig. 2. The configuration file is written in YAML format and provides basic parameters necessary to run the backend: IP address, port, path to database data files, path to DICOM records. This file must be modified for each new instance of the backend that is run.

System security

Due to data confidentiality requirements for healthcare applications [21], security solutions were implemented on frontend, database, transport layer, and file system.

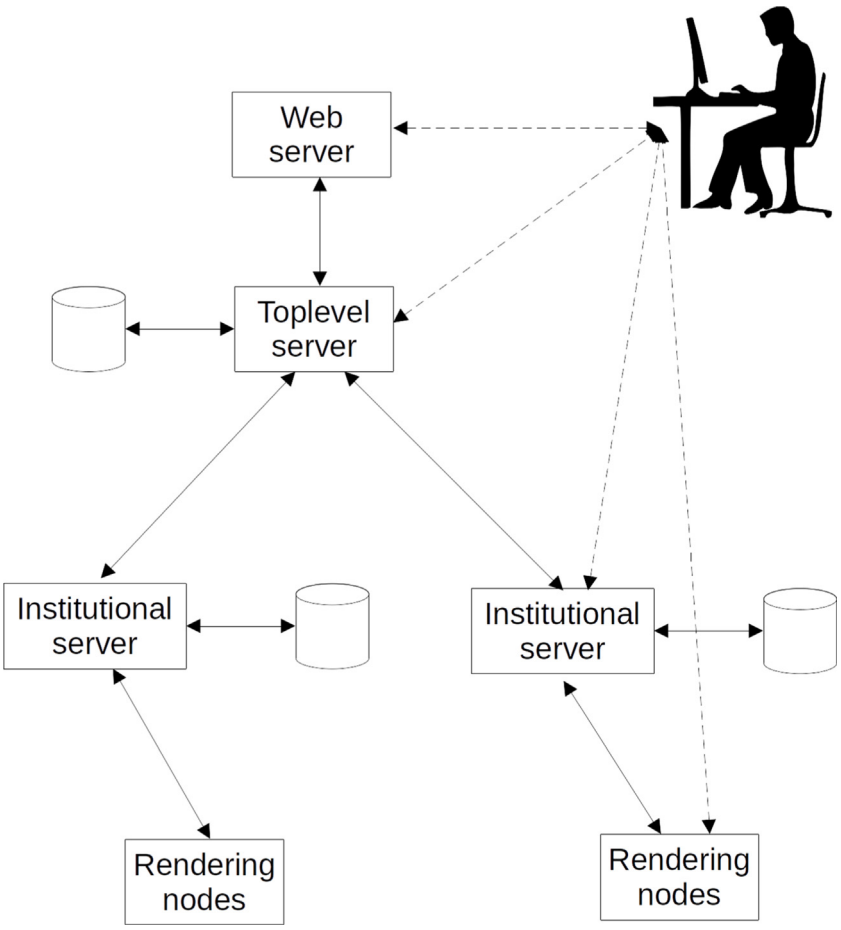

Fig. 1. System architecture overview.

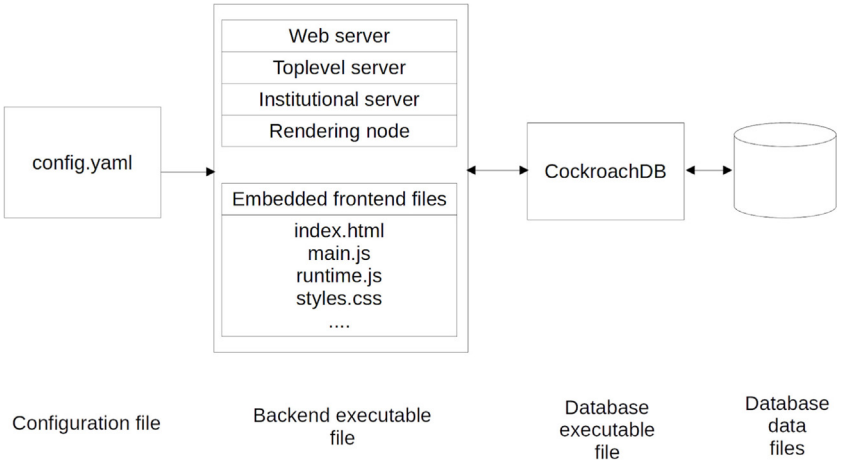

Fig. 2. System's executable and data files.

