Future Generation Computer Systems 158 (2024) 530–544

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Distributed realtime rendering in decentralized network for mobile web augmented reality

Huabing Zhang [a,b], Liang Li [a,b,*], Qiong Lu [b], Yi Yue [c,d], Yakun Huang [e], Schahram Dustdar [f]

[a] Key Lab of Film and TV Media Technology of Zhejiang Province, Hangzhou, Zhejiang 310018, China
[b] School of Media Engineering, Communication University Of Zhejiang, Hangzhou, Zhejiang 310018, China
[c] China Unicom Research Institute, Beijing, China
[d] National Engineering Research Center of Next Generation Internet Broadband Service Application, Beijing, China
[e] State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China
[f] Distributed Systems Group, Technische Universität Wien, 1040 Vienna, Austria

ARTICLE INFO

Keywords:
Mobile web-based augment reality
Real-time rendering computing
Decentralized network
Distributed collaborative computing
Mobile computing
Device-to-Device

ABSTRACT

Low-latency, real-time rendering of 3D objects is critical for mobile web-based augmented reality (MWAR) applications. While cloud-based or edge-based server rendering offloading can reduce the latency for mobile devices, a high volume of user requests in user aggregation scenarios can overload servers and transmission channels. It can result in a poor user experience due to resource consumption and high latency. This paper presents a decentralized and collaborative real-time rendering offloading network architecture (CRCDnet) to address this issue. CRCDnet makes contributions in the following three areas: (1) In the user aggregation scenarios, mobile devices that perform the same services are used as service nodes to perform the offloading of rendering computing and a collaborative rendering computing network is established. (2) For data exchange in decentralized networks, a data sharing middle layer based on blockchain key indexes separates sensitive service data and ensures a secure, reliable exchange mechanism and efficient data exchange. (3) A data request and computing offloading scheduling approach is proposed for the collaborative rendering computing network to optimize the rendering computation delay.

1. Introduction

Augmented reality (AR) is a popular technology that enhances users' perception of the real world. However, native-based AR solutions suffer from issues with portability and service construction costs. Mobile web-based augmented reality (MWAR) is a lightweight and cross-platform alternative that is expected to enable users to interact with the real world conveniently [1]. In user aggregation scenarios, MWAR can quickly establish a rich information interaction with external objects through a wide range of user communication channels, as shown in Fig. 1. However, the lack of computing resources, Web security restrictions, and interpreter execution mechanism of mobile device's Web browsers lead to high rendering response delay and poor media rendering quality [2].

To address the problem of insufficient computing resources in mobile Web browsers, real-time computing offloading based on remote servers such as cloud or edge servers is a popular solution [3]. In this approach, the remote server performs rendering computing in real-time to generate multimedia streams for mobile devices, which are then loaded onto the mobile browser via the HTTP protocol [4,5]. However, current approaches for rendering computing offloading are limited in the user aggregation scenario. The first approach uses public cloud servers with sufficient computing resources and scalability advantages to perform low-latency rendering computing for mobile devices. However, this approach consumes more communication resources than traditional 3D model files for rendered data transmission, especially when multiple users request real-time rendering. This leads to delivery delays that users cannot tolerate. The second approach uses edge servers with high-bandwidth connections to provide low-latency and high-quality rendering computing offloading services. However, real-time rendering computing requires high computing resources, particularly for large 3D scenes and complex user interactions [6]. When multiple mobile devices send many rendering computing requests in the short term, the mobile edge servers undertaking render computing or cache management computing experience a high delay in responding. The third approach is collaborative computing offloading deployed between cloud servers, edge servers, and clients, which can address

* Corresponding author at: Key Lab of Film and TV Media Technology of Zhejiang Province, Hangzhou, Zhejiang 310018, China.
E-mail addresses: huabingzhang@cuz.edu.cn (H. Zhang), liliang@cuz.edu.cn (L. Li), luqiong@cuz.edu.cn (Q. Lu), yuey80@chinaunicom.cn (Y. Yue), ykhuang@bupt.edu.cn (Y. Huang), dustdar@dsg.tuwien.ac.at (S. Dustdar).

https://doi.org/10.1016/j.future.2024.04.050
Received 27 April 2023; Received in revised form 24 April 2024; Accepted 25 April 2024
Available online 27 April 2024

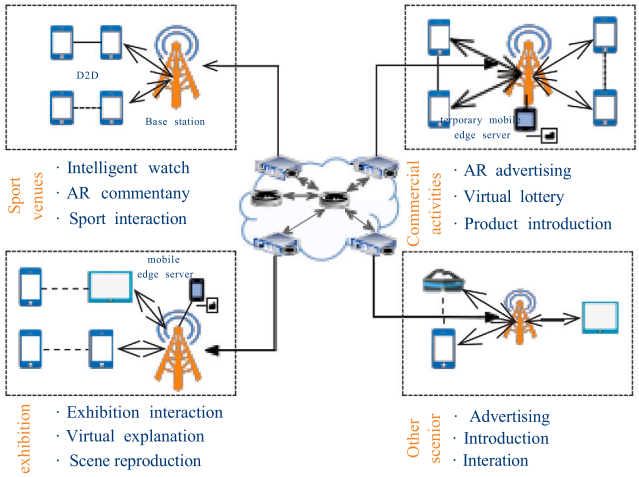


Fig. 1. Augmented reality in the user aggregation scenarios.

the issue of computing and communication costs in user aggregation scenarios. However, the dependence of collaborative approaches on core computing devices and communication links can also bring huge reliability issues to computing offloading in user aggregation scenarios. Furthermore, deploying mobile edge servers can incur additional deployment costs for operators, particularly for temporary active service scenarios.

In this paper, our main motivation is to address the computing and transmission pressure caused by centralized offload approaches by proposing a decentralized rendering approach. In user aggregation scenarios, many mobile devices with similar or varying service purposes and real-time rendering computing services can be shared based on computing resources or rendered data. Inspired by this, we introduce a distributed collaborative real-time rendering computing offloading network (CRCDnet) for MWAR. However, implementing this network presents three key technical challenges:

- The centralized rendering computing offloading framework cannot provide low-latency rendering and computing offload services for application terminals in the peer-to-peer device architecture of user aggregation scenarios [7], which requires a decentralized approach.
- The existing data exchange approach based on blockchain is complex and can cause huge transmission delays, making it unsuitable for exchanging rendered data in a timely manner [8].
- The current computing offloading scheduling algorithm cannot effectively organize multiple mobile devices to perform collaborative real-time computing offloading [9], especially in multi-device collaborative service environments with random participation.

To overcome these challenges, we present an approach for constructing a decentralized real-time rendering computing service network using mobile devices as computing offloading devices. This approach is based on WebGL (Web Graphics Library)'s underlying rendering computing interface in a Web browser and builds a distributed collaborative rendering environment in a decentralized network. Our approach consists of three key components. First, we utilize a distributed computing approach to provide low-latency rendering computing services in user aggregation scenarios. Second, we introduce the concept of an index that uses blockchain to enable high-capacity data exchange between mobile devices. Lastly, we propose an optimal algorithm for constructing a data-sharing pool to accommodate multiple mobile devices' spontaneous and random participation in rendering computing offloading. The contributions of our work can be summarized as follows:

- We propose a collaborative real-time rendering offloading service framework based on a decentralized network and a multi-source rendering approach based on the WebGL rendering binary data. By aggregating the computing capability of mobile devices in user aggregation scenarios, the framework provides lower latency and higher quality real-time rendering services than other approaches based on WebGL rendering.
- We propose a data exchange approach based on the blockchain key index to facilitate the direct and efficient exchange of rendered data between mobile devices. This approach separates sensitive service data from desensitized rendered data, simplifies computing offloading requests, and ensures a secure and trustworthy mechanism for distributed exchange.
- We propose a collaborative-driven algorithm based on a data-sharing pool to schedule real-time rendering on distributed computing efficiently. All mobile devices connected to the data and resource sharing ring are considered part of the collaborative rendering system. The scheduling strategy for rendering computing is dynamically determined to optimize delays.

2. Preliminary

For real-time rendering computation offloading in user aggregation scenarios, collaborative and distributed rendering with decentralized networks is the main motivation to address the three challenges referred to in Section 1. Meanwhile, sharing rendering data between mobile devices and direct connections between mobile devices are the foundation of collaborative rendering computing based on decentralized networks. This section will discuss the decentralized network service paradigm, the direct connection approaches of mobile devices, and rendering data interaction approaches in a decentralized network service environment, providing a foundation for our work.

2.1. Decentralized network service

In the user aggregation scenarios, many users with the same service purpose will trigger real-time requests for computing or data resources to a remote server in a short amount of time and within a specific area. Once the 3D model is loaded and rendered, users will engage in personalized and differential interaction according to their individual needs. These time and space-intensive scenarios can lead to computing overload and communication congestion on the remote server. As a result, traditional centralized service architecture, such as Browser/Server(B/S) and Client/Server(C/S) structures, is unable to meet the requirements of constructing low-latency and stable services in such a highly time- and space-intensive user aggregation scenario.

The decentralized network service mode represents a paradigm shift in network service construction from centralization to decentralization. It is based on a decentralized network architecture, where many complex feedback loops exist between service terminals [10]. This decentralized network has several characteristics that make it particularly suitable for rendering computing offloading in user aggregation scenarios.

- **Distributing the computing pressure:** By allowing each device to provide computing resources and rendered data services, the computing pressure can be distributed across the network, avoiding overloading a single server [11].
- **Enabling short-distance communication:** Computing devices can directly communicate with each other using short distance communication channels, which can alleviate communication pressure between the central server and the edge base station [12].
- **Protecting users' privacy:** The decentralized network's information encryption and token mechanism based on blockchain technology can protect users' privacy while maintaining network operation [13].

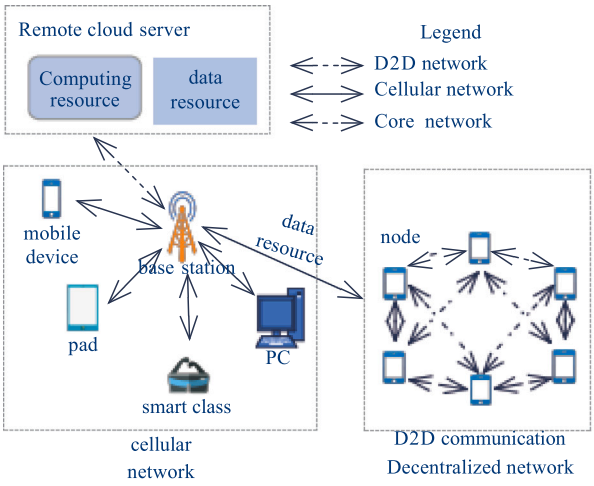


Fig. 2. Structure of decentralized network.

Based on these characteristics, a decentralized network service provides a promising opportunity to construct rendering computing offloading services for MWAR applications in user aggregation scenarios. In such a network, the mobile devices of different users can act as 3D model rendering computing and data service [10]. Each relevant mobile device loads 3D model file blocks and performs rendering computing according to a certain control strategy. The rendered data can then be communicated through short-distance channels such as Device-to-Device (D2D) or Wi-Fi Direct between devices, and the service client can aggregate the rendered computing data. Finally, the service client reorganizes the rendered computing data to implement the 3D scene construction of the MWAR application. The structure is shown in Fig. 2.

In this process, the construction of communication channels and the exchange of rendered data are the foundation of the decentralized collaborative rendering computing service network. The following subsection will analyze these works further.

