Protocol For Dynamic Load Distributed Low Latency Web-Based Augmented Reality And Virtual Reality

Rohit T P[1][0000−0002−2064−5020], Sahil Athrij[2], and Sasi Gopalan[1]

[1] Cochin University of Science and Technology, Kalamassery, Kerala India
20cs078rohi@ug.cusat.ac.in sasigopalan@cusat.ac.in
[2] University of Washington, Seattle, WA 98195, United States
sahilathrij@gmail.com

Abstract. The content entertainment industry is increasingly shifting towards Augmented Reality/Virtual Reality applications, leading to an exponential increase in computational demands on mobile devices. To address this challenge, this paper proposes a software solution that offloads the workload from mobile devices to powerful rendering servers in the cloud. However, this introduces the problem of latency, which can adversely affect the user experience. To tackle this issue, we introduce a new protocol that leverages AI-based algorithms to dynamically allocate the workload between the client and server based on network conditions and device performance. We compare our protocol with existing measures and demonstrate its effectiveness in achieving high-performance, low-latency Augmented Reality/Virtual Reality experiences.

Keywords: 2D kernel · Augmented reality · Cloud computing · Dynamic load distribution · Immersive experience · Mobile computing · Motion tracking · Protocols · Real-time systems · Web-based augmented reality application.

1 Introduction

The motivation for writing this paper is to address the increasing demand for computing power required to run AI-based algorithms in augmented reality (AR) applications. While these algorithms can enhance the physical world of the user by embedding virtual information seamlessly, the need for dedicated applications and hardware has always been a barrier to entry into the XR world for common users. This limitation is now being solved with the use of AI-based algorithms in place of hardware solutions (e.g. depth measurement, object tracking, etc.). However, this approach has resulted in an increased workload on consumer computing devices, which has made cloud-based solutions popular in most industries. Unfortunately, cloud-based AR/VR applications are still being left behind due to the two major barriers.

– Latency

– Server Cost

The common approach for solving these problems is a trade-off between quality and cost. We propose this protocol as a hybrid to attain high quality with minimal cost. In our technique only the bare minimum required amount of computing will be done on the server, that is the device will be utilized to its maximum. This allows the reduction of server costs. The protocol will also allow the allocation of load dynamically between the client and the server depending on the network conditions and device performance to attain minimum latency.

2 Related Works

2.1 Portable Augmented Reality

The quantity of programs utilizing Portable Augmented Reality (PAR)[3] has grown significantly in recent times due to the increased processing power available on portable devices. While most PAR programs can operate on current high-end portable devices, achieving an immersive experience requires processes such as accurate object recognition, lifelike model rendering, seamless occlusion, and precise augmentation in real-time, which cannot be handled by portable devices. Additionally, most mid-range and low-end portable devices are incapable of providing acceptable quality augmentation due to their limited resources and battery capacity.

2.2 Remote Gaming Technology

Remote gaming employs portable devices to transmit inputs to a distant server that renders and executes the game. The resulting video is streamed back to the portable device, allowing users to play high-quality games on a variety of devices without downloading them. This makes gaming devices more versatile and allows for improved performance scaling. Services such as Google Stadia[2], Microsoft xCloud, Amazon Luna, Nvidia GeForce Now, and PlayStation Now all utilize this technology to stream games to users' devices. In research on this topic, minimizing latency in the communication between the portable device and the server is a primary concern[4][13]. This approach can also be applied to developing protocols for web-based augmented reality to dynamically manage the distribution of computing tasks between the client and server.

