MobiDepth: Real-Time Depth Estimation Using On-Device Dual Cameras

Jinrui Zhang[1†], Huan Yang[1], Ju Ren[2], Deyu Zhang[1*], Bangwen He[1], Ting Cao[3], Yuanchun Li[4], Yaoxue Zhang[2], Yunxin Liu[4*]
[1]School of Computer Science and Engineering, Central South University
[2] Department of Computer Science and Technology, Tsinghua University [3]Microsoft Research
[4] Institute for AI Industry Research (AIR), Tsinghua University
[1]{zhangjinrui, yanghuan9812, zdy876, hebangwen}@csu.edu.cn
[2]{renju, zhangyx}@tsinghua.edu.cn, [3]ting.cao@microsoft.com
[4]{liyuanchun, liuyunxin}@air.tsinghua.edu.cn

ABSTRACT

Real-time depth estimation is critical for the increasingly popular augmented reality and virtual reality applications on mobile devices. Yet existing solutions are insufficient as they require expensive depth sensors or motion of the device, or have a high latency. We propose MobiDepth, a real-time depth estimation system using the widely-available on-device dual cameras. While binocular depth estimation is a mature technique, it is challenging to realize the technique on commodity mobile devices due to the different focal lengths and unsynchronized frame flows of the on-device dual cameras and the heavy stereo-matching algorithm.

To address the challenges, MobiDepth integrates three novel techniques: 1) iterative field-of-view cropping, which crops the field-of-views of the dual cameras to achieve the equivalent focal lengths for accurate epipolar rectification; 2) heterogeneous camera synchronization, which synchronizes the frame flows captured by the dual cameras to avoid the displacement of moving objects across the frames in the same pair; 3) mobile GPU-friendly stereo matching, which effectively reduces the latency of stereo matching on a mobile GPU. We implement MobiDepth on multiple commodity mobile devices and conduct comprehensive evaluations. Experimental results show that MobiDepth achieves real-time depth estimation of 22 frames per second with a significantly reduced depth-estimation error compared with the baselines. Using MobiDepth, we further build an example application of 3D pose estimation, which significantly outperforms the state-of-the-art 3D pose-estimation method, reducing the pose-estimation latency and error by up to 57.1% and 29.5%, respectively.

† This work was done during internship at Institute for AI Industry Research (AIR), Tsinghua University.
* Corresponding author.

CCS CONCEPTS

• **Computer systems organization → Real-time systems**; • **Human-centered computing → Ubiquitous and mobile computing**.

KEYWORDS

Real Time, Depth Estimation, Dual Camera, Mobile Device, OpenCL

ACM Reference Format:
Jinrui Zhang[1†], Huan Yang[1], Ju Ren[2], Deyu Zhang[1*], Bangwen He[1], Ting Cao[3], Yuanchun Li[4], Yaoxue Zhang[2], Yunxin Liu[4*]. 2022. MobiDepth: Real-Time Depth Estimation Using On-Device Dual Cameras. In *The 28th Annual International Conference On Mobile Computing And Networking (ACM MobiCom '22), October 17–21, 2022, Sydney, NSW, Australia.* ACM,New York, NY, USA, 14 pages. https://doi.org/10.1145/3495243.3560517

1 INTRODUCTION

In recent years, the mobile industry and research community have significantly invested in augmented reality (AR) and virtual reality (VR) applications for mobile devices [8, 11, 20, 37, 53, 54]. Statistics have shown that the AR/VR market has reached 30.7 billion dollars in 2021 and will rise to about 300 billion dollars by 2024 [50]. Among many technologies that enable diverse AR/VR applications on mobile devices, real-time depth estimation is a fundamental building block that connects the physical world to its 3D digital representation. For example, a key feature in AR applications is to render virtual objects on the digital surface of physical objects, and the surface is computed through depth estimation.

Currently, there are mainly three types of solutions for depth estimation on mobile devices: 1) **Dedicated depth sensors.** Some devices are equipped with dedicated depth sensors such as LiDAR, ToF camera, and structured-light sensor. These sensors work by emitting light in a specific spectrum and calculating the depth based on the light reflected back. Although these sensors can achieve precise depth estimation, they are only available on a few high-end mobile devices due to the high cost. 2) **Learning-based depth prediction.** Machine learning models, such as convolutional neural networks (CNNs), can learn to predict the depth by training on labeled data [48, 52]. However, the capability of learning-based approaches relies heavily on the training dataset. They are usually unable to achieve satisfactory accuracy for new scenes and new objects that are not included in the dataset, as shown in Figure 1 (c)(d)(e). Furthermore, the models are generally heavy and difficult to run in a real-time manner on computing limited mobile devices.

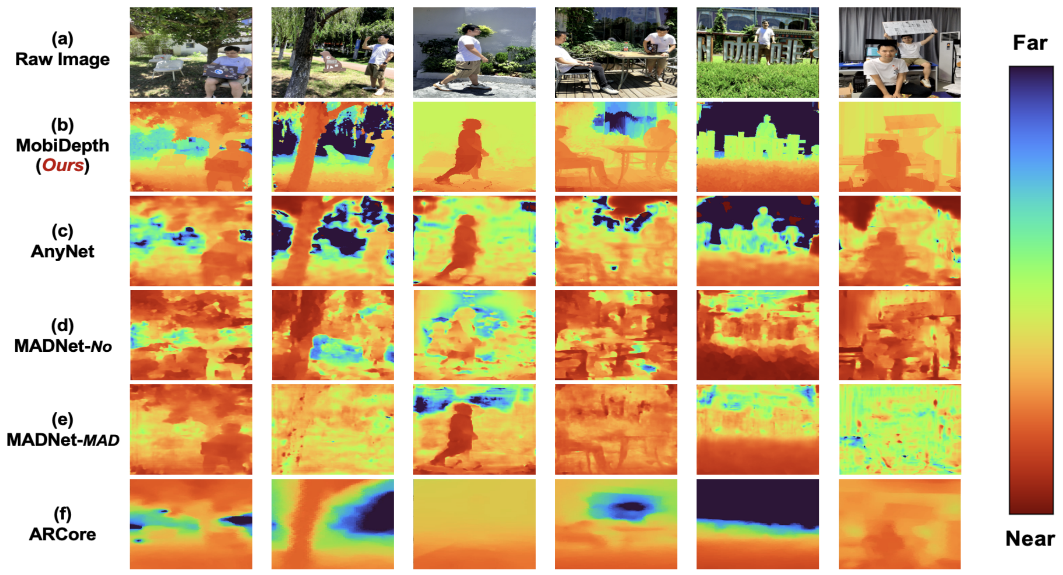


Figure 1: Example depth maps generated by MobiDepth, AnyNet, MADNet with and without online adaptation (named MADNet-MAD and MADNet-No, respectively), and ARCore, with the person sitting, walking and standing. (a) raw images. (b) our approach estimates accurate depth with crisp edges. (c) and (d) the learning-based depth estimation models, i.e., AnyNet and MAD-No, poorly perform in scenarios different from the training dataset. (e) the performance of MADNet-MAD is still unsatisfactory, even with the extremely time consuming online adaptation module. (f) ARCore barely estimates depth in these images.

The latency of the state-of-the-art (SOTA) models [48, 52] are as high as 80ms to 550ms on the high-end Huawei Mate40Pro smartphone, as shown in Section 8. 3) **Depth from motion.** The most common solution adopted by existing mobile systems (including ARKit in iOS [12] and ARCore in Android [13]) is the depth-from-motion algorithm [49]. The algorithm works by using visual-inertial odometry (VIO) to combine information from inertial measurement unit (IMU), with computer vision analysis of the scene visible to the camera to obtain the depth, *i.e.,* select keyframes during the motion of camera and estimate depth based on stereo matching between the most recent image and a past keyframe. Although this solution does not rely on dedicated sensors or large-scale training data, it requires the camera to be moving and expects the target object to be stationary, which signicantly restricts its usage scenarios. Figure 1(e) shows the performance of ARCore to get the depth of the moving person, it is obvious that the accuracy of ARCore's depth estimation is terrible in this scenario.

Inspired by the success of binocular depth estimation techniques, we nd that the distribution of the rear-facing cameras on mobile devices brings a great opportunity for depth estimation. Ideally, the disparity can be readily obtained by comparing the pair of frames captured by the dual cameras. The displacement of the dual cameras provides a stable baseline, compared to the depth from motion solution. However, our in-depth analysis reveals several challenges in using the dual rear-facing cameras for depth estimation, as follows: 1) *How to reduce the impact of the diverse focal lengths of the dual cameras.* The rear-facing cameras are originally designed to serve various application scenarios, such as macro shooting or wide angle shooting. Their focal lengths are thus quite diverse. It greatly impacts the accuracy of the epipolar rectication which serves as the basis for depth estimation. 2) *How to synchronize the frame flows captured by the dual cameras.* The frame ows are highly out of sync, due to the impacts of dierent frame periods of the cameras, as well as the frequent garbage collection (GC). Suering from the out-of-sync frame ows, estimating the depth of objects in motion becomes impossible, since the objects have displacement in the pair of frames. 3) *How to accelerate the stereo matching on computation-limited mobile devices.* The state-of-the-art stereo matching algorithms and CNN models, *e.g.,* Semi-Global-Block Matching (SGBM), MADNet, and HITNet, are computation heavy, leading to long runtime on mobile devices. The stereo matching needs to run in an online manner to nd the correspondence between the points in the pair of frames. The long runtime leads to the low refresh rate of the depth estimation applications.

