International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

A Study on the Classification of Hand Gesture for Mobile Virtual Reality with MediaPipe

Beom Jun Jo[1], Seong Ki Kim*[2]

Submitted: 05/02/2024 **Revised:** 13/03/2024 **Accepted:** 20/03/2024

Abstract: Virtual devices' mobile application processors continue to evolve, and technologies are emerging accordingly. Although the virtual reality with hand tracking enables to manipulate the contents without a controller, the devices support only the limited actions, and the hand tracking can be used only for the simple games. However, hand tracking can provide users with an intuitive, comfortable feeling of operation and preventing accidents by using their hands right away. To track hands, there is a MediaPipe from Google that enables hand tracking with a normal webcam. This paper describes how to use a new motion for the virtual reality contents using MediaPipe for utilizing the advantages of hand tracking as a user interface. The implementation is a game using a hand gesture only. Also, this paper compares three implementations with different implementation methods of MediaPipe: ported from C++ to C#, using tflite, using Barracuda. Comparisons were made on both PC and mobile. On PC, Barracuda was the fastest with a maximum of 208 frames per second, but on mobile, Barracuda was the slowest with a minimum 12 frames per second. For this reason, this may vary depending on the project, it seems that it is still difficult to apply Barracuda to mobile contents.

Keywords: *Augmented Reality, Barracuda, MediaPipe, Virtual Reality*

1. Introduction

Virtual Reality (VR) with a hand tracking provides users with the controlling methods of the game character's movements without using a controller. Due to its importance, Apple's Vision Pro leads in advocating for the use of hand tracking, and PICO also acknowledges the technology's significance in the future. While hand tracking currently has limitations for gaming purposes, it offered convenience to users and could be utilized to control the virtual reality, mixed reality (MR), and extended reality (XR). As a result, many companies developed the hand gesture. And Fig. 1 illustrates the supported gestures by Meta's devices as an example. In Fig. 1, three gestures are depicted: Point and Pinch for selecting something, Pinch and Scroll for scrolling, and Palm and Pinch for bringing the user back to the Meta Home Menu. Besides the Meta's devices, other VR and XR devices support the hand tracking with their own gestures. These supported gestures place a limitation on the supported actions, thus current hand tracking technology can support casual games. As an example, there is a game named Cubism that player solves block puzzles using hands. Despite its limited actions, hand tracking expands the scope of manipulation to non-virtual devices, offering players greater intuitiveness and convenience. For example, players don't have to find a controller if hand tracking is the main control method. Moreover, it has potential in a range of fields outside of gaming, including rehabilitation and sign language.

This study aims to address the issue of limited hand gestures by expanding recognizable hand movements. In this paper, we implement a game that can recognize various hand postures and tried to recognize a sword-wielding motion as an example. For the purpose, MediaPipe is employed to track the hand. Also, we compare their performance with other MediaPipe implementations.

The contributions of this paper can be summarized next. First, this paper shows a game using MediaPipe implemented using Barracuda in Unity. Second, this paper is the first study that compares a method through Barracuda and the method without it.

This paper is organized as follows: Section 2 shows related works.

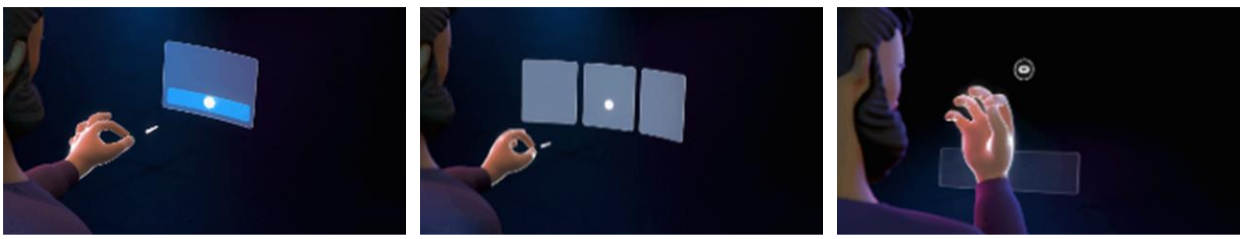

Fig. 1. Three hand tracking gestures of Meta's devices.
The images in turn are Point and Pinch, Pinch and Scroll, and Palm and Pinch.

In Section 3, we implemented a game using hand gesture. In Section 4, we compared MediaPipe's implementation. Section 5 concludes this paper.