2.2. Communication channel between devices

The direct and short-distance communication channel between devices is the foundation of the decentralized collaborative rendering service network [14]. Direct communication between mobile devices is not allowed in cellular networks, but with the rapid popularization of intelligent terminals and the explosive growth of network communication capacity, D2D, Wi-Fi Direct, flash LINQ, and Bluetooth technologies have become the main means of direct connection for mobile devices [15]. D2D is more flexible than other direct connection technologies between mobile devices that do not rely on infrastructure, as it can connect and allocate resources under the control of the base station and exchange information when there is no network infrastructure [16].

In a decentralized rendering service network, each mobile device can send and receive signals and automatically route (forward) messages. Participants in the network share a portion of their computing and data resources, including information processing, storage, and network connectivity [17]. These shared resources are directly accessible by other users without intermediaries. In a D2D communication network, each device simultaneously plays the roles of both a server and a client, as illustrated in Fig. 2. Mobile devices can detect each other's presence and independently form a virtual or physical group. By utilizing the proximity and data pass-through features of D2D, mobile devices can conserve spectrum resources and extend the application scenarios of MWAR.

Cache technology is crucial in satisfying the high-speed and low-delay requirements of rendered data transmission, particularly in the case of large-scene MWAR. The rendered data can be stored by special devices or auxiliary devices distributed throughout the network, which can be accessed by the local network buffer without requiring high throughput, as depicted in Fig. 2. This approach addresses the high communication costs associated with the initial rendering of a 3D model object or scene.

2.3. Data format of computing resource exchange

Multiple mobile devices collaborate in a decentralized network to render computation services or share rendered computation data. The rendered data from various sources needs to meet two conditions: firstly, it can be reused based on frames or media segments in the channel [18], and secondly, it can be serialized and computed on the Web browser Canvas container [19].

A shared channel is used for communication and data transmission in the decentralized distributed collaborative rendering service, involving multiple source rendering services and multiple target rendering services. To meet the requirements of real-time rendering computation, multiple mobile devices need to aggregate the already rendered data to the common rendering service targets in real-time through channel multiplexing. The divisibility and re-usability of the already rendered data become crucial for effective transmission [20]. By implementing the divisibility and re-usability of the already rendered data, multiple mobile devices can aggregate their rendering results in real-time to the common rendering service targets through channel multiplexing. This enables efficient collaborative rendering computation in a decentralized network environment, fulfilling the real-time rendering needs and enhancing overall rendering system performance and scalability [18].

Serialization computation of rendered data on the browser is a critical technological process. Rendering approaches based on continuous media elements like videos can provide client-side offloading services with lower latency and higher response efficiency. However, serializing the rendered video segments generated by multiple mobile devices on a mobile browser consumes substantial computational resources. This is because the computational capabilities of mobile browsers are relatively limited and need help to process and compute large amounts of video data efficiently. Furthermore, decentralized mobile devices cannot support real-time video streaming media rendering services, which limits the applicability of decentralized rendering computation approaches [19].

In this paper, we introduce a primitive approach. This primitive rendering offloading approach uses WebGL to leverage the computational resources of the browser for rendering 3D scenes. It converts the rendered computation data into binary data for exchange among different devices. WebGL loads the model's geometric information into memory, compiling vertex and fragment shaders to handle vertex coordinates and pixel color computation. Then, WebGL creates buffer objects to store 3D model data, such as vertex coordinates, texture coordinates, and colors. These buffer objects are bound to attribute variables of the vertex shader, and the rendering state of the shader is set. By invoking WebGL drawing functions, the scene and the 3D model are drawn and stored as binary data, which can be shared among devices in a decentralized network through mobile device network interfaces. Suppose the scene includes data such as animation; the scene and model states can be updated between each rendering frame, such as computing transformation matrices, updating material properties, or modifying 3D model data. Finally, by iterating the rendering steps in a rendering loop, the scene and 3D model are continuously updated and rendered to achieve real-time and continuous sharing of already rendered data. The rendering service targets can receive binary data from multiple sources and directly perform serialized computation, pushing the binary data to the browser's Canvas to meet the user's real-time interaction needs.

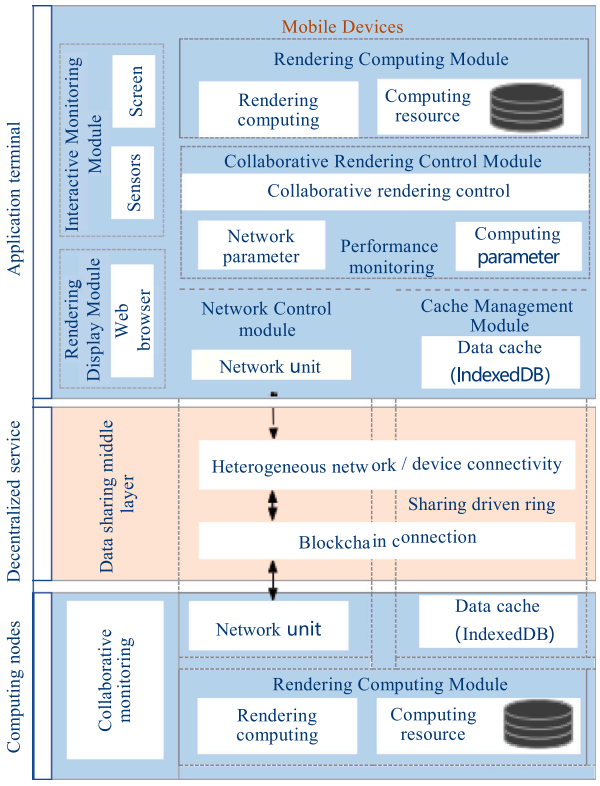


Fig. 3. Structure of decentralized and collaborative real-time rendering offloading network.

3. CRCDnet solution overview

CRCDnet proposes a distributed-based decentralized network framework for real-time rendering services in user aggregation scenarios for MWAR applications. It utilizes decentralized networks and D2D communication to minimize communication costs and speed up rendering response. Meanwhile, CRCDnet uses mobile devices or virtual machines based on mobile devices for rendering. CRCDnet comprises six parts, as shown in Fig. 3.

- **Interactive Monitoring Module:** The Interactive Monitoring Module monitors the user's interaction through feedback from mobile device sensor data and other mobile devices' feedback data through the network. It converts the user's interactive functions, such as zoom, translation, and rotation, into virtual camera parameters in the 3D scene. The Interactive Monitoring Module establishes the camera trajectory parameter sequence of the client interaction, which serves as the basis for rendering computing.
- **Rendering Display Module:** The Rendering Display Module collects the rendered data from the mobile devices and loads it into the Web browser. It creates a new canvas and uses the binary data to draw the corresponding image with the pixel data from multiple mobile devices. The canvas displays the corresponding rendered data in the interactive process through the media stream.
- **Network Control Module:** The Network Control Module establishes the communication link between the application terminal (Web browser) and the auxiliary mobile devices. When the service environment is initialized, CRCDnet establishes a decentralized network through a D2D connection between mobile devices. After establishing the service environment, the Network Control Module controls the transmission of camera interaction trajectory parameters, rendered data, and rendering control instruction data between the application terminal and the auxiliary rendering devices.
- **Rendering Computing Module:** The Rendering Computing Module stores the RGBA pixel values of the rendered data in a custom frame buffer by using the off-screen rendering approach of WebGL to generate binary streams. By default, the screen does not directly read the pixel data in the buffer, which is obtained through the specific API of WebGL and returned to the application terminal.
- **Cache Management Module:** When rendering the 3D model in the Web browser, CRCDnet uses indexedDB as the container for exchanging data, which is a transactional (nonrelational) database deployed on a browser for storing large amounts of structured data on the Web browser. The intermediate data (in the form of binary streams) of rendered data is exchanged through caches to reduce the rendering computing pressure of the Web browser. After receiving the request from the Web browser, the corresponding binary data is found from the indexed database and returned to the Web browser according to the position parameters of the received camera.
- **Collaborative Rendering Control Module:** The Collaborative Rendering Control Module is mainly used to control the collaborative rendering of each device in the decentralized network. The Collaborative Rendering Control Module is deployed on application terminals and each mobile device to monitor and collect network and computing parameters in the service environment. When the service environment is initialized or changed, the Collaborative Rendering Control Module controls each mobile device through the collaborative monitoring unit for rendering computing according to the pre-determined strategy through control instructions.

In CRCDnet's service control, establishing the decentralized network and sharing rendered data are controlled as two independent threads. The first mobile device that joins the service environment requests 3D model data from the server and constructs the service based on independent rendering. When a new mobile device joins CRCDnet, it broadcasts to nearby mobile devices to monitor if they have the same service. If mobile devices have the same service and the network has not been constructed, the network control module triggers the establishment of a D2D connection channel between the mobile devices to construct or join the collaborative rendering decentralized network. Then, CRCDnet calls the computing resources of each mobile device to determine the rendering strategy based on the computing parameters of existing mobile devices in the current network. When a decentralized rendering computing network exists, the new mobile devices register with the computing resource parameters (CPU cycle, cache, etc.) and join the current network as a computing device. The Collaborative Rendering Computing Module of the new mobile device broadcasts to the existing mobile devices to change the current computing strategy.

The collaborative, decentralized rendering network is critical in the CRCDnet service, providing the foundation for end-to-end collaborative rendering. In this process, the Web browser sends interaction requests to mobile devices, which then perform computation or data caching sharing based on the camera parameters. Within the collaborative rendering decentralized network, requests and transaction information between mobile devices are traded through the blockchain. Rendered binary data are exchanged and controlled by the Collaborative Rendering Module and then routed to the application terminal through the D2D network. Finally, the Web browser obtains the corresponding binary data and pushes it on the canvas, implementing the collaborative rendering service process. To ensure the smoothness of the collaborative rendering, it is essential to understand its construction process thoroughly. Therefore, the following sections (Section 4 and Section 5) will provide a detailed introduction to this process.

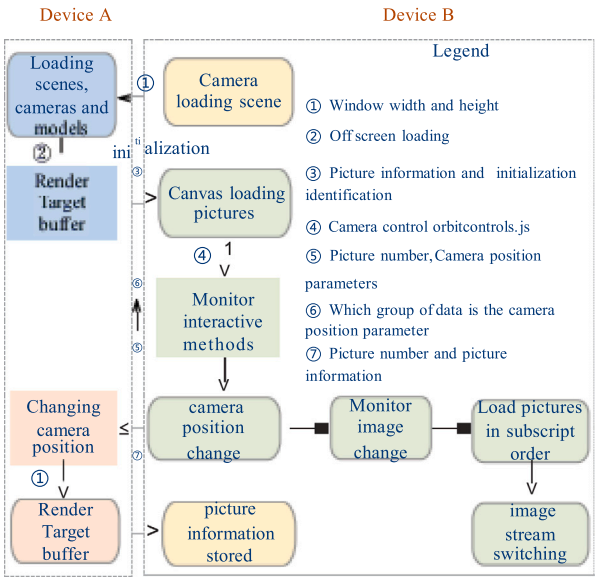


Fig. 4. Single device rendering for another single device.

4. Rendering computing offloading between mobile devices

In user aggregation scenarios, MWAR real-time rendering offloading often requires mirroring rendering computations between mobile devices. However, in the case of CRCDnet, the main task is to exchange camera parameters and the rendered data between mobile devices in real-time. This section, therefore, focuses on the challenges of distributed rendering computing and explains the principle of sharing rendering parameters among mobile devices. Finally, we introduce the solution of multi-device collaborative rendering.

4.1. End-to-end collaborative rendering computing offloading

In the collaborative real-time rendering computing of CRCDnet, efficient sharing of computing resources between mobile devices is crucial. This is achieved by establishing a mirroring of rendering computing between mobile devices, which enables the efficient initialization of the computing environment between mobile devices and facilitates the exchange of camera parameters and rendered data. This subsection focuses on the rendering computing mirroring between mobile devices, assuming that a single device serves as a computing offloading server to provide rendering computing services to another single device, as depicted in Fig. 4. CRCDnet can render the 3D model in images frame-by-frame on canvas or other computing units.