Addressing the above challenges, we propose MobiDepth to leverage the rear-facing dual cameras to estimate depth in real-time on mobile devices. MobiDepth resolves all the issues of the three existing solutions, *i.e.,* it does not rely on any dedicated sensors or pre-training, and works well for target objects in motion. MobiDepth integrates several new techniques, each addressing one of the above challenges. Albeit simple, we are the rst to apply them for ecient depth estimation using heterogeneous dual cameras on commodity mobile devices. 1) *Iterative field-of-view cropping.* It iteratively crops the eld-of-view (FoV) of one camera, until it matches that of the other camera. As such, the dual cameras achieve the equivalent focal lengths. It improves the accuracy of epipolar rectication. 2) *Heterogeneous camera synchronization.* It lters out the frames that are generated at prominently dierent time. Moreover, it timely releases the metadata created by Android to avoid frequent garbage collection (GC). As such, the frame ows from the dual cameras are synchronized to avoid the displacements of moving objects across the frames in the same pair. 3) *Mobile GPU-friendly stereo matching.* We use SGBM for stereo matching due to its relatively high accuracy and eciency. However, it still cannot achieve real-time performance on mobile devices. Our insight is that the limited memory bandwidth of mobile GPU lowers the computation eciency. As such, we fuse the calculations in SGBM to reduce the overhead of accessing the global memory. Furthermore, we carefully enlarge the data slicing to decrease the number of concurrent threads. It reduces the access contention on the shared memory and the synchronization cost among threads.

Based on the techniques, we build the end-to-end MobiDepth system. MobiDepth rst crops the FoVs of the dual cameras and synchronizes the frame ows. Then, it runs the stereo matching to estimate the depth in a real-time manner. We implement MobiDepth on dominated mobile devices equipped with multiple cameras, and show the running demo in Figure 1(b). We also conduct extensive experiments under various object distances and motion conditions. Evaluation results show that MobiDepth achieves high performance in terms of both latency and accuracy. For example, it achieves real-time depth estimation of 22 frames per second (FPS) on the Huawei Mate40Pro, *i.e.,* an average latency of 45ms, with a small mean depth-estimation error of 1.1%~10.4% for stationary objects at the distance ranging from 0.5m to 5m, without requiring the motion of device. MobiDepth signicantly outperforms the learning-based depth models, *e.g.,* AnyNet [52] and MADNet [48], as well as the state-of-the-art depth estimation system ARCore [13]. For example, MobiDepth achieves a speedup of 1.66× and 12.13× compared to AnyNet and MADNet without online adaptation on Huawei Mate40Pro, respectively. With the motion of device, MobiDepth achieves a small mean error of 8.7% for the objects moving at a speed of 30-80 cm/s (at a distance of 100cm on the Huawei P30), while the mean error of ARCore is as high as 43.5% under the same settings. Note that ARCore does not work without the motion of device. Furthermore, we build an example application of 3D pose estimation based on MobiDepth and our application signicantly outperforms the state-of-the-art 3D pose estimation method, *i.e.,* MobileHumanPose [8], reducing the pose estimation latency and error by up to 57.1% and 29.5%, respectively.

In summary, the main contributions are as follows:

- Conduct in-depth analysis on the performance bottleneck of on-device dual camera-based depth estimation;
- Propose iterative FoV cropping and heterogeneous camera synchronization to achieve equivalent focal lengths and synchronized frame ows for the dual cameras, respectively;
- Propose the mobile GPU-friendly stereo matching based on SGBM, which signicantly reduces the memory access overhead and synchronization cost among threads;
- Implement the MobiDepth system and a MobiDepth-based 3D pose estimation application on commodity mobile devices to demonstrate the eectiveness of MobiDepth.

2 BACKGROUND AND CHALLENGES

We rst introduce the background of using dual cameras system to estimate depth in an ideal case. Then, we elaborate on the challenges of implementing such a dual camera system on commodity mobile devices.

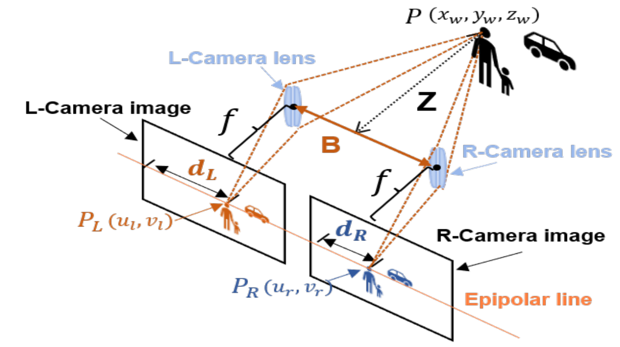


Figure 2: Illustration of the imaging process in an ideal binocular system.

2.1 Background

Depth estimation in a dual camera system. The dual cameras system simulates the principle of human vision for depth estimating in the 3D world, as shown in Figure 2.

The dual cameras are in the same orientation at dierent locations with distance B. To get the accurate depth Z of the point $P(x_w, y_w, z_w)$, there are several prerequisites: 1) the focal length f of the dual cameras should be equivalent. It guarantees that the cameras have the same FoV, so that the imaging points, *i.e.,* $P_L(u_l, v_l)$ and $P_R(u_r, v_r)$, lie on the same epipolar lines on the two image planes; 2) the dual cameras need to capture frames simultaneously; 3) the disparity $d_L - d_R$ between the two points P_L and P_R should be accurately estimated.

Given that the above conditions are met, the depth Z can be derived by Equation. 1:

$$Z = \frac{B \cdot f}{d_L - d_R} \tag{1}$$

The equation indicates that the depth can be readily derived based on the accurate disparity, *i.e.,* $d_L - d_R$, and equivalent focal length f for the dual cameras.

Semi-Global Block Matching (SGBM) SGBM is a commonly used computer vision algorithm in binocular camera systems for depth estimation. It compares the similarity of pixels in two images captured by binocular cameras to calculate the disparity of a pixel [19]. Figure 3 shows a block diagram of SGBM. It takes a pair of rectied images as input and outputs the disparity map. In specic, SGBM consists of four steps: 1) *Cost Computation.* It measures the similarity between the pixels to be matched and candidate pixels through three operations, *i.e.,* Census Transform [41], Hamming distance computation for cost of pixels [51], and Cost optimization with sliding window. 2) *Cost Aggregation.* It aggregates the cost values of pixels on the same row and column, respectively. 3) *Disparity Computation.* It uses the Winner-Takes-All algorithm [31] to determine the optimal disparity value for each pixel according to the aggregated cost. 4) *Disparity Refinement.* It renes the quality of the disparity map by ltering out the peaks.

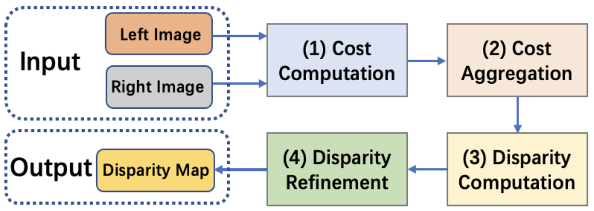


Figure 3: The process of SGBM. It takes a pair of rectified images as input, and outputs the disparity map. It consists of four steps, *i.e.,* Cost Computation, Cost Aggregation, Disparity Computation, and Disparity Refinement.

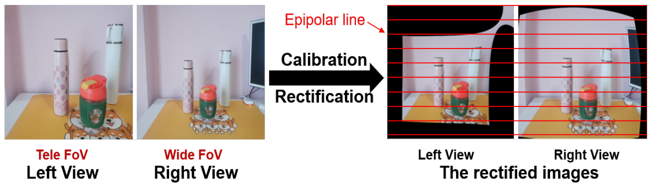


Figure 4: The raw images with different FoVs from dual cameras are on the left. Through the calibration and rectification, the images are severely distorted as shown on the right.

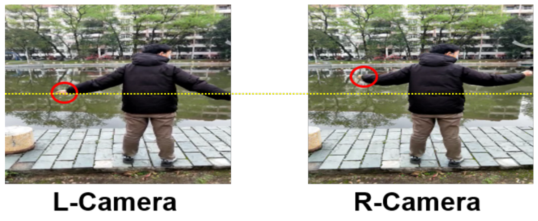


Figure 5: Example of frames out-of-sync. The same key-point, *i.e.,* left hand circled in red, is not aligned horizontally.

2.2 Design Challenges

However, using the dual cameras on commodity mobile devices to estimate accurate depth in real-time is diﬃcult. We expose the key issues that hinders the accurate and real-time on-device depth estimation: 1) diverse camera focal lengths; 2) out-of-sync frame ﬂows of the dual cameras; 3) computation heavy stereo matching.

The focal lengths of the dual cameras are quite diverse, leading to inaccurate epipolar rectification. The oﬀ-the-shelf commercial mobile devices have cameras with diﬀerent focal lengths. The focal length determines the camera's ﬁeld of view (FoV). For instance, a primary camera with short focal length has a wide FoV (WFoV). In contrast, the secondary camera with a long focal length has a tele FoV (TFoV). If we directly calibrate and rectify the captured images of the dual cameras, the considerable diﬀerence in focal lengths between the dual cameras could cause the epipolar rectiﬁcation[1] disastrously, as shown in the raw images in Figure 4.