1 Department of Game Design and Development, Sangmyung University, Seoul, 03016, KOREA
ORCID ID : 0009-0006-3967-1372
2 Department of Computer Engineering, Chosun University, Gwangju, 61452, KOREA
ORCID ID : 0000-0002-2664-3632
** Corresponding Author Email: skkim@chosun.ac.kr*

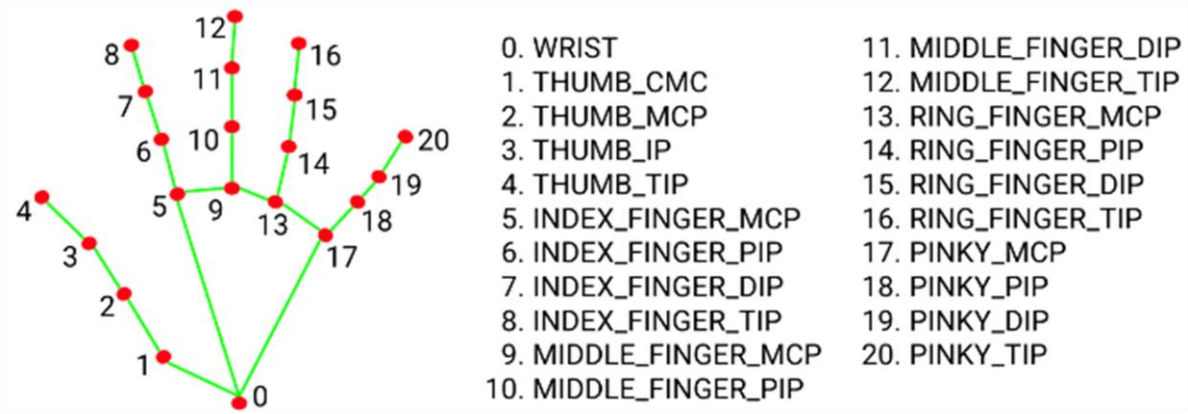

Fig. 2. 21 Hand Landmarks available by MediaPipe

2. Related Works

2.1. MediaPipe

MediaPipe [1] is a machine learning solution created by Google. MediaPipe has been developed since 2019 for ease of use by researchers or other developers. Because it combines computer vision, natural language, and audio, it provides a wide range of functions, including vision functions such as object and face detection, image segmentation, pose tracking, text-related functions such as character classification, text embedding, language detection, audio classification and embedding. As shown in Fig. 2, MediaPipe provides 21 landmarks.

MediaPipe [2] has been used in many research studies. For example, [3] implemented detection of Vietnamese sign language using a recurrent neural network (RNN) with MediaPipe. [4] proposed a system that recognizes human body movements in video using Blazepose of MediaPipe, body tracking solution, for the field of sports and compared it with an Inertial Measurement Unit (IMU)-based motion capture.

2.2. Barracuda

Unity has two libraries using AI: one is ML-agent for reinforcement learning and the other is Barracuda [5] for inferencing in this paper. Barracuda first came out in 2019 and continue to being developed under the name “Sentis”. In the case of Sentis, it is a preview version, so this paper uses the stable Barracuda.

Barracuda is an inference library for Unity using neural network and uses Open Neural Network eXchange (ONNX) [6]. ONNX is specialized in inferencing and a system designed to make models developed in different frameworks compatible with each other as shown in Fig. 3. Thus, one of the advantages of ONNX is a framework interoperability. Another is a shared optimization, which allows you to optimize based on the intermediate representation of ONNX.

Because of the above advantages, it can be run anywhere if the device supports it. In the case of Unity, Barracuda is what makes ONNX run. Also, the current Unity supports many devices such as desktop, mobile, and VR. And this also expects to support the visionOS for Apple Vision Pro that hasn’t been released yet. Just as Unity is a cross-platform, Barracuda supports cross-platform [7].

[8] uses Barracuda to integrate an object detector. Using this, the paper implements detecting interactions between human hands and objects. [9] also used Barracuda to create models using Pytorch and its operate in Unity. [10] created traditional culture-based metaverse content by recognizing human movements in Unity and mapping them with lion mask avatars. Barracuda was used in this process. [11] mentioned that they used to run a separate machine learning server and set up to use mobile as a client, but with Barracuda, they could run it on the client without an ML server. As such, studies using Barracuda can be found in recent papers.