In the example illustrated in Fig. 4, $Device_A$ provides assistance to $Device_B$ in rendering, but does not display the 3D model. $Device_B$ obtains the rendering computing of $Device_A$ and loads the 3D model images frame-by-frame. In the following discussion, we refer to $Device_B$ as the application terminal and $Device_A$ as the auxiliary mobile device.

In CRCDnet, after loading the camera parameters, $Device_B$ records the window width (W_{width}) and height (W_{height}) and communicates with $Device_A$ through the D2D channel to request the initialization image for the 3D model. $Device_A$ loads the 3D model off-screen based on the received message, and instead of rendering the 3D model directly to the screen, it is stored in a screen description function. CRCDnet executes the browser's 3D rendering interface to load pixel data rendered off-screen into a binary array instance. The binary information is converted into Base64 code and sent to $Device_B$. Upon receiving the Base64 code and converting it into binary information, $Device_B$ transforms it into a uint-8 array (an 8-bit unsigned integer fixed array). Then, a new canvas is created on $Device_B$, which can be obtained through the Web browser's rendering approach to get a 2D object. It enables the initialization 3D model image to be successfully displayed on $Device_B$.

After the user interaction, the camera parameters are obtained by monitoring the location of the user interaction in $Device_B$'s Web browser. The CRCDnet uses a tool (*OrbitControls.js*) to control the 3D model object or scene, enabling zooming, panning, and rotation, changing the camera parameters rather than the scene. The $Device_B$ Web browser then encapsulates the camera's position parameters and the current window's width and height into an object and assigns a number to record which image it corresponds to.

After the off-screen rendering as initialization, $Device_A$ converts the binary information into Base64 encoding and sends the corresponding image identification package to $Device_B$. Upon receiving the package, $Device_B$ checks the identification to see if the media fragment is the next to be played. If it is, playback starts to cycle back while simultaneously checking whether the next picture in the picture array has been obtained. If the media fragment is not the next one to be played, it will save the picture array or media fragment with the corresponding subscript.

It should be noted that, in this Subsection, we introduce a more primitive approach. This primitive approach offloads rendering computation by utilizing the computational resources of the Web browser through WebGL for rendering the 3D model. Some rendering computation approaches based on images or binary data streams exhibit better performance in terms of latency, we will further discuss in Section 6.5.

4.2. D2D collaborative rendering data sharing

As discussed in Section 2.3, the client's Web browser can leverage caching technology to exchange intermediate rendered data and reduce the computing pressure on mobile devices when rendering a 3D model. CRCDnet utilizes indexedDB as the container for exchanging data of collaborative rendering computing in a cache-based rendered data sharing scheme, as shown in Fig. 3. Similar to Section 4.1, $Device_A$ performs off-screen rendering and saves the rendered data in the local cache to assist $Device_B$ in rendering. $Device_B$ obtains the matching image information in the local cache through the D2D communication channel with $Device_A$. Additionally, $Device_B$ communicates with $Device_A$ using the D2D communication channel to send the location parameters of the camera after initialization or user interaction, as well as the sequence identification of the pictures. The sequence identification of the picture is a unique number assigned to each picture or fragment, representing the order in which they are displayed or played. $Device_B$ uses this sequence identification to identify the corresponding data in the local cache for rendering. The location camera parameters of $Device_B$ are defined as follows, where 'position' represents the three-dimensional spatial coordinates of the camera, and 'rotation' represents the polar coordinates of the camera's spatial rotation angle.

```
camera:{
        position:{
                x:0,
                y:0,
                z:0
        },
        rotation:{
                x:0,
                y:0,
                z:0
        }
}
```

During the rendered data sharing process, $Device_A$ receives the location camera parameters, opens the indexedDB, and searches for the rendered data that matches the camera's location parameters. If the data is found, the rendering result and corresponding image sequence

identification are packaged and sent to $Device_B$. $Device_B$ monitors the picture acquisition and checks the picture identification. If the picture is the next one to be played, the playback cycles back while checking whether the next picture in the picture array has been obtained. Otherwise, CRCDnet saves the rendered data in the picture array with the corresponding subscript, generating a media stream fragment.

4.3. Multi collaborative rendering computing

In a service environment, multiple mobile devices are required to provide computing offloading or rendered data-sharing services for various devices to achieve collaborative rendering. For instance, multiple mobile devices can provide services for a single mobile device. Similar to Sub Section 4.1, multiple $Device_{A0}$, $Device_{A1}$, ..., $Device_{An}$ jointly perform rendering computing service for $Device_B$. The construction of multi-device collaborative rendering offloading provides data services for a single mobile device in two aspects, i.e., computing resource sharing and computing data sharing. To effectively control multiple mobile devices to provide rendering service, it is necessary to identify the mobile devices on the common data connection channel and sort various devices on the $Device_B$ side. For complex media sequence computing of picture-based rendering offloading, the auxiliary computing device needs to perform generation computing of media stream segments. We refer to the approach proposed in the literature [21] for this purpose. This literature proposes a low-complexity framework based on media stream segments to minimize the computing latency for all tasks. This subsection mainly discusses the control of multi-source media stream segments for collaborative rendering computing offloading. The service process of rendering computing is as follows:

- $Device_B$ loads the scene and records the window's width (W_{width}) and height (W_{height}) as rendering computing parameters. Then $Device_B$ communicates with auxiliary mobile devices queue $Q(Device_{Ai}, 0 \leq i \leq n)$ (including $Device_{A0}$, $Device_{A1}$, ..., $Device_{An}$) through the D2D communication channel and initializes the sending of the message 'Ping' along with the window parameters (W_{width}, W_{height}).
- Upon receiving 'Ping' and the window parameters, $Q(Device_{Ai}, 0 \leq i \leq n)$ loads the scene and camera according to the window parameters and sends 'helper_Ping' along with the user ID to $Device_B$.
- $Device_B$ receives 'helper_Ping' and the user ID, then adds the user ID to an array called $helperArr$ based on the order received, and returns the subscript index of the user ID in $helperArr$ to the device.

For instance, suppose there are three auxiliary computing devices, denoted as $Q(Device_{Ai}, i \leq 2)$, with user IDs $helper_a$, $helper_b$, and $helper_c$. Upon receiving the "Ping" from $Device_B$, $Q(Device_{Ai}, i \leq 2)$ start sending "helper_Ping" and user ID to $Device_B$. The sequence in which $Device_B$ receives 'helper_Ping' and user ID is $helper_c$, $helper_b$, and $helper_a$. Consequently, the indexes received by $Q(Device_{Ai}, i \leq 2)$ are 2, 1, and 0, respectively. When $Device_B$ initializes the 3D model (i.e., obtains the first media stream fragment of the 3D model) and monitors the interactive movement, it acquires the position parameters of the camera. It appends them to an empty initialized array, $positionArr$. Subsequently, $Device_B$ assigns its number as $0, 1, 2$ based on its subscript in the array. Assuming that each device in $Q(Device_{Ai}, i \leq n)$ has nearly equal computing resources and an equal chance of performing the same computing tasks, the following sections discuss the approaches of collaborative rendering computing and rendered data sharing.

- **Collaborative rendering computing offloading:** $Device_B$ sends a request to $Q(Device_{Ai}, 0 \leq i \leq n)$ to obtain the corresponding binary data information. The following principles apply to the binary data and equipment: The device with ID $helperArr[M\%helperArr.Length]$ encapsulates the binary data rendered by the 3D model with the received number and sends it back to $Device_B$ through off-screen rendering. In other words, $Device_{helperArr[M\%helperArr.Length]}$ processes the i, $i+helperArr.Length$, $i + helperArr.Length * 2$, ..., $i + helperArr.Length * (M helperArr.Length)$ fragments in the binary data. (The changes in the 3D model during mouse movements are reflected in the switching of binary data.) $Device_B$ monitors the acquisition of binary data and checks the fragment number. If the binary data is the next one to be played, the playback starts and contains whether the next fragment has been acquired.
- **Collaborative cache rendering computing output-data sharing:** For each fragment, $Device_B$ sends the parameters and data to the device corresponding to the user ID in $helperArr$. It queries whether there is a corresponding binary data fragment. If the query finds the fragment, the query ends, and the system checks the binary data fragment ID. If the fragment is the next one to be played, CRCDnet starts the playback and checks whether the next binary data has been obtained. Otherwise, the system saves the fragment in the corresponding array index.

The above assumptions assume that each device has roughly equivalent computing resources and equal opportunity to perform the same computing tasks. However, in the actual service environment, the computing capabilities of mobile devices can vary. A greedy algorithm can allocate more rendering tasks to devices with stronger computing capabilities to address this issue. While we will not delve into a detailed analysis of this approach here, relevant literature [22,23] suggests its effectiveness in improving rendering efficiency.

5. Construction of collaborative rendering decentralized network

To enable distributed collaborative rendering services, it is essential to establish a reliable network connection between mobile devices and ensure the sustainability of the collaborative rendering service. In this section, we will discuss the key features of decentralized networks for collaborative rendering computing.

5.1. Construction of rendering computing network

During end-to-end collaborative rendering offloading, it is crucial to establish a robust network connection between mobile devices to sustain the collaborative rendering service. In a decentralized service network, each node has a high degree of autonomy and can provide computing resources and data services for other devices. By distributing the computing pressure of the server to nearby mobile devices, we can prevent the remote server from bearing too much computing pressure. Rendering mobile devices can skip the remote server center and directly carry out short-distance communication between devices, addressing the issue of excessive communication pressure between the remote central server and the scene edge base station. The core of the decentralized network is blockchain technology, and building a blockchain that carries collaborative rendering computing services in user aggregation scenarios is another core task of CRCDnet implementation.

The construction of the decentralized collaborative rendering computing service network follows the construction approach of blockchain. The network mainly carries requests and transaction information. It maps the mobile device's internal network address and port to the external network address and port using the Universal Plug and Play (UPnP) or nat-pmp approach. Rendered data is exchanged using a structured D2D communication network. The concrete implementation involves two steps: (1) surrounding devices return a data packet containing an array of TCP connection ports and IP addresses of many neighbor devices after receiving a request, and (2) the mobile device

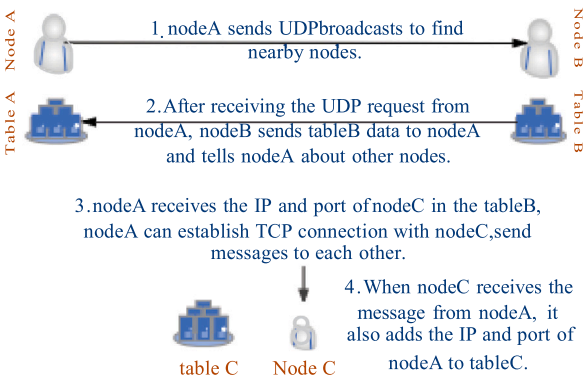


Fig. 5. Establishment of the decentralized network.

communicates with surrounding devices through the D2D communication network connection mode, using the IP and port sent by the surrounding devices to realize the communication of interactive data, control data, and rendering data. Fig. 5 illustrates the establishment process of the decentralized collaborative rendering computing service network.

The decentralized network utilizes the blockchain hash for data storage and management. To ensure non-repetition of the hash, devices (IP and machine port) and keys (resource identification) are mapped to the same space. SHA-1 is selected as the hash function, generating a space of 2^{160}, each of which is a large integer of 16 bytes. These integers can be connected end-to-end to form a ring that is arranged clockwise. Devices and keys are hashed to the ring, and the nearby device maintains each key (resource). The state of the whole D2D communication network is a virtual ring. This way, CRCDnet constructs a blockchain for collaborative rendering computing services.