As shown in Equation. 1, having equivalent focal lengths for the dual cameras is necessary for depth estimation. However, this requirement is not met on commodity mobile devices.

[1]Epipolar rectiﬁcation of a stereo pair is the process of re-sampling a pair of stereo images so that the apparent motion of corresponding points is horizontal, which is an important preliminary step in depth estimation [10].

Table 1: The Inference time of Stereo Matching Solutions on Huawei Mate40Pro (Kirin 9000 SoC).

Deep Learning (DL) Method			Traditional Method		
Method-DL[†]	Latency(s)	FPS	Method-Trad[‡]	Latency(s)	FPS
StereoNet [22]	3.23	0.31	AD-Census*[32]	1100	0.001
MADNet[◇] [48]	0.55	1.81	PMS*[6]	1500	0.0006
HITNet [44]	6.98	0.14	OpenCV-SGBM[19]	0.135	7.4

[†] These stereo matching models were tested by converting their open-source PyTorch model into a TFLite model.
[‡] The input resolution of these algorithms is 640×480, besides, the max disparity is 64 in SGBM.
[*] AD-Census and PMS are implemented by the open-source code from GitHub [1].
[◇] The latency of MADNet without online adaptation.

The frame flows from the on-device dual cameras are highly out-of-sync. Synchronization between the two captured frames of the dual cameras is crucial for the accuracy in depth estimation. As shown in Figure 5, when the two frames are captured at diﬀerent times, the same key-point, such as the left hand in the two frames, could move dozens of pixels between the two frames in scenes with moving, making epipolar rectiﬁcation extremely hard. The reason is that the frame periods, *i.e.,* time interval between consecutive frames, of the dual cameras are not exactly equal even if we set to use the same frame rate. For 30 frames per second (FPS), we ﬁnd that the average frame period of left camera is 33.33ms, while the one of the right camera is 33.31ms, *i.e.,* a 0.02ms time diﬀerence for every frame on average. Importantly, the diﬀerence accumulates as the system runs. It makes the frame ﬂows from the dual cameras highly out-of-sync. Moreover, we observe that frequent garbage collection (GC) could also make the frames out-of-sync.

State-of-the-art stereo matching solutions are computation heavy, leading to intolerable runtime latency. We test the latency of several state-of-the-art CNN models and traditional algorithms for stereo matching running on the CPU[2] of Huawei Mate40 Pro with Kirin 9000 SoC (System on a Chip). Table 1 shows the results that these stereo-matching solutions could not achieve real-time performance on mobile devices. Even with SGBM [19], it still takes $135ms$, limiting the frame rate to only $7FPS$. What's worse, the latency shown in Table 1 does not include the time for data copying and image rendering, which are also time consuming.

Next, we describe how MobiDepth addresses these challenges by inventing novel techniques.

3 MOBIDEPTH SYSTEM OVERVIEW

To enable dual-camera-based real-time depth estimation on mobile devices, we need to achieve three goals: 1) matching the views of the dual cameras with diﬀerent focal lengths; 2) synchronizing the frame ﬂows captured by the dual cameras; 3) boosting the stereo matching algorithm on mobile devices. We design the MobiDepth system to achieve the three goals. It consists of two phases, *i.e.,* the oﬄine phase and the online phase, as shown in Figure 6.

In the oﬄine phase, we design an iterative FoV cropping technique to determine how to match the FoVs of dual cameras (Section 4). It takes the several chessboard patterns as the input for

[2]We use CPU rather than the GPU because the algorithms/models either cannot run or run slower on the mobile GPU.

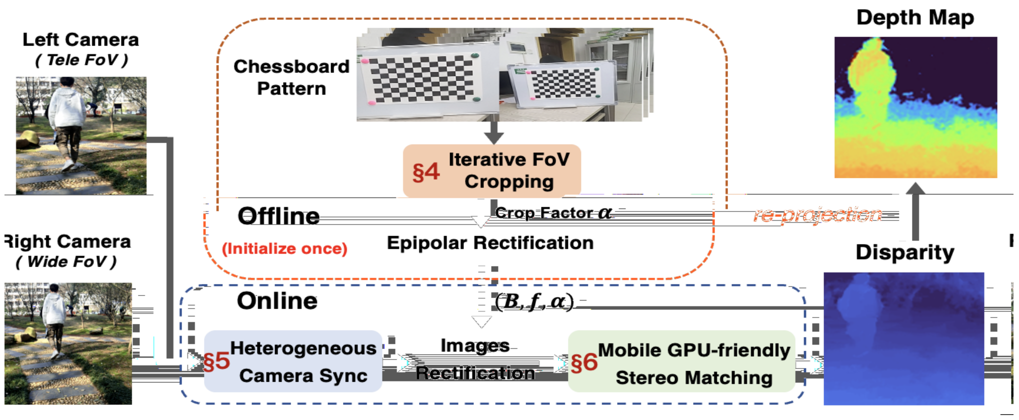


Figure 6: The system overview and workflow of MobiDepth.

camera calibration, and crops the WFoV iteratively until the focal lengths of the dual cameras are equivalent. Based on the WFoV cropped by factor $\bar{\alpha}$ and the TFoV, the following epipolar rectification module calculates the value of baseline B and focal length f. The offline phase runs only once during the initialization of the MobiDepth system.

In the online phase, we introduce a heterogeneous camera synchronization technique (Section 5) to align the frame flows from the dual cameras, such that the time difference between two frames from dual cameras does not exceed a certain threshold. The image rectification module takes a pair of cropped and synchronized images as input. It determines the correspondence between the epipolar lines between the pair of images, based on the parameters obtained in the offline phase. After that, the mobile GPU-friendly stereo matching technique (Section 6) efficiently finds the correspondence points on the epipolar lines to calculate the disparity. The final depth map can be readily estimated based on the disparity, baseline, and focal length.

The details of the proposed techniques can be found in the following sections.

4 ITERATIVE FOV CROPPING

The focal lengths of the dual cameras on mobile devices are usually quite diverse, leading to different FoVs of the dual cameras. Generally, device manufacturers provide the equivalent focal length of each camera. Ideally, we can calculate the crop factor to make the FoV of the dual cameras equal. Yet, the provided equivalent focal length is not accurate enough. For example, the Honor V30Pro officially provides an equivalent focal length equal to 16mm while the measured value is close to 17mm. Cropping the images using the provided equivalent focal length leads to significant error. In addition, the existing approaches typically use image matching algorithms to crop the FoVs, such as SIFT [30], SURF [5], ORB [36]. However, the accuracy of these approaches is not satisfactory since they only perform a homography transformation, with the experimental results shown in Section 8.3.1. In the following, we first analyze the feasibility of finding the equivalent focal lengths through cropping the FoVs. Then we iteratively crop the FoVs until the focal lengths are equivalent for the dual cameras.

FoV cropping analysis. According to the lens imaging rule [35], the field-of-view (FoV) of a camera is determined by its focal length. Specifically, the camera with a smaller focal lengths f^W has a $WFoV$, and the camera with a longer focal length f^T has a $TFoV$, as shown in Figure 7. By cropping the WFoV to be equal to the TFoV, we can make the focal lengths of the dual cameras both equal to f^T. As such, it is a fact that the same FoVs brings the equivalent focal lengths for the dual cameras.

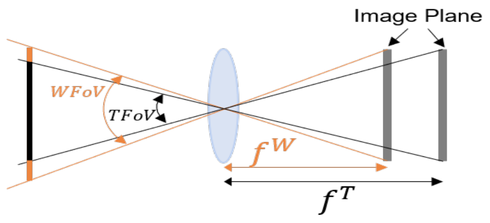


Figure 7: The relationship between focal length and FoV size.

Based on the fact, MobiDepth propose the *iterative FoVs cropping* to iteratively crop the WFoV, until the focal lengths of the dual cameras are equivalent. Since the image captured by the camera is not square, *i.e.*, the size of image is 640×480, we use $\bar{\alpha} = (\alpha_x, \alpha_y)$ to denote the crop factor on the width and height, respectively. We formulate Equation 2 as the loss function which quantifies the difference of focal lengths between the dual cameras.

$$J(\bar{\alpha}) = \frac{1}{2}(f^W_{\bar{\alpha}^i} - f^T)^2 \tag{2}$$

where f^T denotes the focal length of the TFoV camera. We define $f^W_{\bar{\alpha}^i}$ as the focal length of the WFoV camera after i rounds of cropping. The optimal crop factor can be found by iteratively minimizing the loss.

Figure 8 illustrates the workflow of iterative FoV cropping: 1) First, we initialize the crop factor $\bar{\alpha}^0$ to 1, and get the focal lengths of the dual cameras with Zhang's calibration method [58] [3], *i.e.*, f^T and $f^W_{\bar{\alpha}^0}$. To obtain more accurate values, we capture over 15 images of the calibration pattern (*i.e.*, the chessboard) at different

[3]Zhang's method is a camera calibration method that uses calibration pattern, *e.g.*, chessboard, to get the camera parameters, such as focal lengths, lens distortion coefficient, etc.