3. Implementation

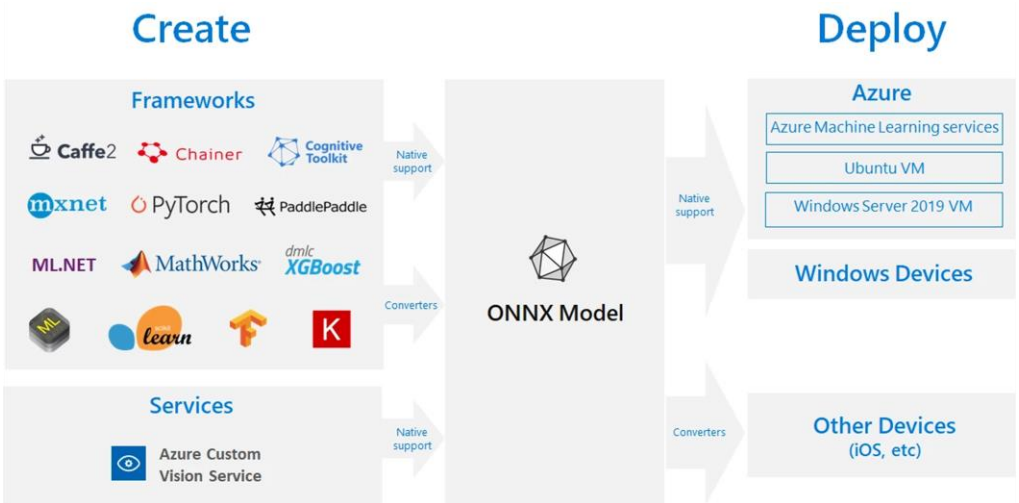

Fig. 3. Existing device discovery in Remote Device Management (RDM)

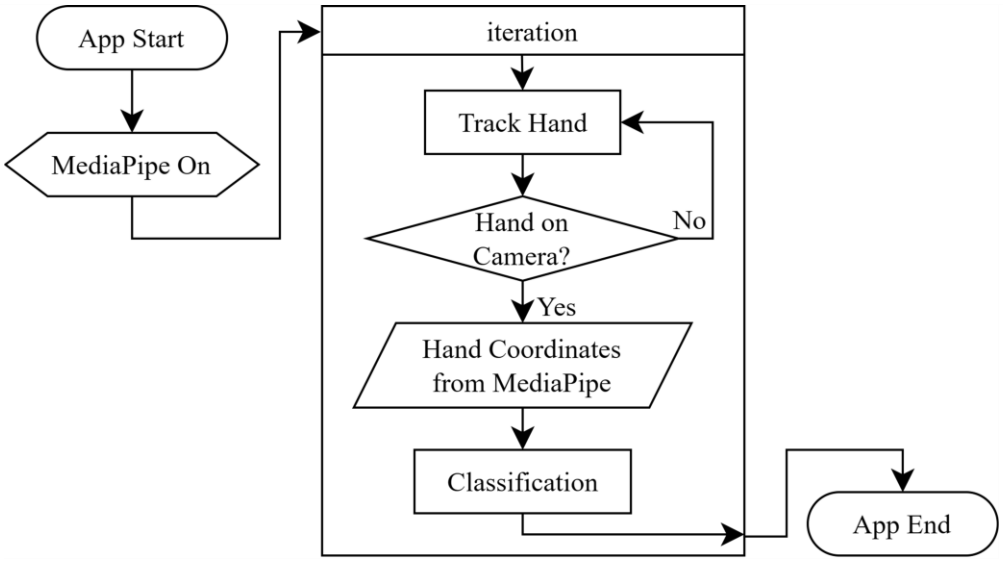

Fig. 4. The flowchart of implementation.

In this research, we developed a game that recognize the hand gesture of holding a knife (fist) in Unity as an example. In the game, players can earn points by slicing fruits that rise from the bottom with their hand gesture.

$$\overrightarrow{AB} = (b_x - a_x, b_y - a_y, b_z - a_z) \quad (1)$$

$$cos\theta = \frac{\vec{c}\cdot\vec{d}}{|\vec{c}||\vec{d}|} \quad (2)$$

In (1), a and b are one of the 21 points in Fig. 2. The two vectors obtained using (1) become c, d of (2) and use them to obtain θ. The implementation operates according to the flowchart of Fig. 4. And the process is as follows:

1. When the game turns on, MediaPipe works at the same time.
2. MediaPipe tracks hand through the camera.
3. When a hand appears on the camera, the hand landmark coordinates are obtained as shown in Fig. 2.
4. Then implementation classifies whether the gesture is necessary for the game with the corresponding data.
5. Repeat 3 and 4 until the app turns off.