In the collaborative rendering computing service blockchain, the mechanisms for devices joining, exiting, and becoming invalid are essential for maintaining the service. Following the traditional blockchain maintenance approach, the mechanisms for device joining, exiting, and failure are described as follows:

- **Device Join:** In this process, a new device, $Device_{new}$, needs the assistance of a known device called the wizard. Any device in the Chord network can play this role. The joining process consists of two stages: the new device's join operation and its discovery by other devices. Here are the steps involved:
  - $Device_{new}$ requests the wizard to find its successor, $Device_{or0}$, and initializes $Device_{new}$'s successor as $Device_{or0}$.
  - $Device_{new}$ connects to its successor, $Device_{or0}$.
  - The predecessor device of $Device_{or0}$, $Device_{or1}$, connects with $Device_{new}$ and disconnects from $Device_{or0}$.
- **Device Exit:**
  We use the chain list structure in the decentralized network to manage device exits. Assuming the exit device is $Device_{exit}$, and the predecessor and successor devices are $Device_{pre}$ and $Device_{suc}$, respectively, the following steps are taken:
  - Change the predecessor device of $Device_{suc}$ to $Device_{pre}$.
  - Change the successor device of $Device_{pre}$ to $Device_{suc}$.
  - Delete $Device_{exit}$ from the successor list of its predecessor.
- **Device Failure:**
  To address the issue of devices leaving the network suddenly without informing other devices, we adopt the following approaches:
  - Each device in the decentralized network periodically detects its preceding and following devices.
  - If a device detects its successor device has failed, it finds the first available device replacement from the successor chain list.
  - The device sets itself as the predecessor of the first available device.

A solution based on the cache invalidation mechanism has been introduced in decentralized network environments for distributed rendering computation. This solution aims to address the issues of data consistency and cache consistency among nodes and leverage the advantages of edge servers to accelerate the rendering computation process [24]. Initially, each node establishes a cache on an edge server and replicates the chunks of rendering tasks within the cache, enabling proximity access and high-speed computation capabilities. When a node needs to access cache data, it first queries its local cache; if the data is missing, it sends a request to other nodes. During the data transmission among nodes, the event-driven mechanism based on cache invalidation plays a crucial role. When the cache data on the edge server changes, it triggers cache invalidation notifications, informing all relevant nodes to update their cache data for consistency. Upon receiving the notification, the node updates its local cache and clears the obsolete data accordingly. The edge server can also dynamically optimize cache allocation and replacement strategies based on node usage and task workload, enhancing overall rendering performance. This combined approach of cache invalidation mechanism and leveraging edge servers enables efficient, reliable, and fast completion of distributed rendering computation in decentralized network environments. Moreover, it reduces data transmission and computational load among nodes by utilizing edge servers' storage and computing capabilities, thereby improving system responsiveness and overall performance. This novel solution provides important methodological and technological support for deploying and applying distributed rendering computation in decentralized network environments.

5.2. Shared scheduling for delay optimization

In CRCDnet, computing resources for rendering computing and rendered data are deployed on each mobile device, managed by a virtual ring constructed by the decentralized network's blockchain. Quickly locating the computing resources or rendered data that the client needs on the shared service ring is crucial to optimize the rendering response delay.

Due to the convenience and ubiquity of mobile browsers, the service environment of MWAR applications exhibits characteristics such as large scale, complexity, and user randomness. These characteristics pose challenges locating the required computational resources or rendered data in the shared service environment. To address these issues, we apply the Chord algorithm [25], based on Distributed Hash Table (DHT) and suitable for large-scale and complex service environments [26]. Firstly, the Chord algorithm ensures the even distribution of data throughout the system by dynamically re-allocating data and devices when devices join or leave. Each device is responsible for maintaining and processing data within its neighboring range, balancing load, and ensuring the equitable distribution of computational resources and data indexing, especially in many user scenarios. Secondly, the Chord algorithm exhibits good scalability, allowing for adding or removing devices without significantly impacting the overall system performance and availability. When new devices join, they only need to connect with a few devices to obtain neighboring index information rather than communicating with every device in the system. In the MWAR service environment, frequent users join, and offline events do not affect the distribution of the shared environment, thus providing robustness to the collaborative rendering computational environment. Lastly, the Chord algorithm provides redundancy and fault tolerance

through data replication. Each data item is typically stored in multiple devices, ensuring accessibility and recovery through backup devices in case of offline or failure.

Combined with the strong dependence on timing in rendering computing, we optimized the Chord algorithm using specific nonlinear search approaches:

- Each device maintains a Finger table, a routing table that stores all devices on the chain. The table's length is m (e.g., 160 bits in Chord Bit). The ith item of the table stores the $(n + 2i - 1) \mod 2^m$ successor ($1 \leq i \leq m$) of $Device_n$.
- Each device maintains a list of predecessors and successors, which allows for quickly locating the predecessor and successor and periodically checking their health status.
- The stored successor is increased proportionally by a multiple of 2. The modulus is necessary because the successor of the last device is the first few devices. For example, the next device of the largest device is defined as the first device.
- The resource key is stored on the following device: along the Chord ring, the first device where $hash(Device) \geq hash(Key)$ is the successor of this key.
- To find the device where the corresponding resource is located, that is, to find the successor of the key when searching for resources (if the search is performed on $Device_n$), follow the steps below.
- When searching for resources, the user attaches a value to the given key to display priority and determines priority based on the sequence of data frames that need to be displayed. Other user devices adjust the data processing queue according to task priority.

In the consensus mechanism of CRCDnet, we adopt the Practical Byzantine Fault Tolerance (PBFT) algorithm. PBFT is an extensively used algorithm in the consensus mechanism for distributed rendering computation in decentralized networks [27]. In decentralized networks, the consensus mechanism is crucial for ensuring the integrity and consistency of distributed rendering computation. PBFT addresses the Byzantine Generals Problem and provides an efficient and validated fault-tolerant algorithm for collaborative rendering computation in large-scale scenarios, considering mobile devices' frequent joining and leaving causing node failures. Firstly, PBFT enables the nodes in the network to reach a consensus on the order of rendering tasks, ensuring the consistency of the final rendered output. It utilizes a three-phase protocol involving message exchanges and computations among nodes to achieve consensus on the generated rendering frames. Secondly, PBFT ensures the fault tolerance of nodes even in situations where up to one-third of nodes could be malicious or faulty, making it suitable for decentralized networks where trust among nodes cannot be guaranteed. By leveraging cryptographic techniques and digital signatures, PBFT provides a defense against various attacks, such as data tampering, message forgery, and collusion among malicious nodes. It enhances the security and reliability of distributed rendering computation in the presence of potentially adversarial participants. Moreover, PBFT offers notable advantages in terms of performance. It exhibits lower communication overhead compared to other consensus algorithms [28], making it suitable for real-time rendering scenarios with high latency requirements. PBFT achieves this efficiency by reducing the required message exchanges and minimizing the computational load on each node during the consensus process.

5.3. Optimization of network initialization

The construction of a service sharing data in a decentralized network is a critical factor affecting service response delay. WebGL's rendering approach requires each device participating in collaborative rendering computing to download the complete 3D model file before performing rendering computing. Rendered data is then deployed to the virtual data-sharing ring in parallel. Multiple mobile devices must share the direct network bandwidth from the server to the base station. The following factors influence this process: (1) The amount of 3D model size impacts the service response delay. In rendering computing, the size of the 3D model affects the downlink delay of transmission from the remote server. Especially when multiple mobile devices participate in the initialization of the service sharing ring, the 3D model size influences the delay. (2) The time distribution of service requests in the service environment, where multiple mobile devices participate in initializing the service sharing ring, presents randomness and directly affects the calculation response delay.

The two points above can be summarized as follows: the number of mobile devices participating in the initialization of the service sharing data affects the response delay. More participating devices lead to lower rendering delay, but too many devices can cause downlink channel blocking and affect response delay. Fewer participating devices result in lower downlink communication delay but larger rendering delay of mobile devices. Therefore, optimizing service construction involves determining the number of network initialization devices in different situations.

To study the validity of this approach, we make the following assumptions. We assume that the shared bandwidth of the connection between the mobile device and the remote server is w, and the bandwidth is relatively constant during the service time. The computing resources provided by each mobile device to the rendering collaborative computing network are relatively constant and equal at f units. M devices request MWAR applications in time t, and the request time distribution probability follows a normal distribution. The unit time t is much less than the download time of participating network initialization devices. The granularity of the rendering task partition is small so that we can ignore its effect in practice.

We assume that the 0-th device initiates the service at time 0, and the initialization time of the sth device is $\sum_{i=1}^{s} \Delta t_i$. Thus, when n devices participate in establishing the service sharing ring in the decentralized network, the total download delay $t(n)$ from the remote server can be calculated using Eq. (1).

$$t_{loading}(n) = 2 \times \sum_{i=1}^{n-1} \Delta t_i + \frac{p - \sum_{i=0}^{n-1} \frac{k \times \Delta t_i}{i}}{\frac{k}{n}}. \tag{1}$$

In Eq. (1), k describes the complexity of rendering computation, which is expressed as the proportional relationship between rendering computation latency and the volume size of the original 3D model, as well as the CPU cycle of the mobile device. The p represents the rendering calculation expansion coefficient of the 3D model, which is expressed as the proportional relationship between the rendered data volume and the original 3D model volume. Moreover, assuming that the delay of a mobile device to complete all rendering computing tasks is T, we can represent the delay of n devices in implementing the rendering computing task as $\frac{T}{n}$. The delay of the nth device joining the service sharing ring to perform the rendering computing task is expressed in Eq. (2):

$$t_{render}(n) = \frac{T - \sum_{i=1}^{n-1} i \Delta t_i}{n} \tag{2}$$

Therefore, when n devices participate in the initialization of the service sharing ring, the total construction delay is expressed in Eq. (3):

$$t(n) = \sum_{i=1}^{n-1} \left(2 - \frac{i}{n} - \frac{n}{i}\right) \Delta t_i + n\frac{p}{k} + \frac{T}{n} \tag{3}$$

CRCDnet determines the optimal initialization construction strategy for the service sharing ring by finding the minimum value of $t(n)$. To simplify the initialization calculation strategy for joining the ring, we examine the relationship between $t(n+1)$ and $t(n)$; if $t(n+1)$ is less than $t(n)$, new mobile devices can join the initialization process. Otherwise,

Algorithm 1: service sharing ring initialization construction strategy

Input: T, p, k, n, $ring_{state}$
/* $ring_{state}$ is the state of service sharing ring initialization construction completion. */
Output: $bool\ State_{part}$
/* $State_{part}$ is the state of a new mobile device participating in the construction of service-shared ring initialization. */
$\Delta t_{limit}(n) = (n^2+n) \times (\sum_{i=1}^{n-1}[(\frac{i}{n^2+n} - \frac{1}{i}]\Delta t_i + \frac{p}{k} - \frac{T}{n^2+n})$ **While**(new mobile device join the network)
{
if ($ring_{state}$ =*true*) **then**
/* If service sharing ring initialization construction complete, return *true*;else,return *false*. */
$State_{part}$=false;
break;
else
if ($\Delta t_{q-n} > \Delta t_{limit}$) **then**
$Device_{new} \rightarrow Ring_{service}$;
$State_{part}$=true;
/* the new mobile device participates in the initialization of the service sharing ring construction */
$\Delta t_{limit}(n) = \Delta t_{limit}(n+1)$;
else
next;
/* wait for the new mobile device */
end
end
}