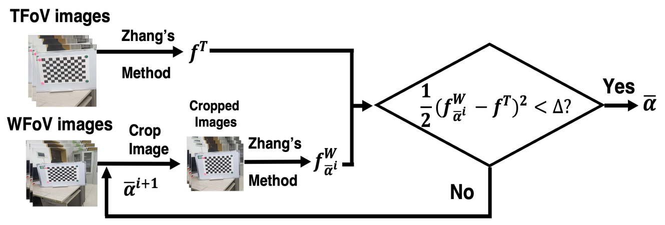


Figure 8: The process of the Iterative FoVs Cropping technique.

positions and filter out the image pairs with re-projection error over ϕ (ϕ is set to 0.05 in our implementation). 2) Then, we crop the images with WFoV from all sides according to $\bar{\alpha}^i$ with the center of the FoV as the basis point, then calibrate the camera after cropping to get the updated focal length $f^W_{\bar{\alpha}^i}$. 3) Next, we update the value of $\bar{\alpha}^i$ to $\bar{\alpha}^{i+1}$ using the following rule: if $f^W_{\bar{\alpha}^i} - f^T > 0$, then $\bar{\alpha}^{i+1} = \bar{\alpha}^i - step$. Otherwise, $\bar{\alpha}^{i+1} = \bar{\alpha}^i + step$, where $step$ is set to 0.01 in our implementation. 4) In case the value of Equation 2 becomes less than Δ (0.01 according to our evaluation) for five consecutive rounds, we obtain the final crop factor $\bar{\alpha}$, based on which the focal lengths of the dual cameras are equivalent.

5 HETEROGENEOUS CAMERA SYNCHRONIZATION

The accurate depth estimation of objects highly relies on the synchronization of frame flows of the heterogeneous dual cameras. In case of out-of-sync flows, the target object will move a dozen of pixels between the pair of rectified images, leading to a disastrous performance of stereo matching.

To obtain time-aligned frames, previous works often use additional external hardware signals to simultaneously trigger cameras to capture images [21, 40]. However, there is no such dedicated hardware to trigger cameras on most mobile devices. As discussed in Section 2.2, the actual frame rates of the dual cameras are slightly different even under the same setting. The time difference of the frames captured by the heterogeneous cameras would accumulate at runtime if they are not carefully synchronized.

In this section, we introduce a simple yet effective technique to reduce the time difference between the frames from two heterogeneous cameras to under a threshold. Figure 9 illustrates the process of the technique. T_L and T_R denote the frame periods of the left camera and the right camera, respectively. The small circles in the figure represent frames, and the number above the circle represents the order of the frames in the frame flow. The camera synchronization works as follows based on the recorded timestamp of each frame captured by both cameras: 1) Compare the timestamps of two frames with the same order in the frame flows. 2) If the difference between the timestamps of a frame pair is lower than the threshold θ, regard the two frames as a matched pair, denoted by the solid line in Figure 9. The matched frame pair is used as the input in the next step (image rectification). 3) If the time difference exceeds the threshold θ, discard the frame in the faster flow and goes to step 1 to compare its next frame with the frame in the other flow. For example, in Figure 9, the $n-th$ frames in the two flows are not

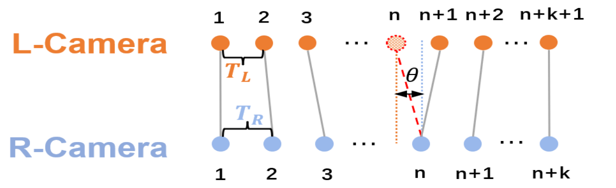


Figure 9: The illustration of heterogeneous camera synchronization.

Table 2: The latency of each step of SGBM using existing parallel optimization strategy on the GPU of Huawei Mate40Pro (Kirin 9000 SoC).

Steps	GPU(ms)
Cost Computation	30
Cost Aggregation	150
Disparity Computation	10
Disparity Refinement	9
Total latency	199

matched, so the $n-th$ frame from the left camera is discarded, and the $(n+1)-th$ frame in the left camera are compared with $n-th$ frame in the right camera.

By fine tuning θ, we have a trade-off between the synchronization of frame flows and the loss of frames. With a larger value of θ, we can retain more frames, but the frames may have larger time difference and lead to inaccurate depth estimation due to object displacement, and vice versa. The value of θ should be determined according to the application scenario. For example, in a scenario where the target object is mostly static, we can use a larger θ to tolerate frames that are slightly out-of-sync. While if the target objects move fast, the value of θ should be smaller to ensure higher accuracy.

In addition, when implementing the synchronization technique, we observed that retrieving frame flows from dual cameras would frequently trigger Android garbage collection (GC) events, which caused the frame periods of the dual cameras to fluctuate. By tracing the memory usage of MobiDepth, we found that the frequent GCs were due to the un-recycled metadata, such as the `CameraMetadata` objects which contains the settings of the camera [14]. After running for a while, the un-recycled metadata would take all the pre-allocated memory. To solve this issue, we used the Java reflection method to timely release the metadata once it is no longer useful.

6 MOBILE GPU-FRIENDLY STEREO MATCHING

We use SGBM as the stereo matching algorithm in MobiDepth due to its relatively low latency (*e.g.,* 135ms on the CPU of the Kirin 9000 SoC) and satisfactory accuracy (*e.g.,* 4.41% erroneous pixels in total with error threshold as 5 pixels on KITTI 2012 [16]).

Although SGBM is efficient, it still cannot achieve real-time performance on mobile devices. Using GPU may be a promising direction to further optimize its performance. However, existing works of SGBM acceleration are for desktop GPUs [4, 18]. We adopt the similar optimizations to implement SGBM on mobile GPU, but the performance is even worse than running on the CPU, as shown in Table 2. This is due to the following limitations of mobile GPUs.

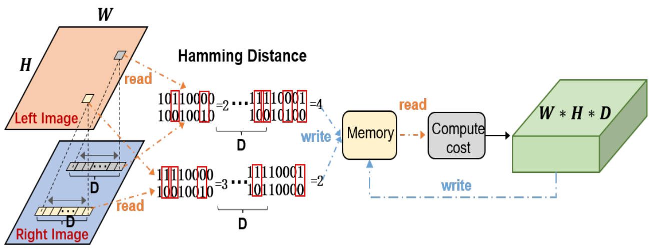


Figure 10: The process of Cost Computation in SGBM.

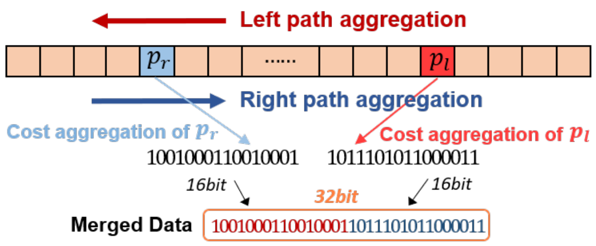


Figure 11: The process of data merged memory write in Cost Aggregation (Take left-right aggregation as an example).

Limited memory bandwidth. The most time-consuming steps in SGBM are Cost Computation and Cost Aggregation since they require frequent memory read/write operations to calculate the disparity of each pixel. However, the memory bandwidth of mobile GPUs is quite limited compared with desktop GPUs [27, 55]. For instance, the Mali-G78 GPU in the Kirin 9000 SoC has 25.98 GB/s memory bandwidth shared with the CPU, which is only $\frac{1}{24}$ of the memory bandwidth of the NVIDIA RTX 2080Ti GPU (*i.e.,* 616 GB/s). As a result, the frequent memory read/write operations in SGBM lead to a long latency on mobile GPUs.

Limited memory architecture support. In Cost Aggregation of SGBM, we need to use thread synchronization to get the minimum aggregated cost of a pixel with all disparities. In the desktop GPU, thread synchronization is performed in the fast on-chip shared memory, while mobile GPUs such as the Mali GPU only has o-chip shared memory which is much slower.

To tackle the challenges, we propose multiple techniques, including *calculation fusion* and *data merged memory write* to reduce the cost of memory read and write in SGBM, and *enlarge data slicing* to reduce the overhead of thread synchronization.

6.1 Reducing the Memory Read and Write Overhead

Memory read and write overhead analysis. In the SGBM algorithm, each step utilizes the results of the previous step, which incurs a large amount of read and write operations to the global memory. 1) In Cost Computation, after Census Transform converts each pixel of the pair of stereo images to a binary string, SGBM calculates the Hamming distance between each pixel in the left image and the corresponding disparity range pixel in the right image and writes the data to the memory, and then reads the data out for cost computation with sliding window. Figure 10 shows the original process of Cost Computation step. We set the image resolution as $W \times H$ and the max disparity range is D. Calculating the Hamming distance for one disparity requires reading two pixels from the memory and writing one result back into memory, which requires $W \times H \times D \times 3$ times for memory read and write. After the calculation of the Hamming distance of all pixels, SGBM reads two Hamming distance results of the two disparities from the memory, and input them to cost computation operation to derive the disparity cost. Thus, the times of memory read and write to obtain the whole image's disparity cost is $W \times H \times D \times (3 + 3)$. 2) Cost Aggregation adopts four paths aggregation, *i.e.,* left-right, up-down. The existing approach to calculate pixels' aggregation cost is to compute the results for each path individually, which also causes considerable amount of memory read and write operations. 3) In Cost Aggregation, SGBM needs to get the aggregation cost of each pixel in each of the four paths. It requires to write these values back to the memory which requires $2 \times H \times D + 2 \times W \times D$ times writing.