User passwords are stored in the database encrypted, in MD5 format. The password is encrypted on the frontend before it is sent to the backend over the network. This level of protection ensures that if a database containing passwords is compromised, the passwords cannot be misused to access other services on the Internet.

The next level of protection is implemented using JWT (JSON Web Tokens) [22]. The transmission of each message between the frontend and the backend contains a JWT token. It is used to protect the data from modification, not to ensure data confidentiality. The protection sum is generated using a secret key which resides on the backend and is never transmitted to the client. The frontend automatically renews the JWT token every few minutes while the user is logged in.

The third level of protection is securing communication through the transport layer. Only protocols that use encryption are supported: HTTPS, HTTP/2, and HTTP/3 [23]. The HTTP/2 and

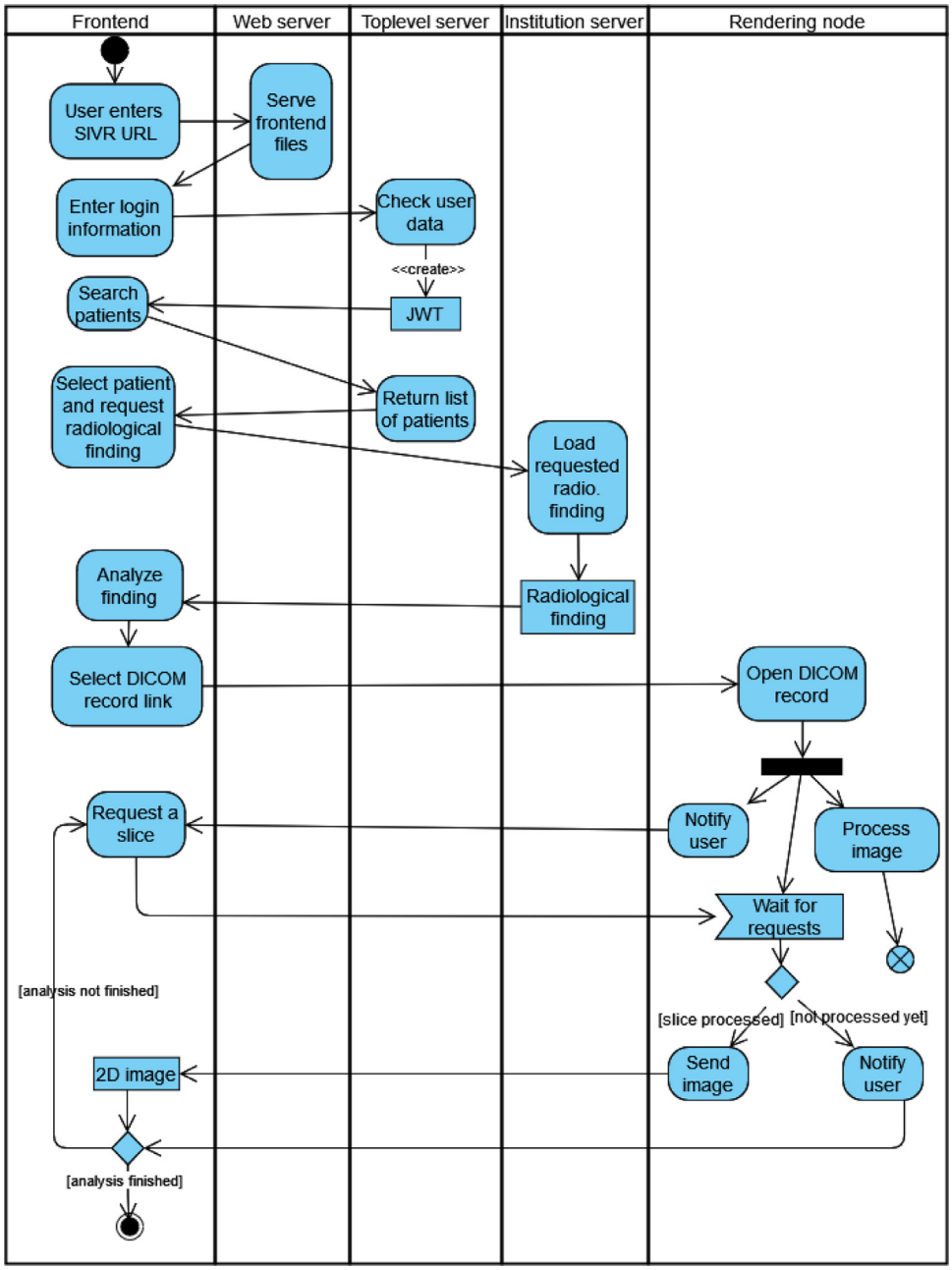

Fig. 3. Activity flow for loading DICOM records.

HTTP/3 protocols also improve system performance by multiplexing requests over a single connection.

The final layer of protection, the OpenZFS file system, is used to store DICOM records. It is an advanced file system that supports software RAID arrays, field portability between different architectures, logs, deduplication, compression, and encryption.

2.2. Software functionalities

The intended users of this system are physicians and administrators. All users can view their personal information and change passwords. Access to other functionalities depends on the user's authorization level and the user's context.

Functionalities available to physicians are:

- manage patient records,