$t(n)$ is the optimal state. We use $\Delta t(n) = t(n+1) - t(n)$ to represent the difference between these two states, as expressed in Eq. (4):

$$\Delta t(n) = \sum_{i=1}^{n-1}[\frac{i \times \Delta t_i}{n^2+n} - \frac{\Delta t_i}{i}] - \frac{\Delta t_n}{n^2+n} + \frac{p}{k} - \frac{T}{n^2+n} \tag{4}$$

In this equation, T, p, and k remain constant when the service environment is determined, and the initialization strategy of the CRCDnet service sharing ring is mainly affected by Δt_i and n. The relationship between Δt_i and n that allows new mobile devices to join the initialization of the service sharing ring is expressed in Eq. (5):

$$\Delta t(n) > (n^2+n) \times (\sum_{i=1}^{n-1}[\frac{i \times \Delta t_i}{n^2+n} - \frac{\Delta t_i}{i}] + \frac{p}{k}) - T \tag{5}$$

Therefore, to determine the initialization strategy of the CRCDnet service sharing ring, the algorithm performs the following steps: When the initial construction strategy of the service shared ring changes (n changes), CRCDnet calculates the value of $\Delta t_{limit}(n) = (n^2+n) \times (\sum_{i=1}^{n-1}[\frac{i}{n^2+n} - \frac{1}{i}]\Delta t_i + \frac{p}{k} - \frac{T}{n^2+n})$. When a new mobile device joins the network, CRCDnet determines the relationship between Δt_{q-n} (the time span between the current device request (q-thdevice) and the nth device request) and Δt_{limit}. If $\Delta t_{q-n} > \Delta t_{limit}$, the current device participates in the initialization of the service sharing ring construction. Otherwise, CRCDnet waits for the next mobile device to join the network. The new mobile device will not join the service sharing ring initialization construction and perform computing until the new device's Δt_{q-n} is greater than Δt_{limit} and the service sharing ring initialization construction has not been completed.

The algorithm for the CRCDnet service sharing ring initialization construction strategy is presented as Algorithm 1:

6. Performance evaluation

In this section, we present our evaluation with three objectives. Firstly, we aim to evaluate the optimization efficiency of CRCDnet on mobile devices' average delay in the user aggregation scenarios by comparing it with traditional client file-based rendering and remote cloud-based rendering to highlight its advantages. Secondly, we aim to investigate the impact of blockchain index-based data exchange approaches on latency in collaborative rendering networks by comparing data exchange latency under different data exchange capacities and mobile device numbers. Finally, we aim to validate the effectiveness of the collaborative rendering network construction approach by comparing the collaborative intelligent optimization algorithm with the traditional construction approach to demonstrate the effectiveness of the collaborative algorithm under the actual service environment.

6.1. Experiment setup

We have designed an experimental service environment that consists of multiple dockers and remote cloud servers. In this setup, Docker is used to simulate mobile devices in user aggregation scenarios, and each Docker is set as a single CPU with a main frequency of 2.5 GHz and 2 GB of memory. We deploy a common server with a six-core Intel processor of 2.9 GHz and 16 GB RAM near the docker containers. To ensure stable network conditions, we use Wonder Shaper. This tool allows us to limit the bandwidth of network adapters and control the network conditions between the remote cloud server and mobile devices.

It should be noted here that Docker guarantees validation effectiveness without GPU acceleration. Firstly, the relatively new WebGPUs currently cannot provide support for operation on general mobile devices. The current popular GPU acceleration method for mobile web browsers uses a dedicated Javascript engine to retrieve GPU resources from the bottom layer of the browser for parallel computing. However, this method is limited by the parallel computing power of mobile devices and the use of browser resources, and there is no significant performance improvement. Secondly, the popularity of GPUs in mobile devices is relatively low, and many outdated mobile devices in the market lack GPU support to provide computing services. Moreover, the performance of mobile device GPUs does not match that of PC GPUs, and they cannot provide powerful computing resource support for presenting computing services in an immersive service environment like PC GPUs. Therefore, in our verification process, we simulated mobile devices using Docker containers to execute CRCDnet distributed rendering computing services without GPU computing resources. In addition, when comparing the initialization delays of different methods of collaborative rendering computation, we used the same client environment (with the same CPU, cache, and Docker, without GPU) to ensure the persuasiveness of our validation. Although devices with GPUs exhibit a certain proportion of latency changes compared to simulated mobile devices without GPUs, the overall trend remains consistent in both environments.

To construct our experimental service application, this evaluation selected two models for effect validation (as shown in Table 1). The first 3D model chosen was of moderate size, approximately 6.87 MB, to represent loading the same original 3D model on different application terminals and performing partially (approximately 30%) differentiated distributed rendering based on personalized data (e.g., facial reconstruction). The 3D model was fully rendered in each Web browser and presented to the user for the corresponding service interaction. The second 3D model selected was a high-precision model with a size of 43.4 MB. For service requirements, each browser only needed to display an image of a certain angle of this 3D model. When multiple browsers requested the same 3D model, the viewing angles were evenly distributed in space. Additionally, each browser had a specified viewing field range of 0–120 degrees. Therefore, while different mobile Web

Table 1
The properties of 3D Model #1 and #2.

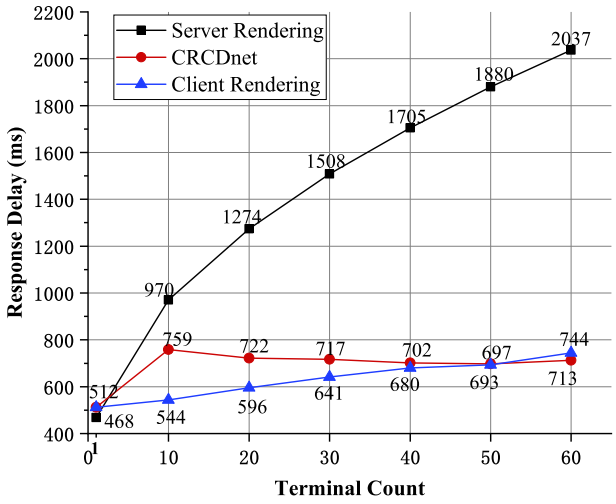
Properties	3D model #1	3D model #2
Volume	6.87 MB	43.4 MB
Triangle	80,328	664,665
Point	45,589	339,794

browsers loaded the same 3D model, each browser only rendered a portion of the 3D model due to different viewing angles. Since the recognition, tracking, and other computational processes in MWAR are not related to our study, we focused solely on the rendering process delay of the 3D model.

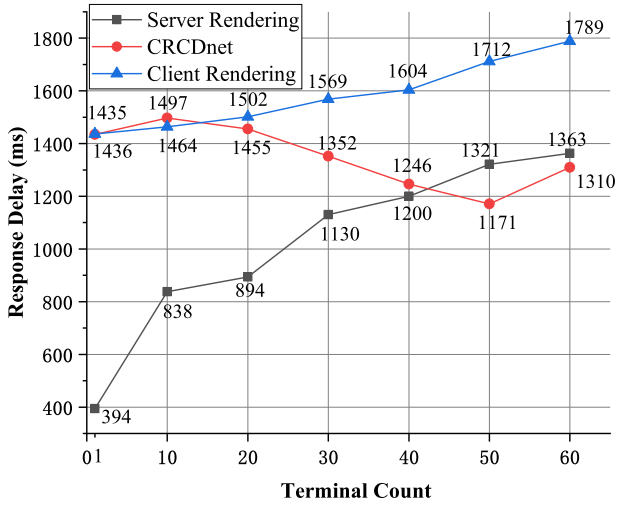
6.2. Effectiveness of initialize rendering

To determine under which circumstances CRCDnet outperforms mobile-based and remote cloud-based rendering, we varied the number of mobile devices in a user aggregation scenario. Each mobile device acted as an application terminal in this evaluation, and its mobile browser sent MWAR service requests to remote servers simultaneously. Here, we use 3D Model #1 and 3D Model #2 referred in Section 6.1 to evaluate. We measured each Web browser of the service environment's average service response delay in MWAR rendering computation. This process involves the overall latency of multiple procedures, including node execution of rendering computation tasks, communication of rendered data, browser aggregation, and loading during the initialization of MWAR applications. Moreover, the rendering data for 3D models #1 and #2 and the rendered data were optimized using appropriate approaches [29] during transmission from remote servers or mobile devices to browsers. This approach allows mobile device browsers to operate without waiting for the full frame sequence data of the model to load in the cache, thereby reducing initialization latency. In cloud-based, client-based, and CRCDnet approaches, the computing nodes on these devices both utilize the WebGL-based rendering approach. In the approach, frames containing fully or partially rendered data of 3D models are stored in the cache, and the corresponding frame data is pushed onto the browser's canvas element. Using Web browsers, we simulated service requests sent simultaneously by mobile devices in the user aggregation scenarios and presented the average delay in Fig. 6.

Fig. 6(a) shows that as the few mobile devices, remote cloud rendering exhibited better performance. This result indicates that the core network's transmission delay primarily affects cloud rendering availability. Cloud rendering can leverage sufficient computing resources to provide low-latency rendering offloading services to application terminals without core network transmission limitations. However, as mobile devices increased, cloud rendering performed poorly compared to other approaches. The performance of CRCDnet and the traditional client-side file approach demonstrated different trends. The average delay of the traditional client-side file approach increased with the mobile devices because many mobile devices obstructed the core network and access network resources when requesting original 3D models, even with edge storage and CDN solutions. On the other hand, the average delay of the CRCDnet approach decreased with an increase in mobile devices. However, many requests for the original data caused congestion in the core and access networks, increasing the number of mobile devices participating in the rendering and reducing the client-side rendering delay. In contrast, in the traditional client-based approach, this delay remained fixed. Collaborative computation and data sharing among mobile devices reduce the response delay in the CRCDnet approach. In Fig. 6(b), the trends observed with increasing devices differ. When the count of devices is small, rendering based on cloud servers exhibits better delays. This is because, for 3D model #2, the partial rendering strategy effectively reduces the transmitted data pressure on the downstream channel. However, as the Web browsers that request



(a) Latency with 3D model #1



(b) Latency with 3D model #2

Fig. 6. Latency of initialize rendering.

MWAR in the service environment increase, the CRCDnet approach outperforms rendering based on remote servers in terms of delay. This is because as the number of mobile devices increases, despite optimizing redundant data during downstream transmission, the concurrency of multiple device requests and the large data volume of rendered data result in high data transfer and response delays for rendering services. For browser-based rendering methods, the large size of the original 3D model imposes significant loading and rendering pressure on mobile Web browsers. Based on these phenomena in Fig. 6(a) and Fig. 6(b), we can conclude that CRCDnet is more suitable for user aggregation scenarios. The decentralized network's distributed rendering approach can effectively resolve the computational and rendering pressures of remote rendering on core computing nodes and networks. However, it should be noted that the mobile devices that decentralized networks can support will be limited by the complexity and scalability of the blockchain running on them, which will be discussed in Section 6.5.

Additionally, we compared the delay performance of the 3D models #1 and #2 for different numbers of mobile devices. 3D model #2 exhibited less delay performance when using a cloud-based rendering method than 3D model #1. This is because the partial rendering strategy used for the 3D model #2 resulted in smaller rendered data

Table 2
The data exchange delay with different Web Browser (ms).