Calculation fusion. To reduce memory read and write overhead, we fuse some calculations in Cost Computation and Cost Aggregation. In Cost Computation, we compute the two Hamming distances with four pixels each and then use the results directly to get the disparity cost. In this case, the times of memory read and write is $W \times H \times D \times (4 + 1)$, which reduces $W \times H \times D$ times compared with the original approach. In Cost Aggregation, the left and right aggregations operate on the same data. Therefore, they can be done simultaneously. For instance, we may calculate the I_dth pixel in the left aggregation and the I_{W-1-d}th pixel in the right aggregation at the same time. In this way, we could reduce the times of memory accesses by $W * D + H * D$. This is the same for the up and down aggregations.

Data merged memory write. Merging data before writing back to memory can further reduce memory access overhead. In Cost Aggregation, we combine the two values obtained from the left and right aggregations, or the up and down aggregations, into one array for memory write. As such, the times of memory write can be reduced by half. As shown in Figure 11, we combine the two 16-bit values obtained from left and right aggregation into a 32-bit value. It is written into memory only once.

6.2 Reduced Data Synchronization Overhead

In Cost Aggregation, one of the critical steps is to calculate the minimal aggregated cost for each pixel which requires all the disparities of each pixel to be calculated and synchronized. The existing implementation of SGBM typically creates a number of threads equal to the maximum number of disparity(D) to calculate the minimum value of all aggregated costs for a pixel. However, it leads to extensive synchronization cost and contention on memory access. The synchronization is done in on-chip shared memory in desktop GPU with low execution time, while in the mobile GPUs, *i.e.,* Mali GPUs, this operation is executed in o-chip shared memory, which is time-consuming.

Enlarged data slicing. To address this issue, we take 2^n (we set n=3 according to our evaluation) disparities data at once by the vectorized memory access and then put them on a single thread for processing, as shown in Figure 12. By our enlarged data slicing method, each thread calculates 2^n disparity values and gets the minimum disparity cost on that thread. Therefore, we only need

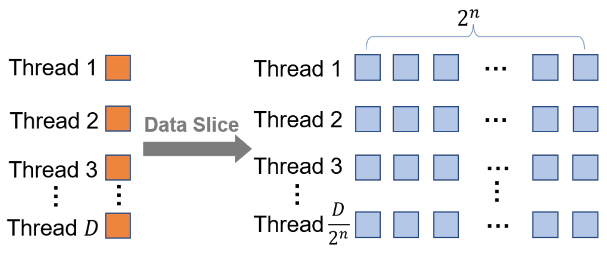


Figure 12: The schematic diagram of data slicing.

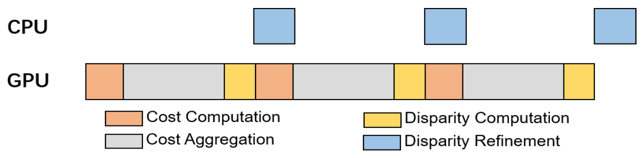


Figure 13: The latency hiding in our SGBM algorithm.

to synchronize $\frac{D}{2^n}$ threads in the mobile shared memory to get the minimum value of all disparities, which significantly reduces the data synchronization cost on mobile GPUs. Furthermore, using our method could also reduce the times of shared memory read and write. We read 2^n disparity values into memory with vectorized memory access only once, which decreases memory read times by $D - \frac{D}{2^n}$, and we only need to write $\frac{D}{2^n}$ data to shared memory to get the minimum disparity cost. Besides, we increase the amount of data on each thread, improving the data throughput and bandwidth utilization.

Furthermore, the Cost Computation of one frame can start right after the accomplishment of the disparity computation of the last frame. To further reduce the latency of SGBM on mobile devices, we allocate the Disparity Refinement of the frame onto the CPU, as shown in Figure 13. The other three steps, *i.e.*, Cost Computation, Cost Aggregation, and Disparity Computation, still run on mobile GPU.

7 IMPLEMENTATION

We implement MobiDepth on commodity Android mobile devices in Java [3] and C++ [43] with 5,457 lines of code excluding the testing code, counted by Android Studio Statistic tool. Specifically, we use the official *multi-camera API* [14] to obtain the images captured by each camera at 30 FPS.

To obtain the focal length of camera, we adopt the calibration tool in OpenCV [9] which is based Zhang's method [58] to obtain the parameter. With the method, we capture 20 images of the 12×9 chessboard pattern placed on a plate with a size of $20mm \times 20mm$, and then we filter out the image pairs with reprojection error exceeding 0.2 pixels to improve the calibration accuracy. After we align the FoVs of the dual cameras based on the iterative FoV cropping method, we pass the processed pairs of images to the `cvStereoRectify` function in OpenCV for epipolar rectification.

We implement the SGBM in MobiDepth on mobile GPUs with OpenCL 2.0 [42], which is supported by most mobile device systems. We use the `vloadn/vstoren` functions to vectorize the read buffer format to improve bandwidth utilization, and use the `select` function to reduce branch operation. To ensure stable performance of our system, we set the input image resolution of the SGBM algorithm to 640×480, and the disparity level to 64, and use four path directions for cost aggregation.

In addition, we implement an example 3D pose-estimation application based on MobiDepth. We use a lightweight 2D pose-estimation method [57] based on TensorFlow Lite [28] to get the 2D coordinates of human keypoints. Then, we combine the coordinates of each keypoint and the depth of the corresponding position to form the 3D coordinates.

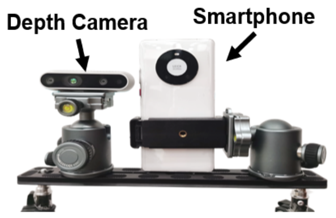


Figure 14: The hardware platform used to evaluate MobiDepth. It aligns the lenses horizontally to guarantee the accuracy of the evaluation.

Table 3: Hardware configurations of the mobile devices used in the experiments.

Device	SoC	CPU	GPU
HWM40P*	Kirin 9000	Cortex-A77	Mali G78
HuaweiP30(HWP30)	Kirin 980	Cortex-A76	Mali G76
Google Pixel6Pro	Google Tensor	Cortex-X1	Mali G78

* HWM40P is the abbreviation of Huawei Mate40 Pro.

8 EVALUATION

In this section, we evaluate the overall performance of MobiDepth and its key components. We also evaluate the 3D pose-estimation application and the system overhead of MobiDepth, including the latency in mid-low end mobile device, the power consumption, and the memory usage.

8.1 Experimental Setup

Hardware platform. We built a hardware platform to evaluate the depth estimation performance of MobiDepth. As shown in Figure 14, we placed a mobile device and a depth camera (Intel RealSense D435i) horizontally on a bracket, facing the same active area. We install the MobiDepth and baseline solutions on the mobile device to calculate the depth of objects in the active area, and the depth camera was used to produce the ground-truth depth values. We tested three mobile devices that covered different mobile SoCs and diverse computing capabilities, as shown in Table 3.

Operating conditions. We considered a wide range of operating conditions to evaluate the depth estimation systems, including the situations when the mobile device is stationary or moving and the target object is stationary or moving at different distances. Specifically, we considered different statuses of the target object, including stationary, moving slowly, and moving quickly. For each object status, we considered different distances of the target object, including 50cm, 100cm, 300cm, and 500cm. The target object is a box with a plate surface for the convenience of accuracy computation.

We also included the cases where the depth estimation device is stationary or moving.

Baselines. The primary baselines which we compared MobiDepth against are AnyNet [52] and MADNet with and without online adaptation[4] [48], which are the state-of-the-art depth estimation model on mobile devices, and ARCore [13], which is the official framework for depth estimation on Android. For simplicity, we name MADNet with and without online adaption as MADNet-MAD and MADNet-No, respectively. We also considered several other baselines for evaluating different components of MobiDepth. For example, we compared with the SIFT method [30] on the performance of FoV matching. When evaluating the example application (mobile 3D pose estimation) based on MobiDepth, we selected a SOTA 3D human pose estimation model, MobileHumanPose [8], as the baseline.

Metrics. We evaluated the performance of MobiDepth in terms of accuracy, latency, energy cost, and memory usage. For depth-estimation accuracy, we used the *mean distance error* (mD_{err}), the average percentage error between the estimated depth of each pixel p in the target object ($Depth_{est}$) and the groundtruth depth ($Depth_{gt}$), as follows:

$$mD_{err} = mean_{p \in Object} \frac{|Depth_{gt}(p) - Depth_{est}(p)|}{Depth_{gt}(p)} \qquad (3)$$

When evaluating the example application, we computed the mean per-joint position error (MPJPE), which is the mean Euclidean distance between the positions of predicted joints and the ground truth.

8.2 Overall Depth Estimation Accuracy

We first evaluate the end-to-end performance of MobiDepth on depth estimation. Table 4 shows the accuracy of MobiDepth, AnyNet, MADNet and ARCore on different mobile devices. Among the three devices, HWP30 supports all methods. ARCore is not supported on HWM40P since it cannot run Google play services for AR. As for the third-party application, MobiDepth is not authorized to access the libopencl.so library in the Pixel series phones, since Google does not publicly support the OpenCL library. Therefore, we only evaluate the performance of ARCore on Google Pixel6Pro.