- send DICOM images to a storage server,
- enter radiological findings,
- view DICOM images (2D slice by slice and 3D volume).

Functionalities available to system administrators are:

- manage system users (physicians),
- manage institutions,
- manage DICOM record types,
- manage rendering nodes.

Access to rendering nodes for storing DICOM records is limited by the institution to which the physicians or administrators belong. This restriction was implemented to limit data transfer over the Internet to improve system performance, as transfer speeds

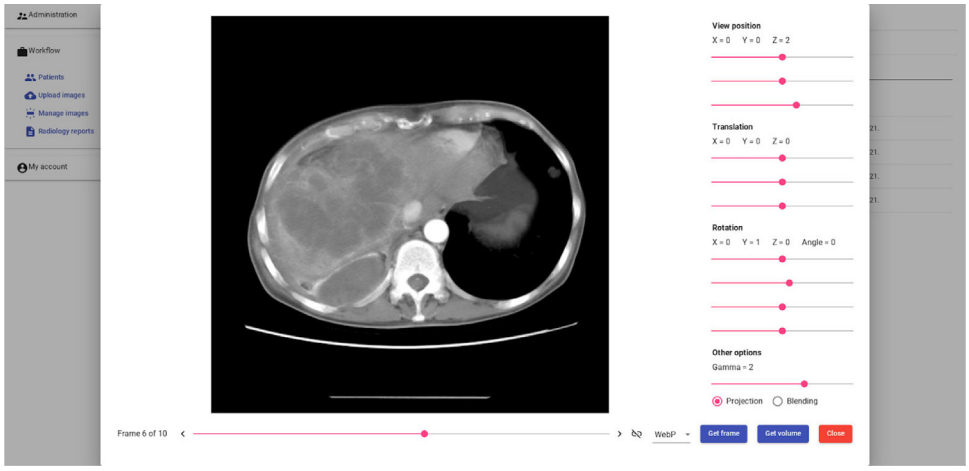

Fig. 4. An example of a remotely rendered slice of a DICOM image. The slider on the bottom is used to select the frame. The menu on the right is used to select the image format.

within facilities are much higher than over the Internet. However, access to stored images is not limited by institutions.

Image processing is performed entirely on the rendering nodes. Many algorithms for visualizing radiological images exist [24], but raycasting algorithms [25], a type of direct volume rendering algorithms [26], are the most used and hence implemented in this system. To hide the latency of rendering a DICOM record, the record is loaded slice by slice into the RAM of the rendering node. In RAM, they are decompressed and stored in a slice of an OpenGL 3D texture. The status of each loaded 3D texture slice is recorded. This way, the user receives rendered images of the first few slices almost immediately, while the rest of the texture slices are being processed. However, latency masking is only possible when displaying individual slices of DICOM records. This is not possible when displaying the volume, because the entire record must be loaded into a 3D texture.