2.3 Internet-Based Augmented Reality

Developing an Internet-based AR experience typically involves using a rendering library such as WebGL or THREE.js, a marker tracking library like AR.js, and APIs like WebXR to manage the AR session. However, browsers that execute these libraries have limited access to device resources and may not be capable of handling computationally intensive tasks like markerless detection and lifelike

rendering. Additionally, technologies like WebXR are still relatively new and do not provide options for custom detection algorithms. To enhance the performance of Internet-based AR, technologies such as SceneViewer by Google and ARQuickLook[19] by Apple have been created to handle rendering, lighting, surface detection, and placement within the browser. However, these technologies are specific to certain browsers and do not offer compatibility across different browsers. Additionally, they limit developer options such as custom detection algorithms and interaction. As a result, alternative solutions such as server-based AR systems are being developed to address these limitations.[21]

3 Components Of The Protocol

3.1 Client Side

The client architecture is divided into independent submodules for easy extension/modification. The client will be responsible for capturing frames from the camera feed, doing pre-processing, and optionally streaming the frames to the server. The client will also take care of displaying the final output in the desired format.

Frame Preprocessing The raw frames captured from the video source (most likely webcam/smartphone camera) will be of varying resolution and FPS depending on the video source. In the prepossessing stage, the client ensures the captured frames are below the maximum resolution supported by the server and also limits the frame rate if necessary. In cases where the video source is underperforming the preprocessing module just forwards the stream as is. The maximum resolution and frame rate can either be obtained from the server or hardcoded in the client itself. The recommended resolution is 1280x720 at 30 FPS.

Landmark Detection The landmark detection can happen in two modes depending on the device performance and network conditions, Low Compute Mode (LC) or High Compute (HC) mode. In LC the landmark detection is done by the server and the client merely streams the captured frames. In HC the client runs a comparatively lightweight (which can be changed by the developer) AI model to do the landmark detection and sends the detected landmarks to the server as a JSON.

3.2 Server Side

Transport Module It is the duty of the transport module to send and received data from the client. The module uses the data type of the received data to determine whether the client is in LC or HC mode. The transport module depending on this information automatically adjusts the pipeline to continue

the further process. The transport module is also responsible for spinning up as many instances of the pipeline as required to meet the client's needs. The transport module uses WebSockets to maintain consistent data communication between the client and server. To obtain maximum performance separate WebSockets running on different ports (if they are available else the same port is shared) are used for different clients.

Frame Processing This module is an optional part of the pipeline that only runs when the client is in LC mode. The frame processing module is responsible for decompressing the received frame and doing the landmark detection. This module produces normalized landmark points that are of the same structure as the landmark points generated by the client in HC mode.

Overlay Rendering Depending on the landmark points received (either from the client or Frame Processor) the overly has to be rendered. This involves 3D rendering and is computationally expensive and therefore is reserved for the server only. Appropriate 3d models are loaded and transformed to fit the landmarks and then rendered from the required perspective. The rendered images along with the anchor points are then passed on to the Transport module to send to the client.‘