As shown in Table 4, MobiDepth outperforms the deep learning model baselines, *i.e.,* AnyNet, MADNet-MAD and MADNet-No, in most cases. For instance, when the device and the target object are stationary, the estimation error of MobiDepth is 2.8%, 1.1%, 5.1% and 10.4% at various distances, and yet, AnyNet achieves 3.5%, 2.4%, 6.5%, 12.0% and MADNet-No is 10.6%, 12.4%, 47.6%, 61.5% at the corresponding distances. Only on HWP30 with object at distance 50 cm, the estimation error of MobiDepth is slightly higher than AnyNet, *i.e.,* 3.8 % compared to 3.4%. One reason for the higher error of deep learning based models is that the pair of images for model training has been perfectly rectified. However, the rectification of images captured by the on-device dual cameras suffers from inevitable error due to the diverse setting of cameras, *e.g.,* a normal lens and wide-angle lens. Another reason is from the limited scenes in the training dataset. In comparison, MobiDepth tolerates slight image rectification error as the SGBM uses block matching method to calculate the disparity. More importantly, MobiDepth does not require any pre-training. It not only enables MobiDepth to work in new scenes, but also saves lots of human efforts in collecting and labeling a training dataset.

From Table 4, MobiDepth also always outperforms ARCore. For example, when the two systems are tested under the exact same condition (moving device and stationary target) on HWP30, MobiDepth achieves average accuracy of 5.0%, 4.2%, 8.9% and 23.2% at different distances, while the accuracy of ARCore is 14.3%, 7.1%, 11.7% and 25.5% respectively.

The greater advantage of MobiDepth is found when the target object is moving. Due to the algorithmic basis of ARCore, it cannot work well on moving targets, because the motion of the target object will harm the relative displacement between frames measured by IMU. However, it is not an issue in MobiDepth since our depth is estimated in a per-frame manner. As a result, we can observe a significantly superior performance of MobiDepth (*e.g.,* 8.7% vs. 43.5% on HWP30 at distance=100cm) when the target is moving slowly ($30 - 80cm/s$).

In addition, the accuracy of MobiDepth is even better when the device is stationary. For example, on HWM40P, the accuracy with the device stationary and moving are 1.1% and 3.2% respectively for stationary target at distance=100cm. On the contrary, ARCore is unable to obtain depth information without motion, which may limit its usage scenarios such as live streaming with a fixed mobile device camera.

The accuracy of MobiDepth may be affected by the device. We can notice that the performance of MobiDepth on HWM40P is better than others, as the quality of cameras and the distribution of dual cameras on HWM40P are better. The performance of MobiDepth may get worse if the secondary camera on the device is too poor or lies too close to the main camera. Nevertheless, MobiDepth can still achieve a reasonable accuracy on most devices since most mobile devices today are equipped with powerful dual cameras.

8.3 Performance Breakdown

We next evaluate the performance of the key components of the MobiDepth system in detail.

8.3.1 *FoV Matching.*

As introduced in Section 4, MobiDepth uses an iterative FOV cropping method to match the FoVs of dual cameras. An alternative to our method is the SIFT-based method, and thus we tested the performance of MobiDepth if the FoV matching part is implemented with SIFT. The result is shown in Table 5. We can see that the SIFT baseline can lead to higher mean distance errors than our original method on depth estimation, which are 7.3% vs. 2.8%, 6.6% vs. 1.1%, 8.2% vs. 5.1%, 13.3% vs. 10.4% when the target object distance is 50cm, 100cm, 300cm and 500cm, respectively. This demonstrates the effectiveness of our iterative cropping-based FoV matching method.

8.3.2 *Frame Synchronization.*

We introduced a frame synchronization technique (Section 5) to reduce the time difference between the image streams of dual cameras. Figure 15 shows that the time difference of two frames accumulates as the time goes without the frame synchronization. The time difference exceeds 80ms after about 4000 frames. With our frame alignment method, we can keep the time difference of the dual cameras within 16ms (as we set the

[4] We use an average of 1280 frames for online adaptation.

Table 4: The accuracy of MobiDepth, AnyNet, MADNet and ARCore on depth estimation under different operating conditions. Each cell is the average *mean distance error* (mD_{err}).

Methods \ Object status			Obj-Stationary†				Obj-Moving†(30~80cm/s)				Obj-Moving (80~150cm/s)			
			50cm	100cm	300cm	500cm	50cm	100cm	300cm	500cm	50cm	100cm	300cm	500cm
HWM40P	AnyNet	Dev-Stationary‡	3.5%	2.4%	6.5%	12.0%	4.0%	5.2%	11.1%	21.4%	13.9%	19.8%	20.4%	31.4%
	MADNet-No	Dev-Stationary	10.6%	12.4%	47.8%	61.7%	13.5%	19.2%	50.3%	60.9%	33.3%	28.8%	55.8%	64.9%
	MADNet-MAD	Dev-Stationary	5.8%	4.9%	11.5%	15.3%	5.9%	8.8%	19.6%	23.7%	14.7%	15.8%	28.3%	38.7%
	MobiDepth	Dev-Stationary	2.8%	1.1%	5.1%	10.4%	3.8%	3.1%	9.3%	19.2%	9.8%	9.5%	15.1%	29.9%
		Dev-Moving‡	4.5%	3.2%	7.4%	14.3%	5.6%	5.8%	13.4%	23.1%	13.3%	15.7%	23.4%	31.1%
HWP30	AnyNet	Dev-Stationary	3.4%	5.2%	13.3%	25.5%	4.4%	7.6%	23.8%	30.9%	11.1%	16.0%	28.3%	39.3%
	MADNet-No	Dev-Stationary	18.8%	15.9%	58.9%	74.4%	26.8%	16.8%	62.7%	74.6%	24.5%	17.6%	68.9%	74.6%
	MADNet-MAD	Dev-Stationary	8.4%	9.1%	26.2%	49.6%	9.4%	10.9%	30.6%	54.8%	14.7%	13.1%	47.4%	54.7%
	MobiDepth	Dev-Stationary	3.8%	4.1%	7.1%	21.4%	4.3%	6.3%	14.2%	28.5%	10.1%	13.7%	21.5%	35.6%
		Dev-Moving	5.0%	4.2%	8.9%	23.2%	7.4%	8.7%	24.4%	35.2%	15.3%	20.1%	30.5%	40.3%
	ARCore*	Dev-Moving	14.3%	7.1%	11.7%	25.5%	56.6%	43.5%	54.8%	57.9%	69.9%	66.6%	58.7%	61.9%

* Since ARCore cannot obtain depth information while the device is stationary, we only tested ARCore with the device moving.
† Obj-Stationary and Obj-Moving denote that the status of target object is static and moving, respectively.
‡ Dev-Stationary and Dev-Moving indicate that the state of mobile device is static and moving, respectively.

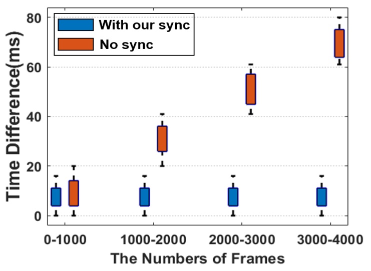


Figure 15: Time difference with and without frame synchronization.

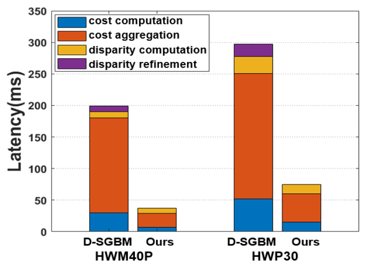


Figure 16: Latency reduction of our optimized SGBM for mobile GPU.

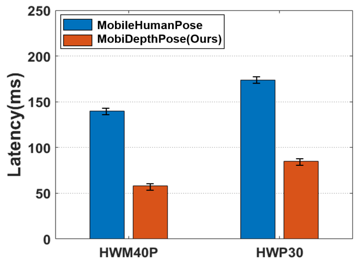


Figure 17: Latency of MobiDepthPose and MobileHumanPose.

Table 5: The average *mean distance error* (mD_{err}) of SIFT and our method for FoV matching. The target object is stationary at different distances.

Distance	Methods	
	MobiDepth-SIFT	MobiDepth-Ours
50cm	7.3%	2.8%
100cm	6.6%	1.1%
300cm	8.2%	5.1%
500cm	13.3%	10.4%

θ to 16ms, the half of the frame period). To test the effectiveness of this technique, we compared the accuracy of MobiDepth with and without frame synchronization, as shown in Table 6.

In Table 6, Unsync represents the cases when the two frames used for stereo matching in MobiDepth are not synchronized (by one frame or two frames). As can be seen from the table, our frame synchronization method allows the MobiDepth to achieve obviously higher accuracy than without synchronization. The advantage of using frame synchronization is more significant when the velocity of the target object is higher.

8.3.3 Stereo Matching Optimizations. In the stereo matching of MobiDepth, we adopted several techniques to optimize the performance of SGBM on mobile devices. Figure 16 shows the latency reduction of our customized SGBM implementation, compared to the existing SGBM implementation that was originally developed for desktop GPU (denoted as *D-SGBM*). The result shows that almost all parts of the SGBM algorithm are significantly optimized, reducing the latency of SGBM by 81.4% on HWM40P and 72.8% on HWP30.