In the 4th process, the distance between the two points and the angle between them can be obtained according to the (1) and (2). The folding of each finger was implemented when each angle of $\overrightarrow{p_1p_5}$ (the vector of point 1 and point 5 in Fig. 2.) and $\overrightarrow{p_6p_7}$, $\overrightarrow{p_1p_9}$ and $\overrightarrow{p_{10}p_{11}}$, $\overrightarrow{p_0p_{13}}$ and $\overrightarrow{p_{14}p_{15}}$, $\overrightarrow{p_0p_{17}}$ and $\overrightarrow{p_{18}p_{19}}$ is above $\pi/2$. In other words, the application recognizes the user's fist when the angle obtained in front of all fingers exceeds 90°. Additionally, when using the angle between $\overrightarrow{p_5p_{17}}$ and the y-axis determines that his fist is raised vertically with the back of hand facing outward as holding a knife, the slicing starts. And if the angle of one finger becomes smaller than 90° and the fist is loosened, the slicing stops. Fig. 5 shows the example that our algorithm recognized a hand.

As shown in Fig. 5, our algorithm recognized the hand. As stated in this section, when the user moves his fist, he can slice fruits in the game. However, because swift swinging motion is not yet recognized in real-time tracking, the game can't recognize if the user make an action like brandishing a knife quickly for getting a lot of points at once.

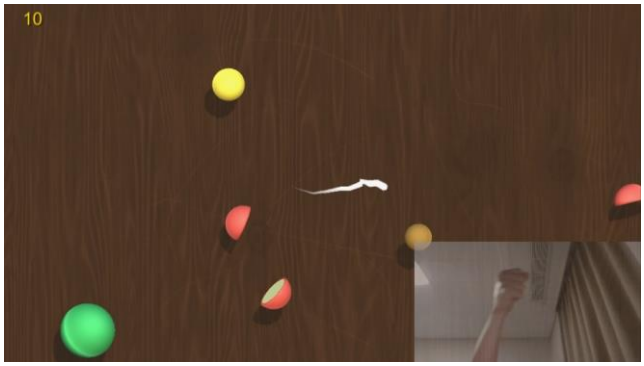

Fig. 5. Playing game with a recognized hand

4. Comparison

In this study, MediaPipe was used and we implemented the game in three ways. First, there is a MediaPipe ported to C#. The original MediaPipe is available in Java on Android, Objective-C on iOS, Python, and C++. This version ported C++ to C#, the language of Unity. Second, there is a implementation that uses tflite. Tflite is a tensorflow library for mobile and smaller devices, and MediaPipe had versions using tflite. And this is what makes the tflite work in Unity to do hand tracking. Finally, the last one is the version that uses the Barracuda. Barracuda is Unity's own inference library, which allows unity to read and run ONNX models. In this way, each of the three MediaPipes show their own implementation methods, and there will be a difference.

The next tables compare the performance that can be expressed in numbers. A laptop with an AMD Ryzen 7 4800H and a GTX 1660 TI and a Galaxy Z Flip5 with Qualcomm Snapdragon 8 Gen 2 customized for Galaxy are used as the comparison environment.

Table 1 and Table 2 show the time taken to display the next frame and the FPS calculated from the time taken on a PC and a mobile environment, respectively. The time it takes to output the next frame is the one that takes for MediaPipe to find a hand and to draw a picture on the frame from a single frame that enters the camera. The way to obtain FPS is to divide the time to display for the frame to come out by a one second. Since the unit of time utilized in tables above is milliseconds, FPS was calculated by dividing 1000 msec by the time taken for the frames to appear and rounding it.

As shown in Table 1, Barracuda provides similar speeds to the

ported version, with the lowest being 4.8 msec. On the other hand, the implementation using tflite showed 3 FPS as the lowest FPS, which would not be acceptable for gaming, even on a PC.

Table 1. Comparison of time taken to output the next frame and FPS on PC

	Min. time (msec)	Min. FPS	Max. time (msec)	Max. FPS
Ported	5.2	192	14.1	71
Tflite	155.9	6	347.4	3
Barracuda	4.8	208	8.4	119

However, Table 2 contradicts the results in Table 1. MediaPipe with Barracuda was the slowest. Barracuda's fastest frame output moment was 65.1 msec, which is 15 FPS, but that's only half of the other projects. On mobile, the ported one was the fastest. Additionally, the ported version and tflite version show 30 FPS when it is the fastest, which does not seem to go below 30 FPS by camera setup. If the resolution can be lowered and the framerate of the camera can be raised, it will show faster speed.