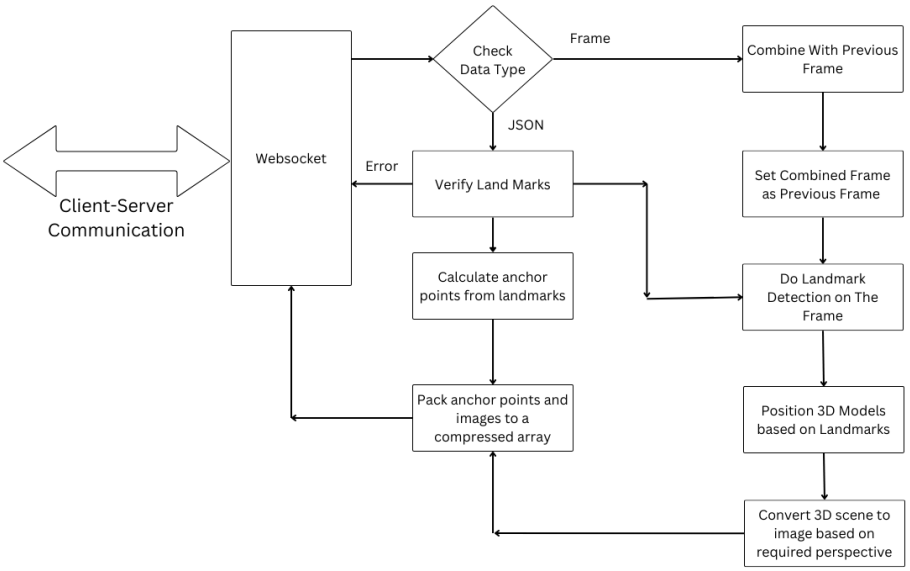

Fig. 1. Server Architecture

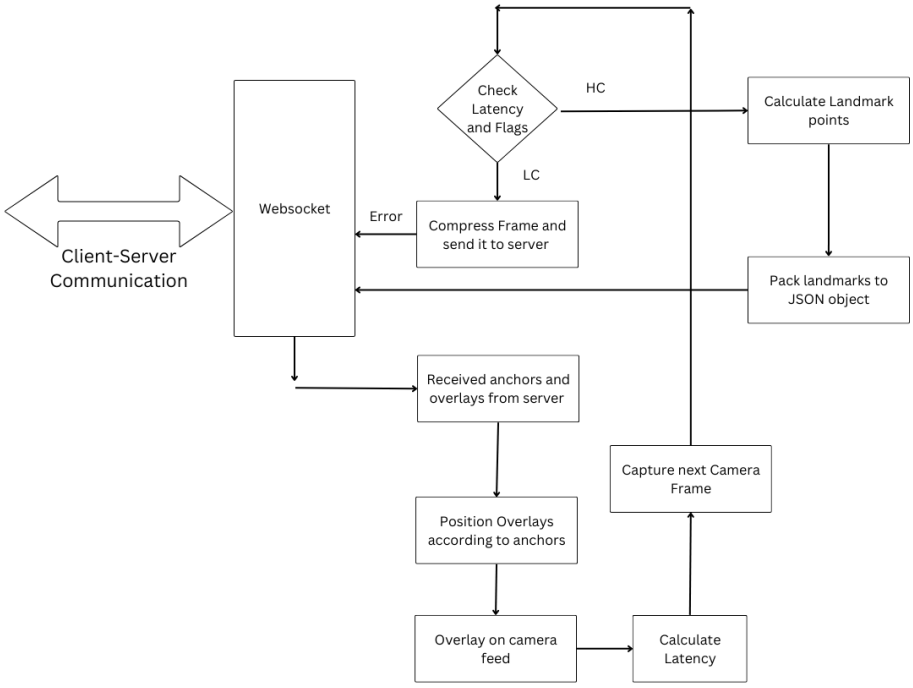

Fig. 2. Client Architecture

4 Model of Operation

4.1 Low Compute Mode

In LC mode the client device is deemed to be underpowered and the network condition is stable enough. In LC the client device doesn't do the landmark detection rather it sends the frame to the server for this task. To keep the latency low the images are compressed by the client.

Here it is very critical that the frames streamed should be of high quality since the AI needs to be able to detect landmarks from it. But streaming high-quality frames at rates near 30 FPS will require impractical bandwidth usage. The client solves this problem by using an improved inter-frame compression algorithm.

Compression

The common approach to inter-frame compression is to just compute the difference between consecutive frames and discard areas without change. Implementing this algorithm on CPU (sequential) will be $O(n^2)$ time complexity. So for an image of 720x720 pixels, it would take 518400 operations. This will slow down the process significantly. So the only viable solution is to use the GPU. In browser-based platforms, the GPU can be accessed using WebGL API. Using the WebGL shaders to perform the subtraction brings down the time complexity to practically $O(1)$. This is the approach commonly followed.

The shortcoming of this approach is that due to random noise and floating point precision issues the image difference may not always yield the required results. If there is noise in the image taking the pixel vise difference will always yield a huge number of non-zero pixels. This reduces the compression efficiency. Another caveat is that due to the floating point precision issue, the color produced in the subtracted frame will not be accurate. This hinders the reconstruction process.