Table 6: The average *mean distance error* (mD_{err}) of MobiDepth when the two cameras are synchronized (Sync) or not synchronized (Unsync). Unsync1 and Unsync2 have a 33ms (one frame) and 66ms (two frames) time difference between two frames, respectively. The target object is moving at different distances and speeds.

Distance \ Speed	Moving (30~80cm/s)		
	Original	Unsync1	Unsync2
50cm	3.8%	17.2%	41.2%
100cm	3.1%	12.1%	31.1%
300cm	9.3%	25.6%	42.2%
500cm	19.2%	36.3%	50.7%

8.4 Case Study: 3D Pose Estimation

MobiDepth may enable various 3D applications on mobile devices. We take 3D pose estimation (PE) as an example case study to show

Table 7: Accuracy of MobileHumanPose and MobiDepthPose in terms of MPJPE (mm).

Methods	Walk	Kick	Sit	Hit	Avg.
MobileHumanPose [8]	290.3	231.4	213.8	229.9	241.4
MobiDepthPose (Ours)	195.3	164.8	141.6	178.5	170.1

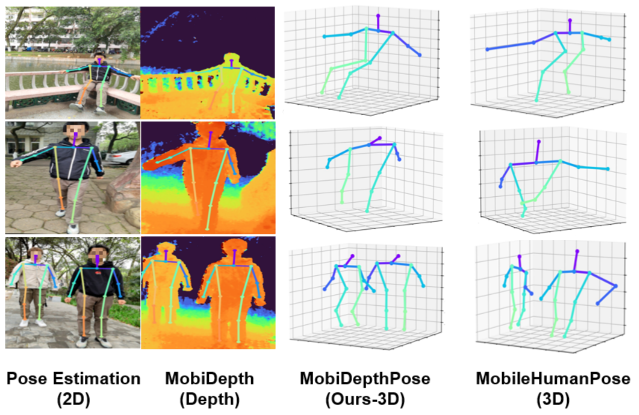


Figure 18: Visualization of some 3D poses predicted by MobiDepthPose and MobileHumanPose.

the effectiveness of MobiDepth. To do it, we implement a mobile 3D PE application based on MobiDepth, named MobiDepthPose. The current SOTA mobile 3D PE system MobileHumanPose [8] adopts an end-to-end neural network to predict the 3D keypoint coordinates (x, y, z) of human joints. Instead, our MobiDepthPose can directly utilize MobiDepth to obtain accurate depth information, so that the model only needs to predicts the 2D keypoint coordinates (x,y). In MobiDepthPose, we use an accurate and lightweight 2D PE model [56, 57] in combination with MobiDepth to obtain the 3D keypoint coordinates (x, y, z).

We evaluated the accuracy of MobileHumanPose and MobiDepthPose under different settings when a person was walking, sitting, kicking, or hitting something in the camera view. In each setting, we collected 300 samples, and the ground truth was produced by the combination of a full-featured depth camera (Intel RealSense D435i) and a SOTA 2D PE model (HigherHRNet-w48 [7]). As we can see from Table 7, MobiDepthPose is able to achieve a much better accuracy than MobileHumanPose in all cases, with the overall mean per joint position error (MPJPE) significantly reduced from 241.4mm to 170.1mm, *i.e.,* a reduction of 29.5%.

Moreover, since the neural network can be much simplified in MobiDepthPose, the latency of 3D PE can be reduced as well. As shown in Figure 17, the latency of MobiDepthPose is reduced by 45.4% to 57.1% compared with MobileHumanPose on different devices. Some examples of the estimated 3D pose can be found in Figure 18.

8.5 System Overhead

Finally, we evaluate the system overhead of MobiDepth in terms of latency, power consumption, and memory usage. Figure 19(a) shows the latency of MobiDepth, AnyNet, MADNet-No and ARCore on the three mobile devices[5]. Compared to AnyNet and MADNet-No, MobiDepth can reduce latency by 40.02% and 91.81% on HWM40P, and by 24.04% and 88.34% on HWP30, respectively. Yet, as the official framework adopted by Android, ARCore achieves better latency than MobiDepth on both high-end and low-end devices. Especially, the latency gap between ARCore and MobiDepth is large (about 40ms) on HWP30. This is because that ARCore utilizes keyframes and IMU to reduce the computation, while MobiDepth computes depth in a per-frame manner. Nevertheless, the result has also shown that the latency of MobiDepth is greatly reduced to around 45ms on higher-end devices (*e.g.,* HWM40P), which shows the ability of MobiDepth to achieve real-time experience.

The results of power consumption are shown in Figure 19(b). We use PerfDog [34] developed by Tencent to collect the power consumption of different systems. Since MobiDepth exploits both the CPU, GPU and the two cameras, its average power consumption is about 50% higher than ARCore. However, MobiDepth can reduce 7.14% and 8.08% power consumption than AnyNet and MADNet-No, respectively on HWM40P. This is because the most operations of MobiDepth are simple additions and comparisons, which have a low utilization of the ALU on GPU, *i.e.,* 4% to 6% according to our test.

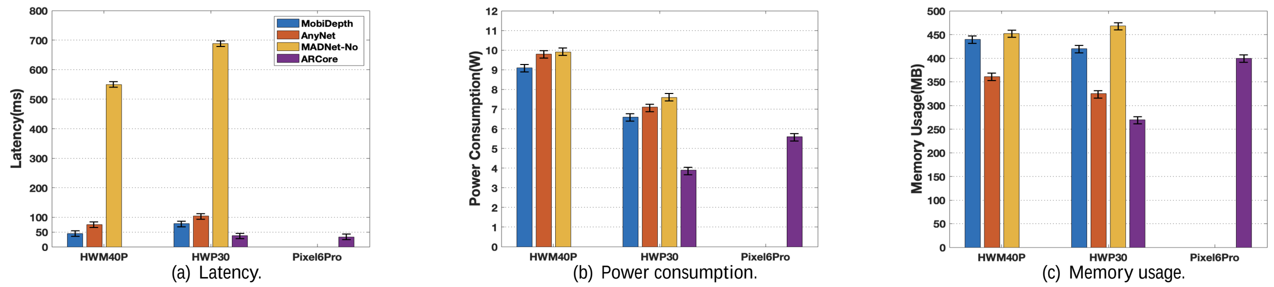
The memory usage of MobiDepth, AnyNet, MADNet-No and ARCore are obtained using the *Monitors* tool in Android Studio. As shown in Figure 19(c), the average memory usage of MobiDepth exceeds AnyNet's by 17.9% and ARCore's by 35%, since it computes the disparity of each pixel on both CPU and GPU simultaneously. However, the total memory usage of MobiDepth is less than 450MB, which is relatively small as mainstream mobile devices have multi-gigabytes memory.

To sum up, MobiDepth introduces a higher overhead than ARCore. However, it reduces the latency and power, compared with the AnyNet and MADNet. We believe that the overhead is tolerable given the great advantages offered by MobiDepth.

9 RELATED WORK

Stereo matching. To achieve the accurate depth, many works have focused on stereo matching algorithm [17, 47]. Traditional stereo matching methods usually utilize the low-level features of image patches around the pixel to measure the dissimilarity, which can be grouped into three categories: 1) Local method. Both Lazaros et al. [33] and Kristina et al. [2] exploit the region-based local algorithms, which is based on feature vectors extracted in a window for matching. 2) Global method. Vladimir et al. [26] and Andreas et al. [24] select the disparity with the minimal global energy function. 3) Semi-global method. Heiko [19] optimizes a path-wise form of the energy function in multiple direction. With the development of deep learning, many stereo matching works use CNN models to improve the accuracy of depth estimation [25, 38, 44]. Akihito et al. [38] propose the SGM-Nets model to improve the accuracy of the model by providing a learning penalty to the SGM. Patrick et al. [25] learns smoothness penalties through a conditional random field(CRF) and combines it with a correlation matching cost

[5]We do not evaluate the latency of MADNet-MAD on mobile devices since the model with the online adaptation module is extremely time consuming, which its latency is much higher than that of MADNet-No.



(a) Latency. (b) Power consumption. (c) Memory usage.

Figure 19: System overhead of MobiDepth, AnyNet, MADNet-No and ARCore.

predicted by CNN to integrate long-range interactions. Vladimir et al. [44] design a CNN model that propagate the information across different resolution levels. However, these works are only suitable for identical dual cameras. Besides, none of these works consider the specific architecture of mobile SoCs. MobiDepth aims to estimate depth through a dual-camera system with diverse settings, and well use the memory architecture of mobile GPU for acceleration.

Depth estimation on mobile devices. Current methods of getting depth on mobile devices can be divided into 3 categories: 1) Dedicated depth sensors. Kim et al. [23] and Tian et al. [46] use the on-device time-of-flight(ToF) camera to obtain 3D images, and Shih et al. [39] and Stefano et al. [45] use LiDAR instead. Depth sensors are currently only available on a few high-end mobile devices due to the high cost. 2) Learning-based depth prediction. Liu et al. [29] present a CNN field model to estimate depths from single monocular images, aiming to jointly explore the capacity of CNN model and continuous CRF. David et al. [15] propose a two-scale CNN model trained on images and the corresponding depth maps. However, the 3D CNN models are computing-heavy, which cannot achieve real-time performance on mobile devices. Furthermore, these methods suffer from limited scalability, *i.e.,* they cannot estimate the depth of new objects that are unseen in the training dataset. 3) Using monocular camera on mobile devices. Yang et al. [54] propose a keyframe-based real-time surface mesh generation approach to reconstruct 3D objects from single RGB image. ARCore [49], the well-known AR framework, obtains depth from motion, *i.e.,* using monocular camera combined with inertial measurement unit (IMU) to estimate depth. However, the limitation of these works is that they cannot estimate the accurate depth of the object in motion.