Volume	Approach	$Device_1$	$Device_2$	$Device_3$	$Device_4$	$Device_5$	$Device_6$	$Device_7$	$Device_8$	$Device_9$	$Device_{10}$	Average
100 kb	Blockchain	1786	1820	4209	1551	4833	1431	1877	2039	4890	4452	2888.8
	CRCDnet	189	167	175	168	152	144	151	166	158	170	164
1000 kb	Blockchain	7285	3658	3931	3685	3845	3797	5551	4484	6842	4911	4798.9
	CRCDnet	352	388	367	348	350	332	378	340	388	364	360.7
5000 kb	Blockchain	14 365	25 787	16 542	18 454	31 254	15 474	16 522	17 789	19 122	16 144	19 145.3
	CRCDnet	489	446	512	488	431	425	440	460	478	484	465.3

than the 3D model #1, allowing reduced data transfer pressure on the downstream link. In the CRCDnet-based approach, the delay performance of 3D model #1 was less than that of 3D model #2. This is mainly due to the larger size of the original 3D model file for 3D model #2, leading to a longer delivery time from the server to the Web browser. Furthermore, although 3D model #1 had a higher overall rendering load than 3D model #2, 3D model #1 performed better in terms of delay when loading the original file and rendering computations in WebGL. This is because the decentralized network's distributed rendering approach effectively addresses the delay caused by the high computational complexity of rendering. The loading of larger original 3D model files is limited by insufficient browser caching, resulting in longer delays in the rendering process. Therefore, it can be concluded that CRCDnet is more suitable for service scenarios with smaller 3D model file sizes and higher rendering complexity.

6.3. Effectiveness of exchange rendered data

Additionally, we investigated the data exchange efficiency in decentralized collaborative drawing computation networks. Data are typically encrypted and exchanged through blockchain in decentralized networks to ensure users' privacy and security. However, the complexity of blockchain can affect the responsiveness of distributed rendering in decentralized networks. We proposed an index-based decentralized network data exchange approach. In this subsection, we compare the efficiency of traditional decentralized network data exchange and the CRCDnet data exchange approach from two perspectives. Firstly, we verified the feasibility of the index-based decentralized network data exchange approach by changing the capacity of blockchain-based data exchanged. In this section of the evaluation, to make the evaluation more comprehensive, we separated the data exchange process from the rendering calculation and verified the effectiveness of the CRCDnet data exchange approach by exchanging fixed-capacity data between various mobile devices. The results are shown in Table 2. Secondly, we simulated the performance efficiency of the CRCDnet data exchange approach in service environments with different user capacities by changing the number of application terminals and determining the application scenarios of the CRCDnet data exchange method. The results are shown in Fig. 7.

Table 2 shows that the blockchain-based data volume exchange latency varies significantly with increased data exchanged between clients. In addition to network resource limitations, device computational resources are important factors influencing the latency of rendering data exchange. In blockchain-based data exchange, each device needs to encrypt and process the rendering data, which may result in high latency when encrypting a large amount of data on client devices with low computational resources. CRCDnet reduces the data exchange volume on the blockchain, reducing the device's demand for computational resources. Therefore, CRCDnet demonstrates better adaptability to various environments than traditional decentralized network data exchange methods. Furthermore, in terms of data exchange latency performance, blockchain-based data exchange shows poor latency stability, as shown in Table 2. Some devices experience significant increases in rendering latency compared to others. This is due to computational concurrency and network congestion,

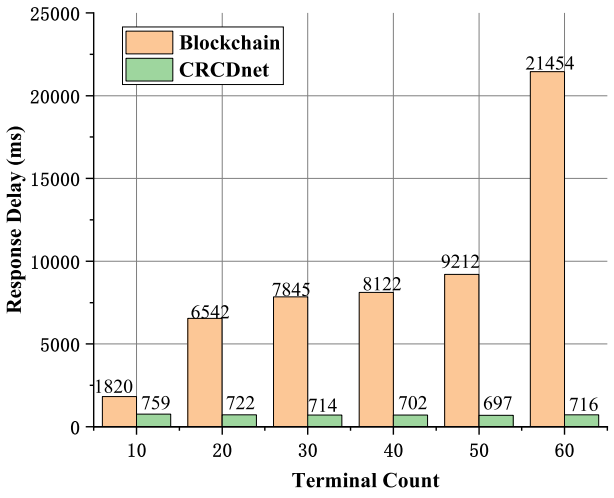
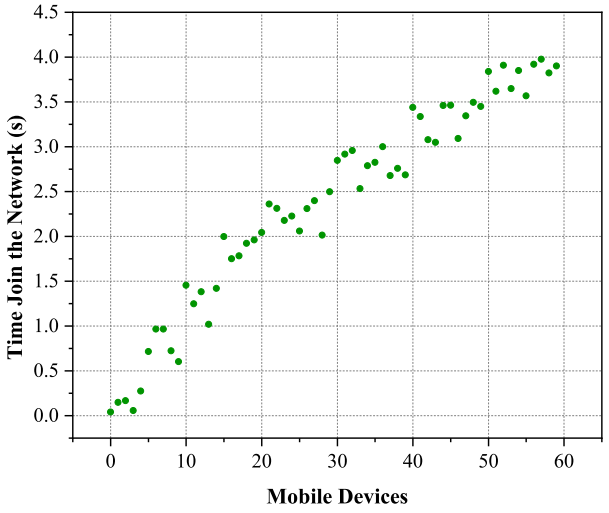


Fig. 7. The latency performance with different count device by blockchain and CRCDnet.

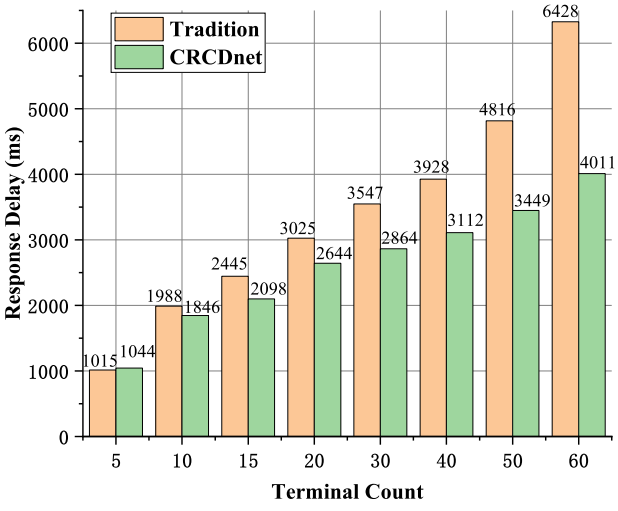
which cause computational congestion on certain devices under limited computational resources. On the other hand, the CRCDnet data exchange method, with a smaller exchange data volume based on blockchain, exhibits more stable latency performance. This ensures that each user has a better user experience in MWAR. Furthermore, Fig. 7 shows the latency changes caused by the increase in mobile devices for both blockchain-based data exchange and the CRCDnet method. Increasing application terminals brings about complex computational consumption and higher data exchange complexity for devices. The performance of the CRCDnet approach is better than that of blockchain-based data exchange. The CRCDnet approach only involves transaction computation for smaller sensitive information data on the blockchain, reducing the amount of data exchanged and improving the efficiency of distributed rendering. Based on these observations, we can conclude that the CRCDnet data exchange approach suits service environments with larger data exchange capacities, especially for applications like MWAR, where the rendered data has a significant volume. Furthermore, in scenarios with increasing users, the CRCDnet data exchange approach performs better in latency than blockchain-based data exchange methods. In other words, the CRCDnet data exchange approach is more suitable for service environments with more users.

6.4. Cooperative service network

To further evaluate the efficacy of the CRCDnet's service sharing ring construction algorithm, we conducted verification using the 3D model #1 referred to in Section 6.1. We varied the time distribution of mobile devices joining the network to investigate the impact of Eq (5) on constructing a service sharing ring in the decentralized network. The time distribution when mobile devices join the network is shown in Fig. 8(a). We record the rendered time of the last mobile device to join the network to evaluate the efficiency of the service sharing ring construction, and the measurement data are presented in Fig. 8(b):



(a) Time join the network



(b) Average delay of devices

Fig. 8. Efficiency of self-organizing networks.

As shown in Fig. 8(b), when mobile devices join the network initialization satisfies Eq. (5), the rendered time of the last mobile device to join the network is lower than in other cases. These results suggest that the CRCDnet service sharing ring construction algorithm performs better regarding service response delay.

6.5. Discussion

The preceding Subsections compare the CRCDnet and cloud rendering computation approaches, browser rendering computation approaches, and data interaction approaches based on blockchain in a decentralized network environment. However, there are a few aspects that require clarification to enhance the effectiveness of this work.

First, as referred to in Section 4.1, in this work, the decentralized rendering computation on application terminals and mobile devices is performed using a primitive remote rendering offloading approach based on the WebGL engine. This primitive approach, compared to the novel approaches [30–32], is considered more rudimentary and exhibits poorer performance in remote rendering latency. However, this approach clarifies the remote rendering process, especially when

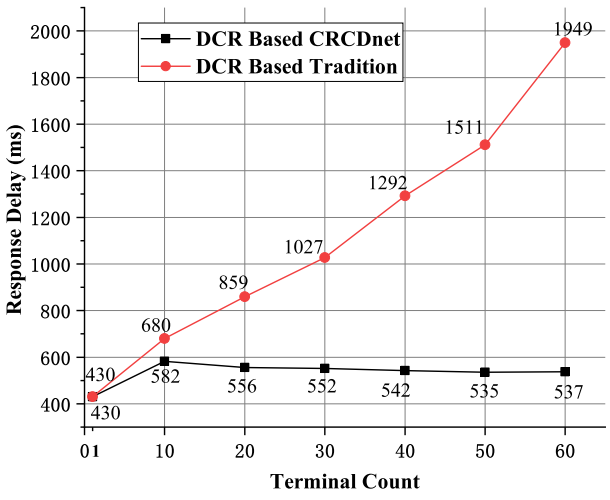


Fig. 9. The latency performance on other remote rendering approach based CRCDnet.

incorporating lighting, animation, and other physics-based phenomena in the 3D model scenes. The Web browser can execute the corresponding physics calculations through the underlying WebGL interface and achieve frame multiplexing between multiple devices in media fragments. Additionally, we did not employ rendering computation acceleration approaches such as WebGPU in our experiments. Due to the diversity of service environments, outdated mobile devices may not support newer rendering acceleration approaches.

Furthermore, certain novel remote rendering approaches can leverage the decentralized network organization proposed in this study and the data-sharing-driven approach to optimize decentralized distributed rendering computation. In our experiments, we employed the DCR approaches referred to in the literature [7] for decentralized rendering computation. In this work, we compared the delay performance of distributed rendering computing services using DCR alone and using DCR and CRCDnet collaboratively to validate the effectiveness of the CRCDnet approach. We will verify the latency performance of the 3D model #1 referred to in Section 6.1 under different numbers of mobile devices. The evaluation results are shown in Fig. 9.

As shown in Fig. 9, the distributed collaborative rendering computing approach based on CRCDnet exhibits lower response latency compared to the traditional distributed collaborative rendering computing approach. In particular, with some novel remote distributed client rendering computing methods (such as DCR [7]), CRCDnet can better organize multiple mobile devices in the user aggregation scene to provide rendering computing services. This is attributed to adopting a collaboration-driven algorithm based on a data-sharing pool in the CRCDnet approach. This algorithm collaboratively maximizes the computing capabilities of multiple mobile devices and constructs a shared pool to effectively drive rendering computation services for mobile devices, reducing network congestion issues caused by multiple mobile devices. Consequently, it can be concluded that CRCDnet, in collaboration with other channel-based frame or media segment reusability methods, provides rendered computational services for immersive applications, offering lower latency rendering computational services for mobile devices, especially in user aggregation scenarios with multiple devices for MWAR environments. However, it should be noted that during the collaboration process, the efficiency of frame or media segment reusability directly impacts the efficiency of rendering computation services, as discussed in Section 2.3. Therefore, in collaboration with CRCDnet rendering computation services, it is advisable to adopt a rendering method based on data formats with low-latency reusability (such as JSON) to meet multiple users' real-time rendering computation demands.