3. Illustrative examples

In this section we describe one of the most common use cases: the physician retrieves a patient's record to examine radiological findings. The flow starts with a successful login, followed by a search for the patient's record. When the record is found, the user is notified that it was successfully loaded and can immediately start reviewing the record and request slices. The DICOM image is processed concurrently in the backend and prepared for rendering slices on demand. The user interacts with the system through the frontend, which connects to the appropriate backend layers depending on the functionality performed, as described in Section 2. We illustrate this scenario using a UML activity diagram in Fig. 3.

The search for patient records and radiological findings can be performed simultaneously on multiple servers at the facility level. Also, due to network latency and user response speed, the processing of most slices will likely be complete by the time the user issues the first request for a slice. In this way, disk latency is masked, giving a good impression of the system's responsiveness. Fig. 4 shows a slice of a DICOM image rendered remotely.

The performance in rendering slices (2D) and entire volumes (3D) was tested using DICOM records from publicly available repositories [27,28] on a system consisting of a web server, a toplevel server, an institution server, and a rendering node. All servers were deployed on a computer with Ubuntu 21.10 operating system and the following hardware configuration: Intel i7-11700 CPU (4.9 GHz), 64 GB DDR4 RAM, Intel UHD Graphics 750, and separate SSD drives for DICOM storage and the server executables. Network connectivity between the backend servers was simulated using Chromium (v96) web browser development tools [29] for three configurations:

1. without simulation,
2. 2G (D: 750 kbit/s, U: 250 kbit/s, LAT: 100 ms),
3. 4G (D: 4 Mbit/s, U: 3 Mbit/s, LAT: 20 ms).

Four DICOM records were selected as test cases with properties (original and rendered in 2D and 3D), Table 1.

Performance test results for 2D and 3D rendering for all three configurations are compared in Fig. 5. The setup without network ("w/o") is used to determine the time required to render the image, and it can be observed that the rendering time mainly depends on the compression algorithm (PNG vs. WebP). Rendering 3D volumes takes longer, which is to be expected because it is computationally more demanding than rendering layers. Image loading times from the hard drive are given in Table 2. It can be concluded that they do not depend on the form of the file (compressed, encrypted, etc.), but only on the file size.

For 2G network simulation, the rendered image size has an impact on the transmission time. The WebP algorithm, which was the slowest in rendering, produced smaller images compared to PNG and overall achieved better time. Therefore, the rendering time under real network conditions is masked by the latency and bandwidth of the network. The same is observed for the 4G network simulation.

4. Impact

The proposed system provides fast and easy access to DICOM medical images over the web without requiring large initial investments. Deployment of the backend requires no installation and is easily scalable. Physicians only need a web browser and no high-throughput internet connection to access the frontend. The system offers a high level of security and data protection: communication over an encrypted connection, and all data on disk is also encrypted.

Although the system is not yet in commercial use, we see its impact in the future primarily in improving the quality of medical care in isolated and difficult to reach areas, and in healthcare facilities with a lack of appropriate professional workers. One possible application would be integration with the RIS system (Radiology Information System) of the Croatian Institute of Emergency Medicine to expand its current teleradiology capabilities [30]. It may also be useful for academic purposes for students

Table 1
Image properties.

Image	DICOM				2D rendering		3D rendering	
	Rows	Cols	Slices	File size [MB]	PNG size	WebP size	PNG size	WebP size
A	512	512	10	5	76.00	23.40	95.60	29.20
B	512	512	54	27	158	68.9	185	62.3
C	512	512	555	277	95.1	32.7	146	48.6
D	1996	2457	96	898	94.4	22.50	88	22.30