Tunable Kernel and Region Of Interest We propose an improved version of inter-frame compression that overcomes the shortcomings of the conventional method. Instead of taking the pixel-wise difference of consecutive images a kernel is used to get values for each pixel before doing the subtraction.

For pixels a_{00} to a_{nn} and b_{00} to b_{nn}
Let $k(i, j)$ be the kernel value at i^{th} column and j^{th} row of the kernel

$$d_{ij} = (b_{ij} - a_{ij}) \times k(i, j) \tag{1}$$

$$sigDiff = \sum_{i=0}^{n} \sum_{j=0}^{n} d_{ij} \tag{2}$$

$$out_{(n/2)(n/2)} = \begin{cases} 0, & \text{if } sigDiff \leq threshold \\ b_{(n/2)(n/2)}, & \text{if } sigDiff > threshold \end{cases} \tag{3}$$

Here out_{ij} is the color of the pixel at location i, j

After calculating the image difference using equations 1 to 3 the pixels with values 0 are discarded. This use of kernel enables noise removal. The floating point precision error is also eliminated since the output doesn't directly depend on any result of float operations. The values of the kernel can be tuned as required to change how the noise is eliminated. The size of the kernel determines the granularity of regions of difference in the obtained image. With a larger kernel size, a more continuous part of the image will be selected whereas smaller values will select discrete scattered points/blobs.

Fig. 3. Output of 33×33 kernel

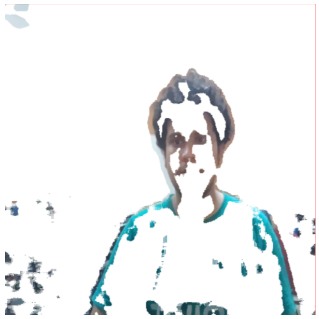

Fig. 4. Output of 3×3 kernel

At the receiving end the frames can be reconstructed by,

$$frame = framePrev[diffFrame \neq 0] = diffFrame[diffFrame \neq 0] \quad (4)$$

That is replace all changed pixels from the previous frame with sent pixels and keep pixels from the previous frame where there is no change. It has to be noted that the change discussed here is not the pixel-wise change but rather the change after applying the kernel.

4.2 High Compute Mode

High Compute mode is selected for devices with powerful processing capacity or when network conditions are very poor. In the High Compute mode, the client-side tries to do as much computation as possible. There are two configurations for HC mode for doing landmark detection,

Region Of Interest Tracking In this method instead of detecting and tracking the landmarks directly only a region of interest is calculated and tracked. Here the logic is that if a landmark is initially found in the region bounded by a

polygon P then the landmarks can change only when the pixels inside the set polygonal region change.

The client sends the first frame to the server for landmark detection. The server sends back landmark points along with the ROIs. The client keeps track of the landmark points according to the motion of ROIs.

The relative position of a particular landmark at time $t+\Delta t$ can be calculated by adding $V_c \times \Delta t$ to position of that landmark at time t

Here V_c is the velocity of the center of mass of the region enclosing that particular landmark. Where the center of mass of a region is defined as the average location of changing pixels.

Given two consecutive frames, the center of mass C can be calculated as,

$$C = (\frac{\Sigma x}{w}, \frac{\Sigma y}{h}) \tag{5}$$

w, h are the width and height of the region of interest

From this V_c will be,

$$V_c = \frac{|C(f_i) - C(f_{(i-1)}|}{\tau} \tag{6}$$

In order to reduce computation, the approximate velocity of motion can be calculated by taking the center of mass of the differential frame f$''$ given by,

$$f'' = f'_i - f'_{(i-i)} \tag{7}$$

$$V(f) = C(f'') \tag{8}$$

Object Detection In this configuration, the client runs an object detection model using pre-trained weights provided by the server. The weights are collected from the server to enable hot replacement of the detection model while maintaining the expected structure of landmark points. The client is responsible for requesting this mode. So developers can opt out of setting up an environment for running a model in the front end if necessary.