Secondly, the count of mobile devices will impact the performance of CRCDnet. As demonstrated in Section 6.3, we constructed a collaborative decentralized rendering computation network with 60 mobile devices as the maximum number of nodes. However, based on Fig. 6(a), we further increase the number of mobile devices using the CRCDnet approach. When the mobile device reaches 150, the delay is 693 ms. Although the delay performance is smaller than that of 60 mobile devices, the optimization efficiency is significantly lower than that of the decentralized network formed by the 60 mobile devices. It is a delay of 867 ms for the number of mobile devices reaches 300, the latency performance is already higher than that of the decentralized network composed of 60 and 150 mobile devices. As the number of devices in decentralized networks continues to increase, the latency of collaborative rendering computation will increase. It is due to two reasons. Firstly, the computational complexity of blockchain-based data exchange introduces additional computation latency for mobile devices, especially weaker computational mobile devices. As the count of devices in the network increases, the blockchain approach we have chosen imposes significant computation pressure on application terminals and mobile devices. Second, each mobile device in the decentralized collaborative rendering computation needs to receive the rendered binary data from multiple devices and render it in the container of the mobile browser. The browser needs to perform serialized calculations on the data from multiple devices to support smooth interaction with the rendered 3D objects. When the count of mobile devices is large, serialized calculations will result in significant latency, affecting the efficiency of overall rendering offloading. In our experiment, virtual machines have relatively limited computing resources and cannot support overly complex blockchain networks. To address these challenges, we can adopt a decentralized network grouping approach that draws inspiration from multi-layer federated learning (M-FL). Taking into account factors such as the participation time and available resources of mobile devices, we can establish a scientifically layered collaborative rendering computation network architecture [33,34]. By leveraging collaborative distributed rendering computation within and between groups in the layered structure, we can effectively address the excessive consumption of mobile device computational resources in a decentralized network. Additionally, with the organization of rendered image data by specific computing nodes (edge servers) in the decentralized network under the layered structure, we can alleviate the excessive computation latency generated by serialized calculations of multi-source binary data on the terminals. However, as these concepts draw on relevant approaches from the existing literature, further discussion of them is beyond the scope of this work. It will also be the focus of our future work.

7. Related work

In CRCDnet, collaborative rendering approaches mainly involve two core technologies: distributed remote rendering and decentralized network. We will discuss the relevant literature on these two aspects and the current challenges.

Distributed rendering computing is currently the main means to solve the computational and communication pressure problems of core devices and networks in rendering computing. Distributed computer rendering has been explored by A Joshi et al. who presented initial findings on this topic [35]. They discussed the potential benefits and challenges of distributing rendering tasks across multiple devices. Distribution scenarios, requirements, and synchronization techniques were classified by Hoppen et al. providing a comprehensive understanding of distributed rendering [36]. In the context of mobile devices, C. Glez-Morcillo, etc., presents a novel approach based on grid approach and the P2P model for distributed rendering [37]. In Distributed rendering computing, computing resource distribution in networks is challenging and requires addressing gaps in existing network setups. Alexander Clemm et al. identified these challenges, highlighted areas that require improvement in current networks, and proposed an improvement approach for Distributed rendering computing [38]. Dan Liu et al. introduced load-balancing strategies for distributed rendering, with the aim of effectively distributing tasks and optimizing resource utilization [39]. Constantin Nandra et al. experimented with a general-purpose distributed processing solution for rendering 3D object scenes [40]. Dante Abate et al. implemented a platform for efficient multi-user online sharing of high-quality 3D textured models, leveraging a remote HPC infrastructure [41]. In the rendering approach in distributed computing, Huan Cheng et al. [42] proposed visualization algorithms to quickly display massive data. Hao Fang et al. introduced an image-based distributed rendering architecture that enables multiple clients sharing a space to receive rendering results [43]. These works contribute to the development and improvement of distributed rendering techniques. However, most existing approaches are based on the assumption that the surrounding computing resources are idle and sufficient without fully considering the computational and network congestion caused by many concurrent service requests from mobile devices in user aggregation scenarios. This computation and network blocking will cause more latency in delivering rendered data. Meanwhile, the random addition of mobile devices belonging to different users in user aggregation scenarios can also bring uncertainty to distributed collaborative computing. Ineffective scheduling of changes in computing resources will increase collaborative rendering latency, leading to a negative user interaction experience.

In decentralized networks, the data exchange mechanisms enable efficient service provision. Mobile devices in the decentralized network are connected through blockchain technology, and existing research focuses on the security aspects of data exchange between blockchain nodes. For instance, literature [44] proposed a data exchange approach based on smart contracts to facilitate secure information exchange. Similarly, the literature [45] proposed a Distributed Denial of Service (DDoS) data exchange platform for the Internet of Things (IoT) environment, leveraging blockchain technology to overcome trust and fairness issues. However, existing research has primarily focused on ensuring high security and non-real-time data exchange, neglecting the efficiency of data exchange. Some studies have explored real-time performance and switching efficiency to address this limitation. For example, the literature [46] proposed an approach of offloading block transmission transactions to RDMA NIC, which can improve block broadcast speed and reduce block synchronization delay. Additionally, literature [47] proposed a lightweight blockchain transaction process modeling to enhance the applicability of blockchain in scenarios with weak computing resources. Nonetheless, these studies are inadequate for rendering computing offloading in MWAR due to the large data capacity and high-efficiency data exchange requirements between mobile devices. Moreover, the real-time variability of MWAR can cause traditional approaches to lose data, affecting the integrity of data exchange between devices. Therefore, novel approaches are needed to address these challenges.

8. Conclusions and future work

This study presents CRCDnet, a decentralized and collaborative rendering computing offloading network architecture for MWAR applications in the user aggregation scenario. CRCDnet aims to reduce bandwidth usage and lower latency by introducing a decentralized network to render computing offloading services between mobile devices. We propose a data request and computing offloading scheduling approach for the collaborative rendering computing network to optimize the rendering computation delay. Our results show that CRCDnet provides more private and stable rendering computing offloading devices and performs better in rendering latency and application expansibility in practical Web AR applications. These findings motivate us to expand CRCDnet to more complex service environments in future research. To provide smooth and efficient rendering computing data services for

MWAR applications, we need to further optimize the data exchange approach and data loading mechanism between mobile devices. Additionally, conducting simulations in different systems can provide more insightful knowledge for future development.

CRediT authorship contribution statement

Huabing Zhang: Writing – review & editing, Formal analysis, Data curation. **Liang Li:** Visualization, Software, Methodology, Conceptualization. **Qiong Lu:** Writing – original draft, Funding acquisition. **Yi Yue:** Project administration, Data curation. **Yakun Huang:** Funding acquisition, Conceptualization. **Schahram Dustdar:** Writing – review & editing.

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.

Data availability

The authors are unable or have chosen not to specify which data has been used.

Acknowledgments

This work is supported in part by the National Natural Science Foundation of China under Grant 62202065, in part by the Zhejiang provincial natural science foundation, China under Grant LTGG23F020001, in part by the Key Lab of Film and TV Media Technology of Zhejiang Province, China under Grant 2020E10015, in part by 2023–2025 Young Elite Scientists Sponsorship Program of Beijing Association for Science and Technology, China.

References

[1] I. Coma-Tatay, S. Casas-Yrurzum, P. Casanova-Salas, M. Fernandez-Marin, FI-AR learning: a web-based platform for augmented reality educational content, Multimedia Tools Appl. 78 (5) (2019) 6093–6118.

[2] Y. Huang, X. Qiao, P. Ren, L. Liu, C. Pu, J. Chen, A lightweight collaborative recognition system with binary convolutional neural network for mobile web augmented reality, in: 2019 IEEE 39th International Conference on Distributed Computing Systems, ICDCS, 2019, pp. 1497–1506, http://dx.doi.org/10.1109/ICDCS.2019.00149.

[3] Z. Weini, L. Yongquan, G. Pengdong, Q. Chu, Q. Quan, A new software architecture for ultra-large-scale rendering cloud, in: 2012 11th International Symposium on Distributed Computing and Applications To Business, Engineering & Science, 2012, pp. 196–199, http://dx.doi.org/10.1109/DCABES.2012.10.

[4] J. Doellner, B. Hagedorn, J. Klimke, Server-based rendering of large 3D scenes for mobile devices using G-buffer cube maps, in: Proceedings of the 17th International Conference on 3D Web Technology, in: Web3D '12, Association for Computing Machinery, New York, NY, USA, 2012, pp. 97–100, http://dx.doi.org/10.1145/2338714.2338729.

[5] A. Evans, M. Romeo, A. Bahrehmand, J. Agenjo, J. Blat, 3D graphics on the web: A survey, Comput. Graph. (2014).

[6] E. Gobbetti, F. Marton, M.B. Rodriguez, F. Ganovelli, M. Di Benedetto, Adaptive quad patches: An adaptive regular structure for web distribution and adaptive rendering of 3D models, in: Proceedings of the 17th International Conference on 3D Web Technology, in: Web3D '12, Association for Computing Machinery, New York, NY, USA, 2012, pp. 9–16, http://dx.doi.org/10.1145/2338714.2338716.

[7] L. Li, Y. Huang, X. Qiao, Y. Meng, D. Yu, P. Ren, S. Dustdar, Towards distributed collaborative rendering service for immersive mobile web, IEEE Netw. early access (2023) 1–10.

[8] K. Salah, M.H.U. Rehman, N. Nizamuddin, A. Al-Fuqaha, Blockchain for AI: Review and open research challenges, IEEE Access 7 (2019) 10127–10149.

[9] J. Chen, P. Chen, X. Niu, Z. Wu, L. Xiong, C. Shi, Task offloading in hybrid-decision-based multi-cloud computing network: a cooperative multi-agent deep reinforcement learning, J. Cloud Comput. 11 (1) (2022) 1–17.

[10] X. Qiao, Y. Huang, S. Dustdar, J. Chen, 6G vision: An AI-driven decentralized network and service architecture, IEEE Internet Comput. 24 (4) (2020) 33–40.

[11] Q. Huang, M. Dong, Analysis and design for the decentralized network sharing of static resources, in: Proceedings of the 2020 4th International Conference on Electronic Information Technology and Computer Engineering, EITCE '20, Association for Computing Machinery, New York, NY, USA, 2021, pp. 885–890, http://dx.doi.org/10.1145/3443467.3443873.

[12] S. Liu, F. Yang, D. Li, M. Bagnulo, B. Liu, X. Huang, The trusted and decentralized network resource management, in: 2020 29th International Conference on Computer Communications and Networks, ICCCN, 2020, pp. 1–7, http://dx.doi.org/10.1109/ICCCN49398.2020.9209590.

[13] A. Zwitter, J. Hazenberg, Decentralized network governance: Blockchain technology and the future of regulation, Front. Blockchain 3 (2020) 12.

[14] M. Wang, Z. Yan, A survey on security in D2D communications, Mob. Netw. Appl. 22 (2) (2017-04-01) 195–208, http://dx.doi.org/10.1007/s11036-016-0741-5.

[15] K. Ali, H.X. Nguyen, P. Shah, Q.-T. Vien, N. Bhuvanasundaram, Architecture for public safety network using D2D communication, in: 2016 IEEE Wireless Communications and Networking Conference Workshops, WCNCW, 2016, pp. 206–211, http://dx.doi.org/10.1109/WCNCW.2016.7552700.

[16] J. Liu, B.B. Li, B. Lan, J.R. Chang, A resource reuse scheme of D2D communication underlaying LTE network with intercell interference, Commun. Netw. 5 (3C) (2013) 187–193.

[17] E. Yaacoub, H. Ghazzai, M.-S. Alouini, A. Abu-Dayya, Achieving energy efficiency in LTE with joint D2D communications and green networking techniques, in: 2013 9th International Wireless Communications and Mobile Computing Conference, IWCMC, 2013, pp. 270–275, http://dx.doi.org/10.1109/IWCMC.2013.6583571.