Table 2. Comparison of time taken to output the next frame and FPS on mobile

	Min. time (msec)	Min. FPS	Max. time (msec)	Max. FPS
Ported	33.5	30	36.8	27
Tflite	33.5	30	68.2	15
Barracuda	65.1	15	80.4	12

Fig. 6 describes the captured data of the hand that can be obtained when recognizing the hand like the image in row (a) in each model. In turn, (b), (c), and (d) are the ported version, the version using tflite, and the version using Barracuda. All versions are recognizable, but the case of (c) has difficulty recognizing the first

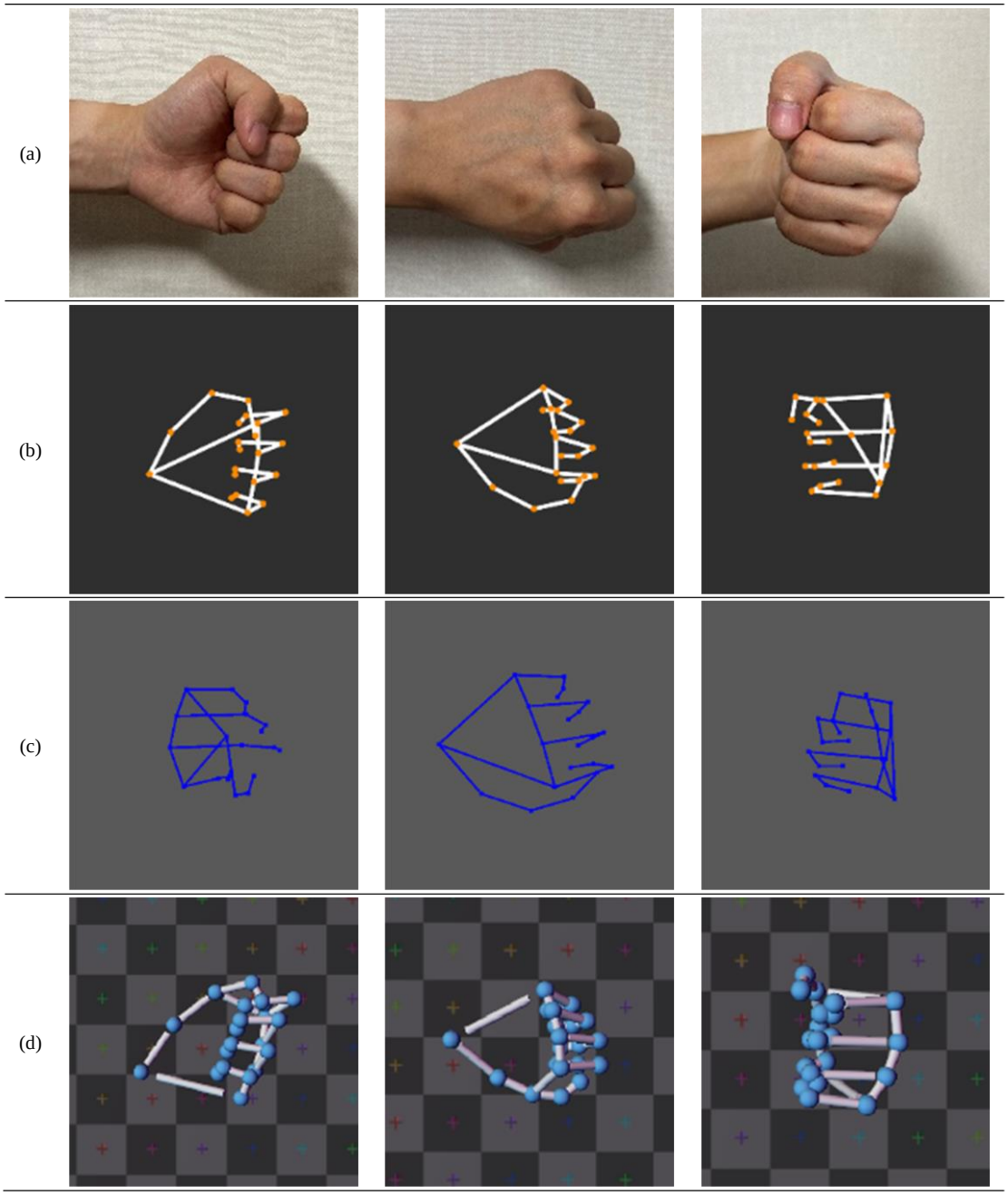

Fig. 6. Comparison of hand gesture recognition in three directions between three MediaPipe (a) hand as input data (b) Ported (c) tflite (d) Barracuda

direction.