4.3 Dynamic Switching

The protocol will switch between LC and HC modes depending on the computed network latency and device performance. The switching information is inferred by the server automatically based on the data type of the transmitted data. This enables the client to react to real-time events without waiting for the server's response. There are four cases for the switching,

Low Compute Latency In this case, the time required for the device to do landmark detection on a frame is calculated. If this latency is below a threshold λ_c then Low Compute Latency is assumed. In this case, the protocol will default to HC mode. This can be overridden by the developer by using the force LC flag. This can be used if there are power consumption limitations for the device.

High Compute Latency In this case, the time required for the device to do landmark detection on a frame is calculated. If this latency is above a threshold λ_c then High Compute Latency is assumed. In this case, the protocol will switch dynamically between LC and HC modes. This can be overridden by the developer by using the force LC flag.

High Network Latency In this case, the round trip time for a frame is calculated. If this latency is above a threshold λ_n then High Network Latency is assumed. In this case, the protocol will switch dynamically between LC and HC modes until the conditions improve. This cannot be overridden by the developer.

Low Network Latency In this case, the round trip time for a frame is calculated. If this latency is below a threshold λ_n then Low Network Latency is assumed. In this case, the deciding factor will be Compute-Latency. The developer can force the use of LC by setting the LC Flag.

5 Results

These results are based on the implementation of this protocol with Flask-Python as the Backend (Server) and JS Frontend (Client).

5.1 Latency

The latency was measured as the time between a frame being captured and the corresponding overlay being drawn on the live camera feed. As expected the latency increases exponentially with an increase in frame resolution. This latency is the least of computing/network latency. It is achieved by dynamically switching between LC and HC modes.

The latency encountered for lower resolution images (lower than 144p) was found to be varying drastically so is not included in the table.

Table 1. Latency Vs Resolution.

Resolution	Latency
1280×720	20ms
852×480	15ms
480×360	12ms
256×144	10ms

5.2 Frame Size

The average frame size in KB was calculated from a sample of 546 frames. RAW indicates the maximum size required for a frame without any compression. From the table, it is evident that our modified inter-frame compression algorithm can reach a compression ratio about three times better than JPEG compression, which is the next best option.

The compressed image size of both PNG and JPEG remained fairly stable, 108–132 KB for PNG and 47–60 KB for JPEG. The compression ratio of our algorithm varied drastically depending on the amount of motion in the transmitted frame. It was in the range of 2.49–108 KB. Here even though the maximum frame size is 108 KB it is only reached when all the pixels of the frames change (e.g. The initial frame, when switching cameras etc.). But since the probability for such a change in a normal use case is very low our algorithm is able to outperform existing methods.

Table 2. Compression Using Each Techniques

Compression	Size
RAW	270 KB
PNG	104.40 KB
JPEG	56.22 KB
Ours	18.04 KB

6 Conclusion

In this paper, we have designed Protocol For Dynamic Load Distributed Low Latency Web-Based Augmented Reality And Virtual Reality (DLDL-WAVR) to introduce the method of dynamic load distribution between the client and the server for Augmented reality / Virtual Reality applications. The protocol was designed to be cross-platform and extremely configurable. The server architecture was proposed so as to enable it to be client agnostic. The mechanism to optimize based on network latency and device performance as well as to provide maximum quality at the lowest latency was devised. The results of the algorithm were rigorously tested and verified to match the theoretical values of the same.

The major drawback of the protocol is that it may not be able to handle large numbers of concurrent users, which can lead to performance degradation and increased latency. This is due to the lack of multiple server support in the backend. Despite these drawbacks, the DLDL-WAVR protocol has been shown to be a powerful and effective solution for dynamic load distribution in AR/VR applications, and further research is needed to address these limitations and improve its performance.