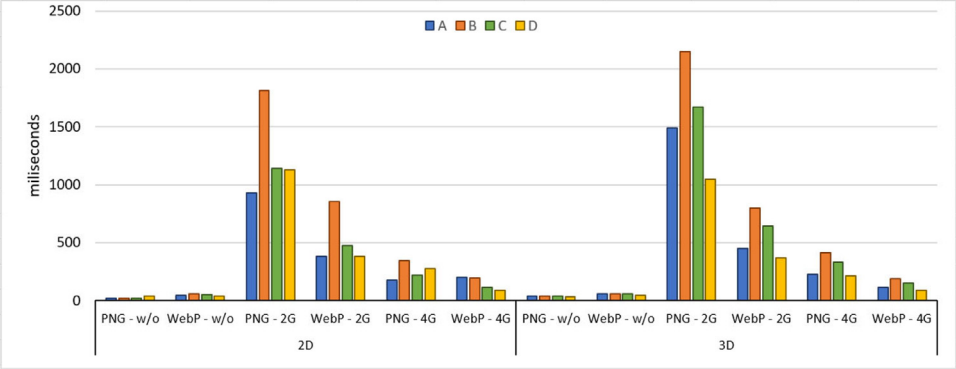

Fig. 5. Performance comparison of 2D and 3D rendering.

Table 2
Loading time from disk (in seconds).

Image	Raw	Compressed	Encrypted	Compressed & Encrypted
A	0.09	0.08	0.08	0.09
B	0.43	0.44	0.45	0.43
C	3.63	3.67	3.57	3.59
D	11.27	11.45	11.07	11.26

and young scientists in the field of medical imaging, who often do not have sufficiently powerful computers to process the DICOM format.

5. Conclusions

We have described the main features of a web-based medical imaging software with scalable architecture that enables seamless medical image examination. Future research could investigate alternatives to OpenGL like Vulkan [31] and OpenCL [32, 33] to enable multithreaded execution. Using OpenCL would allow servers without graphics hardware to be used as rendering nodes or use both the CPU and specialized graphics hardware simultaneously. Additionally, image processing on rendering nodes could be further customized by introducing dynamic brightness adjustment instead of using the window center and width values predefined in the DICOM image. This would facilitate spotting details that may not be visible with the predefined settings.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Vidal FP, et al. Principles and applications of computer graphics in medicine. Comput Graph Forum 2006;25(1):113–37. http://dx.doi.org/10.1111/j.1467-8659.2006.00822.x.
[2] The DICOM Standard – Current Edition, https://www.dicomstandard.org/current.
[3] Genereaux BW, Dennison DK, Ho K, et al. DICOMweb™: Background and application of the web standard for medical imaging. J Digit Imaging 2018;31:321–6. http://dx.doi.org/10.1007/s10278-018-0073-z.
[4] Berkowitz SJ, Wei JL, Halabi S. Migrating to the modern PACS: Challenges and opportunities. RadioGraphics 2018;38(6):1761–72. http://dx.doi.org/10.1148/rg.2018180161.
[5] Caffery LJ, Clunie D, Curiel-Lewandrowski C, et al. Transforming dermatologic imaging for the digital era: Metadata and standards. J Digit Imaging 2018;31:568–77. http://dx.doi.org/10.1007/s10278-017-0045-8.
[6] IAEA. Worldwide implementation of digital imaging in radiology. IAEA Hum Health Ser 2015;(28). http://www-pub.iaea.org/MTCD/Publications/PDF/Pub1647web.pdf.
[7] Ranschaert ER, van Ooijen PMA. Sharing imaging data. In: van Ooijen PMA, editor. Basic Knowledge of Medical Imaging Informatics. Imaging Informatics for Healthcare Professionals. Cham: Springer; 2021, http://dx.doi.org/10.1007/978-3-030-71885-5_6.
[8] Lee KT, et al. An internet-based telemedicine system. IJCSNS Int J Comput Sci Netw Secur 2007;7(1):51.
[9] Ahmad W, Ahmad U. Role of radiology in COVID-19 pandemic and post COVID-19 potential effects on radiology practices. Indian J Radiol Imaging 2021;31(1):196–7. http://dx.doi.org/10.4103/ijri.IJRI_536_20.
[10] Blazona B, Mihajlovic Z. Visualization service based on web services. J Comput Inf Technol CIT 2007;15(4):339–45.
[11] Moraes T, Amorim P, Silva J, Pedrini H. Web-based interactive visualization of medical images in a distributed system. In: Proceedings of the 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications, Prague. Czech Republic; 2019, p. 346–53. http://dx.doi.org/10.5220/0007626103460353.
[12] Arbelaiz A, Moreno A, Kabongo L, Diez HV, Alonso AGarcía. Interactive visualization of DICOM volumetric datasets in the web - providing VR experiences within the web browser. In: Proceedings of the 12th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications, Porto, Portugal. 2017, p. 108–15. http://dx.doi.org/10.5220/0006154801080115.
[13] Egger J, Wild D, Weber M, et al. Studierfenster: an open science cloud-based medical imaging analysis platform. J Digit Imaging 2022;35:340–55. http://dx.doi.org/10.1007/s10278-021-00574-8.
[14] Jozić K, Jović A, Mihajlović Ž. Seamless remote rendering of DICOM images. In: Bilof Randall, editor. Proceeding of 14th International Conference on Advanced Computer Theory and Engineering (ICACTE 2021), Hangzhou, China. IEEE-CPS. 2021, [in press].
[15] Google. Angular, https://angular.io.
[16] Google. Angular Material UI component library, https://material.angular.io/.
[17] Barufaldi B, Zuckerman SP, Medeiros RB, Maidment AD, Schiabel H. Characterization of the imaging settings in screening mammography using a tracking and reporting system: A multi-center and multi-vendor analysis. Phys Med. 2020;71:137–49. http://dx.doi.org/10.1016/j.ejmp.2020.02.018.