[18] X. Yao, J. Chen, T. He, J. Yang, B. Li, A scalable mixed reality platform for remote collaborative lego design, in: IEEE INFOCOM 2022 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2022, pp. 1–2, http://dx.doi.org/10.1109/INFOCOMWKSHPS54753.2022.9798010.

[19] Y. Jiang, J. Kang, D. Niyato, X. Ge, Z. Xiong, C. Miao, X. Shen, Reliable distributed computing for metaverse: A hierarchical game-theoretic approach, IEEE Trans. Veh. Technol. 72 (1) (2023) 1084–1100, http://dx.doi.org/10.1109/TVT.2022.3204839.

[20] C. Wu, B. Yang, W. Zhu, Y. Zhang, Toward high mobile GPU performance through collaborative workload offloading, IEEE Trans. Parallel Distrib. Syst. 29 (2) (2018) 435–449, http://dx.doi.org/10.1109/TPDS.2017.2754482.

[21] L. Zhang, X. Wu, F. Wang, A. Sun, L. Cui, J. Liu, Edge-based video stream generation for multi-party mobile augmented reality, IEEE Trans. Mob. Comput. (2022) 1–15, http://dx.doi.org/10.1109/TMC.2022.3232543.

[22] M. Sun, X. Xu, X. Tao, P. Zhang, Large-scale user-assisted multi-task online offloading for latency reduction in D2D-enabled heterogeneous networks, IEEE Trans. Netw. Sci. Eng. 7 (4) (2020) 2456–2467, http://dx.doi.org/10.1109/TNSE.2020.2979511.

[23] F. Wei, S. Chen, W. Zou, A greedy algorithm for task offloading in mobile edge computing system, China Commun. 15 (11) (2018) 149–157, http://dx.doi.org/10.1109/CC.2018.8543056.

[24] Q. Zheng, T. Yang, Y. Kan, X. Tan, J. Yang, X. Jiang, On the analysis of cache invalidation with LRU replacement, IEEE Trans. Parallel Distrib. Syst. 33 (3) (2022) 654–666, http://dx.doi.org/10.1109/TPDS.2021.3098459.

[25] B. Leong, B. Liskov, E.D. Demaine, EpiChord: Parallelizing the chord lookup algorithm with reactive routing state management, Comput. Commun. 29 (9) (2006) p.1243–1259.

[26] W. Jiang, C. Xu, M. Huang, J. Lai, S. Xu, Improved chord algorithm in mobile peer-to-peer network, in: Advanced Intelligence and Awareness Internet (AIAI 2011), 2011 International Conference on, 2011, pp. 244–251.

[27] W. Li, C. Feng, L. Zhang, H. Xu, B. Cao, M.A. Imran, A scalable multi-layer PBFT consensus for blockchain, IEEE Trans. Parallel Distrib. Syst. 32 (5) (2021) 1146–1160, http://dx.doi.org/10.1109/TPDS.2020.3042392.

[28] J. Ye, J. Liang, X. Li, Q. Chen, A distributed energy trading framework with secure and effective consensus protocol, in: 2022 IEEE 42nd International Conference on Distributed Computing Systems Workshops, ICDCSW, 2022, pp. 1–6, http://dx.doi.org/10.1109/ICDCSW56584.2022.00010.

[29] Z. Cui, Y. Zhao, C. Li, Y. Song, W. Li, Content-aware load balancing in CDN network, in: 2020 IEEE 6th International Conference on Computer and Communications, ICCC, 2020, pp. 88–93, http://dx.doi.org/10.1109/ICCC51575.2020.9345240.

[30] Y. Zhang, Z. Li, S. Xu, C. Li, J. Yang, X. Tong, B. Guo, RemoteTouch: Enhancing immersive 3D video communication with hand touch, in: 2023 IEEE Conference Virtual Reality and 3D User Interfaces, VR, 2023, pp. 1–10, http://dx.doi.org/10.1109/VR55154.2023.00016.

[31] J. Venerella, T. Franklin, L. Sherpa, H. Tang, Z. Zhu, Integrating AR and VR for mobile remote collaboration, in: 2019 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct), 2019, pp. 104–108, http://dx.doi.org/10.1109/ISMAR-Adjunct.2019.00041.

[32] Q. Jiao, G. Sun, Z. Chen, L. Han, J. Fan, A 3D webgis-enhanced representation method fusing surveillance video information, IEEE Access 11 (2023) 97024–97036, http://dx.doi.org/10.1109/ACCESS.2023.3312156.

[33] M. Aloqaily, I. Al Ridhawi, F. Karray, M. Guizani, Towards blockchain-based hierarchical federated learning for cyber-physical systems, in: 2022 International Balkan Conference on Communications and Networking (BalkanCom), 2022, pp. 46–50, http://dx.doi.org/10.1109/BalkanCom55633.2022.9900546.

[34] S. Xin, L. Zhuo, C. Xin, Node selection strategy design based on reputation mechanism for hierarchical federated learning, in: 2022 18th International Conference on Mobility, Sensing and Networking, MSN, 2022, pp. 718–722, http://dx.doi.org/10.1109/MSN57253.2022.00117.
[35] A. Joshi, S. Ismail, Experimental parallel architecture for rendering 3D model into MPEG-4 format, in: Proceedings of World Academy of Science: Engineering & Technology, Vol. 50, 2009, pp. 63–65.
[36] M. Hoppen, R. Waspe, M. Rast, J. Rossmann, Distributed information processing and rendering for 3D simulation applications, Int. J. Comput. Theory Eng. 6 (3) (2014) 247–253.
[37] C. Glez-Morcillo, D. Vallejo, J. Albusac, L. Jimenez, J. Castro-Schez, A new approach to grid computing for distributed rendering, in: 2011 International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 2011, pp. 9–16, http://dx.doi.org/10.1109/3PGCIC.2011.12.
[38] A. Clemm, M.T. Vega, H.K. Ravuri, T. Wauters, F.D. Turck, Toward truly immersive holographic-type communication: Challenges and solutions, IEEE Commun. Mag. 58 (1) (2020) 93–99, http://dx.doi.org/10.1109/MCOM.001.1900272.
[39] D. Liu, L. Wei, Q. Zheng, P. Ding, Y. Shen, Design and implementation of distributed rendering system, in: 2022 IEEE Smartworld, Ubiquitous Intelligence & Computing, Scalable Computing & Communications, Digital Twin, Privacy Computing, Metaverse, Autonomous & Trusted Vehicles (SmartWorld/UIC/ScalCom/DigitalTwin/PriComp/Meta), 2022, pp. 2366–2371, http://dx.doi.org/10.1109/SmartWorld-UIC-ATC-ScalCom-DigitalTwin-PriComp-Metaverse56740.2022.00332.
[40] C. Nandra, V. Bacu, D. Gorgan, Distributed, workflow-driven rendering of 3D object scenes on a big data processing platform, in: 2018 IEEE International Conference on Automation, Quality and Testing, Robotics, AQTR, 2018, pp. 1–6, http://dx.doi.org/10.1109/AQTR.2018.8402773.
[41] D. Abate, S. Migliori, S. Pierattini, B.J. Fenández-Palacios, A. Rizzi, F. Remondino, Remote rendering and visualization of large textured 3D models, in: 2012 18th International Conference on Virtual Systems and Multimedia, 2012, pp. 399–404, http://dx.doi.org/10.1109/VSMM.2012.6365951.
[42] H. Cheng, K. Xie, C. Wen, J.-B. He, Fast visualization of 3D massive data based on improved Hilbert R-tree and stacked LSTM models, IEEE Access 9 (2021) 16266–16278, http://dx.doi.org/10.1109/ACCESS.2021.3051911.
[43] H. Fang, N. Okumura, K. Ishii, S. Saito, Distributed rendering on grid computers for multiple users in shared virtual space, in: 2022 International Conference on Cyberworlds, CW, 2022, pp. 47–54, http://dx.doi.org/10.1109/CW55638.2022.00016.
[44] Y. Zhuang, L.R. Sheets, Y.-W. Chen, Z.-Y. Shae, J.J. Tsai, C.-R. Shyu, A patient-centric health information exchange framework using blockchain technology, IEEE J. Biomed. Health Inf. 24 (8) (2020) 2169–2176, http://dx.doi.org/10.1109/JBHI.2020.2993072.
[45] L.-Y. Yeh, P.J. Lu, S.-H. Huang, J.-L. Huang, SOChain: A privacy-preserving DDoS data exchange service over SOC consortium blockchain, IEEE Trans. Eng. Manage. 67 (4) (2020) 1487–1500, http://dx.doi.org/10.1109/TEM.2020.2976113.
[46] B. Huang, L. Jin, Z. Lu, X. Zhou, J. Wu, Q. Tang, P.C.K. Hung, BoR: Toward high-performance permissioned blockchain in RDMA-enabled network, IEEE Trans. Serv. Comput. 13 (2) (2020) 301–313, http://dx.doi.org/10.1109/TSC.2019.2948009.
[47] T.-S. Kang, M.-I. Joo, B.-S. Kim, T.-G. Lee, Blockchain-based lightweight transaction process modeling and development, in: 2021 23rd International Conference on Advanced Communication Technology, ICACT, 2021, pp. 113–118, http://dx.doi.org/10.23919/ICACT51234.2021.9370771.

Huabing Zhang is currently an associate professor at the Communication University of Zhejiang(CUZ) and deputy director of the School of Media Engineering. Her main research interests lie in digital media technology, computer vision, media communication, and image quality evaluation. She has authored or coauthored over 10 academic papers in core journals and international conferences.

Liang Li is currently a professor at School of Media Engineering, Communication University Of Zhejiang and Key Lab of Film and TV Media Technology of Zhejiang Province, Hangzhou, China. He has authored or co-authored over ten technical papers in international journals and at conferences. His research interests lie in augmented reality, virtual reality, services computing, computer vision, and 5G networks.

Qiong Lu is currently an associate professor at School of Media Engineering, Communication University of Zhejiang (CUZ), Hangzhou, China. Her research interests lie in media content management and computer control.

Yi Yue received the Ph.D. in Computer Science and Technology from the Beijing University of Posts and Telecommunications in 2021. He is currently serving as a senior engineer in the China Unicom Research Institute and the National Engineering Research Center of Next Generation Internet Broadband Service Application. His research interests include next-generation internet, cloud computing, and 6G technology.

Yakun Huang is currently a Postdoctoral Researcher at the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China. His current research interests include video streaming, mobile computing, and augmented reality.

Schahram Dustdar is a Full Professor of Computer Science at the TU Wien, heading the Research Division of Distributed Systems, Austria.He is co-founder of edorer.com (an EdTech company based in the US) and co-founder and chief scientist of Sinoaus.net, a Nanjing, China based R&D organization focusing on IoT and Edge Intelligence.

He is Editor-in-Chief of Computing (Springer). He is an Associate Editor of IEEE Transactions on Services Computing, IEEE Transactions on Cloud Computing, ACM Computing Surveys, ACM Transactions on the Web, and ACM Transactions on Internet Technology, as well as on the editorial board of IEEE Internet Computing and IEEE Computer. Dustdar is recipient of multiple awards: IEEE TCSVC Outstanding Leadership Award (2018), IEEE TCSC Award for Excellence in Scalable Computing (2019), ACM Distinguished Scientist (2009), ACM Distinguished Speaker (2021), IBM Faculty Award (2012). He is an elected member of the Academia Europaea: The Academy of Europe, as well as an IEEE Fellow (2016) and an Asia-Pacific Artificial Intelligence Association (AAIA) Fellow (2021) and the AAIA president (2021).