References

1. T. Bray. 2014. The JavaScript Object Notation (JSON) Data Interchange Format. RFC 7159 (2014), 1–16.
2. Marc Carrascosa and Boris Bellalta. 2020. Cloud-gaming:Analysis of Google Stadia traffic. arXiv:2009.09786 [cs.NI]
3. D. Chatzopoulos, C. Bermejo, Z. Huang, and P. Hui. 2017. Mobile Augmented Reality Survey: From Where We Are to Where We Go. IEEE Access 5 (2017), 6917–6950. https://doi.org/10.1109/ACCESS.2017.2698164
4. De-Yu Chen and Magda El-Zarki. 2017. Impact of Information Buffering on a Flexible Cloud Gaming System. In Proceedings of the 15th Annual Workshop on Network and Systems Support for Games (Taipei, Taiwan) (NetGames '17). IEEE Press, 19–24.
5. Jian-Wen Chen, Chao-Yang Kao, and Youn-Long Lin. 2006. Introduction to H.264 advanced video coding, Vol. 2006. 6 pp.–. https://doi.org/10.1109/ ASP-DAC.2006.1594774
6. I. Fette and A. Melnikov. 2011. The WebSocket Protocol. RFC 6455 (2011), 1–71.
7. Borko Furht (Ed.). 2008. JPEG. Springer US, Boston, MA, 377–379. https://doi.org/10.1007/978-0-387-78414-4_98
8. Borko Furht (Ed.). 2008. Portable Network Graphics (Png). Springer US, Boston, MA, 729–729. https://doi.org/10.1007/978-0-387-78414-4_181
9. IEEE Digital Reality 2020 (accessed September 16, 2020). Standards. IEEE Digital Reality. https://digitalreality.ieee.org/standards 13 AIVR 2021, July 23–25, 2021, Kumamoto, Japan Sahil Athrij, Akul Santhosh, Rajath Jayashankar, Arun Padmanabhan, and Jyothis P
10. K. Kiyokawa, M. Billinghurst, B. Campbell, and E. Woods. 2003. An occlusion capable optical see-through head mount display for supporting co-located collaboration. In The Second IEEE and ACM International Symposium on Mixed and Augmented Reality, 2003. Proceedings. 133–141. https://doi.org/10.1109/ISMAR.2003.1240696
11. Rob Kooper and Blair MacIntyre. 2003. Browsing the Real-World Wide Web: Maintaining Awareness of Virtual Information in an AR Information Space. International Journal of Human-Computer Interaction 16, 3 (1 Jan. 2003), 425–446. https://doi.org/10.1207/S15327590IJHC1603_3
12. Tobias Langlotz, Thanh Quoc Nguyen, Dieter Schmalstieg, and Raphael Grasset. 2014. Next Generation Augmented Reality Browsers: Rich, Seamless, and Adaptive. Proc. IEEE 102, 2 (2014), 155–169.
13. Kyungmin Lee, David Chu, Eduardo Cuervo, Johannes Kopf, Yury Degtyarev, Sergey Grizan, Alec Wolman, and Jason Flinn. 2015. Outatime: Using Speculation to Enable Low-Latency Continuous Interaction for Mobile Cloud Gaming. In Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services (Florence, Italy) (MobiSys '15). Association for Computing Machinery, New York, NY, USA, 151–165. https://doi.org/10.1145/2742647.2742656
14. Li Liu, Wanli Ouyang, Xiaogang Wang, Paul Fieguth, Jie Chen, Xinwang Liu, and Matti Pietikäinen. 2019. Deep Learning for Generic Object Detection: A Survey. arXiv:1809.02165 [cs.CV]
15. Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. 2016. SSD: Single Shot MultiBox Detector. In Computer Vision – ECCV 2016, Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling (Eds.). Springer International Publishing, Cham, 21–37.