[18] Labs Cockroach CockroachDB. https://www.cockroachlabs.com/.
[19] Freire SM, Teodoro D, Wei-Kleiner F, Sundvall E, Karlsson D, Lambrix P. Comparing the performance of NoSQL approaches for managing archetype-based electronic health record data. PLoS One 2016;11(3):e0150069. http://dx.doi.org/10.1371/journal.pone.0150069.
[20] Lee G. Storage networks. In: Cloud Networking. Elsevier; 2014, p. 139–61. http://dx.doi.org/10.1016/B978-0-12-800728-0.00008-4.
[21] Desjardins B, Mirsky Y, Ortiz MP, et al. Dicom images have been hacked! now what? Am J Roentgenol 2020;214(4):727–35. http://dx.doi.org/10.2214/AJR.19.21958.
[22] Internet Engineering Task Force (IETF). JSON Web Token (JWT), https://datatracker.ietf.org/doc/html/rfc7519.
[23] The IETF QUIC Working Group. QUIC, https://quicwg.org/.
[24] Zhang Q, Eagleson R, Peters TM. Volume visualization: a technical overview with a focus on medical applications. J Digit Imaging 2011;24(4):640–64. http://dx.doi.org/10.1007/s10278-010-9321-6.
[25] Movania MM. OpenGL Development Cookbook. first ed.. Packt Publishing; 2013.
[26] Hansen C, Johnson CR. Visualization Handbook. Elesevier; 2014.
[27] NEMA repository. ftp://medical.nema.org/medical/dicom/Multiframe/CT/nemamfct.images.tar.bz2.
[28] Clunie DA. UPMC Breast Tomography and FFDM Collection, https://www.dclunie.com/pixelmedimagearchive/upmcdigitalmammotomocollection/index.html.
[29] Google Developers. ChromeDevTools, https://developer.chrome.com/docs/devtools.
[30] Croatian Institute of Emergency Medicine, Teleradiology. https://www.hzhm.hr/en/telemedicine/teleradiology.
[31] The Khronos® Group Inc. Vulkan, https://www.vulkan.org/.
[32] The Khronos® Group Inc. OpenCL, https://www.khronos.org/opencl/.
[33] Stone JE, Gohara D, Shi G. OpenCL: A parallel programming standard for heterogeneous computing systems. Comput Sci Eng 2010;12(3):66–72. http://dx.doi.org/10.1109/MCSE.2010.69.