5. Conclusion

This paper described how to use a hand for the gesture interface and recognize a hand with a knife shape. We used a MediaPipe to track hands and a heuristic to recognize hand gestures and compared the three implementations. However, there is still a limit to hand tracking so far because we cannot keep up with your hand when you make a quick motion like swinging. Although Barracuda showed the fastest performance on PC, it was the slowest on mobile when compared to the other implementations. The other two models may be faster by mobile camera setup. Using Barracuda, which is a lightweight cross-platform library that works on Unity, it may be useful to implement other projects including MediaPipe on PCs, but it does not appear to have performance yet for mobile.

As future work, if there is only a camera without device constraints, we are working on improving the operation sensibility through manipulation using hand tracking and aim to improve stability and apply AI rather than heuristic.

Acknowledgement

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIT). (NRF-2023R1A2C1005950). To increase the understandability of this paper, the authors have created a video and uploaded it to https://youtu.be/tHM8f78mBd8.

Author contributions

BeomJun Jo: Conceptualization, Methodology, Implementation, Writing-Original draft preparation **SeongKi Kim:** Conceptualization, Validation, Writing-Reviewing and Editing, Funding.

Conflicts of Interest

The authors declare no conflicts of interest.

References

[1] F. Zhang et al., “MediaPipe Hands: On-device Real-time Hand Tracking,” arXiv, Jun. 2020. doi: 10.48550/arXiv.2006.10214.

[2] C. Lugaresi et al., “MediaPipe: A Framework for Building Perception Pipelines,” arXiv, Jun. 2019. doi: 10.48550/arXiv.1906.08172.

[3] B. Duy Khuat, D. Thai Phung, H. Thi Thu Pham, A. Ngoc Bui, and S. Tung Ngo, “Vietnamese sign language detection using Mediapipe,” in *Proc. of the 2021 10th International Conference on Software and Computer Applications (ICSCA '21)*. New York, NY, USA: Association for Computing Machinery, Jul. 2021, pp. 162–165. doi: 10.1145/3457784.3457810.

[4] A. S. B. Pauzi et al., “Movement Estimation Using Mediapipe BlazePose,” in Advances in Visual Informatics, 2021, pp. 562–571. doi: 10.1007/978-3-030-90235-3_49.

[5] Unity Manual “Introduction to Barracuda.” Accessed: Apr. 11, 2024. [Online]. Available: https://docs.unity3d.com/Packages/com.unity.barracuda@3.0/manual/index.html

[6] ONNX documentation, “Introduction to ONNX.” Accessed: Aug. 28, 2023, [Online]. Available: https://onnx.ai/onnx/intro/

[7] Unity Japan, “Multi-platform operation of ONNX neural network model using Unity Barracuda - CEDEC2021”, Aug. 2021. Accessed: Sep. 04, 2023, [Online]. Available: https://www.youtube.com/watch?v=dMgm4ZYfaUI

[8] M. Mazzamuto, F. Ragusa, A. Resta, G. Farinella, and A. Furnari, “A Wearable Device Application for Human-Object Interactions Detection,” in *Proc. of the 18th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, Lisbon, Portugal, 2023, pp. 664–671. doi: 10.5220/0011725800003417.

[9] R. C. Castanyer, S. Martínez-Fernández, and X. Franch, “Integration of Convolutional Neural Networks in Mobile Applications,” arXiv, Mar. 2021. doi: 10.48550/arXiv.2103.07286.

[10] J. Kim, J. Lee, M. Kim, and D. Kim, “Design and development of traditional lion mask avatar mapping and animation system based on the user motion recognition using deep learning technology,” in *Proc. HCI Korea 2023*, 2023, pp. 5–9.

[11] G. Garg and S. Shivani, “Controller free hand interaction in Virtual Reality,” in *2022 OITS International Conference on Information Technology (OCIT)*, Feb. 2022, pp. 553–557. doi: 10.1109/OCIT56763.2022.00108.