16. Huynh Nguyen Loc, Youngki Lee, and Rajesh Krishna Balan. 2017. DeepMon: Mobile GPU-based Deep Learning Framework for Continuous Vision Applications.. In MobiSys, Tanzeem Choudhury, Steven Y. Ko, Andrew Campbell, and Deepak Ganesan (Eds.). ACM, 82–95. http://dblp.unitrier.de/db/conf/mobisys/mobisys2017.html
17. B. MacIntyre, A. Hill, H. Rouzati, M. Gandy, and B. Davidson. 2011. The Argon AR Web Browser and standards-based AR application environment. In 2011 10th IEEE International Symposium on Mixed and Augmented Reality. 65–74. https://doi.org/10.1109/ISMAR.2011.6092371
18. Thomas Olsson and Markus Salo. 2011. Online user survey on current mobile augmented reality applications. 2011 10th IEEE International Symposium on Mixed and Augmented Reality, ISMAR 2011, 75 – 84. https://doi.org/10.1109/ISMAR.2011.6092372
19. Z. Oufqir, A. El Abderrahmani, and K. Satori. 2020. ARKit and ARCore in serve to augmented reality. In 2020 International Conference on Intelligent Systems and Computer Vision (ISCV). 1–7. https://doi.org/10.1109/ISCV49265.2020.9204243
20. PTCGroup. 2020. Vuforia Developer Portal. https://developer.vuforia.com/
21. X. Qiao, P. Ren, S. Dustdar, L. Liu, H. Ma, and J. Chen. 2019. Web AR: A Promising Future for Mobile Augmented Reality—State of the Art, Challenges, and Insights. Proc. IEEE 107, 4 (2019), 651–666. https://doi.org/10.1109/JPROC.2019.2895105
22. X. Qiao, P. Ren, S. Dustdar, L. Liu, H. Ma, and J. Chen. 2019. Web AR: A Promising Future for Mobile Augmented Reality—State of the Art, Challenges, and Insights. Proc. IEEE 107, 4 (2019), 651–666. https://doi.org/10.1109/JPROC.2019.2895105
23. J. Redmon and A. Farhadi. 2017. YOLO9000: Better, Faster, Stronger. In 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 6517–6525. https://doi.org/10.1109/CVPR.2017.690
24. Hafez Rouzati, Luis Cruiz, and Blair MacIntyre. 2013. Unified WebGL/CSS Scene-Graph and Application to AR. In Proceedings of the 18th International Conference on 3D Web Technology (San Sebastian, Spain) (Web3D '13). Association for Computing Machinery, New York, NY, USA, 210. https://doi.org/10.1145/2466533.2466568
25. Ryan Shea, Andy Sun, Silvery Fu, and Jiangchuan Liu. 2017. Towards Fully Offloaded Cloud-Based AR: Design, Implementation and Experience. In Proceedings of the 8th ACM on Multimedia Systems Conference (Taipei, Taiwan) (MMSys'17). Association for Computing Machinery, New York, NY, USA, 321–330. https://doi.org/10.1145/3083187.3084012
26. N. Sun, Y. Zhu, and X. Hu. 2019. Faster R-CNN Based Table Detection Combining Corner Locating. In 2019 International Conference on Document Analysis and Recognition (ICDAR). IEEE Computer Society, Los Alamitos, CA, USA, 1314–1319. https://doi.org/10.1109/ICDAR.2019.00212
27. Ta Nguyen Binh Duong and Suiping Zhou. 2003. A dynamic load sharing algorithm for massively multiplayer online games. In The 11th IEEE International Conference on Networks, 2003. ICON2003. 131–136. https://doi.org/10.1109/ICON.2003.1266179
28. Y. Tao, Y. Zhang, and Y. Ji. 2015. Efficient Computation Offloading Strategies for Mobile Cloud Computing. In 2015 IEEE 29th International Conference on Advanced Information Networking and Applications. 626–633. https://doi.org/10.1109/AINA.2015.246

29. X. Zhou, W. Gong, W. Fu, and F. Du. 2017. Application of deep learning in object detection. In 2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS). 631–634. https://doi.org/10.1109/ICIS.2017.7960069
30. Christian Zimmermann and Thomas Brox. 2017. Learning to Estimate 3D Hand Pose from Single RGB Images. In IEEE International Conference on Computer Vision (ICCV). https://lmb.informatik.uni-freiburg.de/projects/hand3d/ https://arxiv.org/abs/1705.01389.