In comparison, MobiDepth obtains the disparity from the dual cameras, instead of moving a single camera. Therefore, MobiDepth can accurately estimate the depth for objects in motion. Furthermore, the stereo matching algorithm used in MobiDepth, *i.e.,* SGBM, does not need to pre-train on any 3D datasets, and thus achieves better scalability.

10 DISCUSSION

The current MobiDepth system has several limitations. 1) *MobiDepth does not take into consideration the impact of auto-focus.* In auto-focus, a motor moves the camera lens backward and forward to adjust the image distance, making it hard for MobiDepth to do the FoV cropping. However, if auto-focus can be well-handled, it may help improve the depth estimation as it improves the captured image quality. 2) *Due to the short distance between the dual cameras on mobile devices, the effective range of MobiDepth to obtain depth is typically limited to 0.5 to 5 meters, i.e., the distance of the dual cameras on HWM40P is about 2.1cm* (the range varies slightly across mobile devices). If the target object lies outside of the effective distance range, the system may not accurately estimate the depth. However, we believe the effective distance range can already cover most AR/VR application scenarios on mobile devices. 3) *MobiDepth may not work well on devices with white-and-black cameras and relies on the computing power of mobile GPU.* For mobile devices with low-end GPUs, the efficiency of stereo matching in MobiDepth may not be guaranteed. This is not a severe issue since most devices today are equipped with powerful cameras and GPUs. These limitations can also be mitigated by incorporating more advanced algorithmic and system optimizations, which we leave for future work.

11 CONCLUSION

In this paper, we propose MobiDepth, the first system to use the dual cameras on commodity mobile devices for real-time depth estimation. MobiDepth does not require dedicated sensors or large-scale data, and works for a wide range of scenarios, where both the device and the target objects can be stationary or moving. To do so, MobiDepth employs three key techniques, including iterative FoV cropping, heterogeneous camera synchronization, and mobile GPU-friendly stereo matching. Extensive experiments have demonstrated that MobiDepth can achieve high accuracy and low overhead. The accuracy remains high on moving objects and moving devices, significantly outperforming ARCore, the state-of-the-art framework for depth estimation on Android.

12 ACKNOWLEDGMENTS

We sincerely appreciate the anonymous shepherd and reviewers for their valuable comments. This work is supported by National Science Foundation of China (62172439, 62122095, 62072472), National Key R&D Program of China (2019YFA0706403), Natural Science Foundation Major Project of Hunan Science and Technology Innovation Program (S2021JJZDXM0022), Natural Science Foundation of Hunan Province (2020JJ5774, 2020JJ2050) and U19A2067, a grant from the Guoqiang Institute, Tsinghua University.

REFERENCES

[1] 2022. https://github.com/ethan-li-coding.
[2] Kristian Ambrosch and Wilfried Kubinger. 2010. Accurate hardware-based stereo vision. *Computer Vision and Image Understanding* 114, 11 (2010), 1303–1316.

[3] Ken Arnold, James Gosling, and David Holmes. 2005. *The Java programming language.* Addison Wesley Professional.
[4] Christian Banz, Holger Blume, and Peter Pirsch. 2011. Real-time semi-global matching disparity estimation on the GPU. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops).* IEEE, 514–521.
[5] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. 2006. Surf: Speeded up robust features. In *European conference on computer vision.* Springer, 404–417.
[6] Michael Bleyer, Christoph Rhemann, and Carsten Rother. 2011. Patchmatch stereo-stereo matching with slanted support windows.. In *Bmvc*, Vol. 11. 1–11.
[7] Bowen Cheng, Bin Xiao, Jingdong Wang, Honghui Shi, Thomas S. Huang, and Lei Zhang. 2020. HigherHRNet: Scale-Aware Representation Learning for Bottom-Up Human Pose Estimation. In *CVPR.*
[8] Sangbum Choi, Seokeon Choi, and Changick Kim. 2021. MobileHumanPose: Toward real-time 3D human pose estimation in mobile devices. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 2328–2338.
[9] Ivan Culjak, David Abram, Tomislav Pribanic, Hrvoje Dzapo, and Mario Cifrek. 2012. A brief introduction to OpenCV. In *2012 proceedings of the 35th international convention MIPRO.* IEEE, 1725–1730.
[10] François Darmon and Pascal Monasse. 2021. The Polar Epipolar Rectification. *Image processing on line* 11 (2021), 56–75.
[11] Pei-Huang Diao and Naai-Jung Shih. 2018. MARINS: A mobile smartphone AR system for pathfinding in a dark environment. *Sensors* 18, 10 (2018), 3442.
[12] ARKit Developers Documentation. 2018. https://developer.apple.com/documentation/arkit.
[13] ARCore Developers Documentation. 2018. https://developers.google.com/ar.
[14] Android Developers Documentation. 2021. https://developer.android.com/training/camera2/multi-camera.
[15] David Eigen, Christian Puhrsch, and Rob Fergus. 2014. Depth map prediction from a single image using a multi-scale deep network. *Advances in neural information processing systems* 27 (2014).
[16] Andreas Geiger, Philip Lenz, and Raquel Urtasun. 2012. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE conference on computer vision and pattern recognition.* IEEE, 3354–3361.
[17] Rostam Affendi Hamzah and Haidi Ibrahim. 2016. Literature survey on stereo vision disparity map algorithms. *Journal of Sensors* 2016 (2016).
[18] Daniel Hernandez-Juarez, Alejandro Chacón, Antonio Espinosa, David Vázquez, Juan Carlos Moure, and Antonio M López. 2016. Embedded real-time stereo estimation via semi-global matching on the GPU. *Procedia Computer Science* 80 (2016), 143–153.
[19] Heiko Hirschmuller. 2007. Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on pattern analysis and machine intelligence* 30, 2 (2007), 328–341.
[20] Dong-Hyun Hwang, Suntae Kim, Nicolas Monet, Hideki Koike, and Soonmin Bae. 2020. Lightweight 3D human pose estimation network training using teacher-student learning. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision.* 479–488.
[21] Hanbyul Joo, Hao Liu, Lei Tan, Lin Gui, Bart Nabbe, Iain Matthews, Takeo Kanade, Shohei Nobuhara, and Yaser Sheikh. 2015. Panoptic studio: A massively multiview system for social motion capture. In *Proceedings of the IEEE International Conference on Computer Vision.* 3334–3342.
[22] Sameh Khamis, Sean Fanello, Christoph Rhemann, Adarsh Kowdle, Julien Valentin, and Shahram Izadi. 2018. Stereonet: Guided hierarchical refinement for edge-aware depth prediction. (2018).
[23] Hyun Myung Kim, Min Seok Kim, Gil Ju Lee, Hyuk Jae Jang, and Young Min Song. 2020. Miniaturized 3D depth sensing-based smartphone light field camera. *Sensors* 20, 7 (2020), 2129.
[24] Andreas Klaus, Mario Sormann, and Konrad Karner. 2006. Segment-based stereo matching using belief propagation and a self-adapting dissimilarity measure. In *18th International Conference on Pattern Recognition (ICPR'06)*, Vol. 3. IEEE, 15–18.
[25] Patrick Knobelreiter, Christian Reinbacher, Alexander Shekhovtsov, and Thomas Pock. 2017. End-to-end training of hybrid CNN-CRF models for stereo. In *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2339–2348.
[26] Vladimir Kolmogorov and Ramin Zabih. 2001. Computing visual correspondence with occlusions using graph cuts. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, Vol. 2. IEEE, 508–515.
[27] Rendong Liang, Ting Cao, Jicheng Wen, Manni Wang, Yang Wang, Jianhua Zou, and Yunxin Liu. 2022. Romou: Rapidly Generate High-Performance Tensor Kernels for Mobile GPUs. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking.* https://doi.org/10.1145/3495243.3517020
[28] TensorFlow Lite. 2021. https://www.tensorflow.org/lite/.
[29] Fayao Liu, Chunhua Shen, Guosheng Lin, and Ian Reid. 2015. Learning depth from single monocular images using deep convolutional neural fields. *IEEE transactions on pattern analysis and machine intelligence* 38, 10 (2015), 2024–2039.
[30] David G Lowe. 2004. Distinctive image features from scale-invariant keypoints. *International journal of computer vision* 60, 2 (2004), 91–110.
[31] Wolfgang Maass. 2000. On the computational power of winner-take-all. *Neural computation* 12, 11 (2000), 2519–2535.
[32] Xing Mei, Xun Sun, Mingcai Zhou, Shaohui Jiao, Haitao Wang, and Xiaopeng Zhang. 2011. On building an accurate stereo matching system on graphics hardware. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops).* IEEE, 467–474.
[33] Lazaros Nalpantidis and Antonios Gasteratos. 2010. Stereo vision for robotic applications in the presence of non-ideal lighting conditions. *Image and Vision Computing* 28, 6 (2010), 940–951.
[34] PerfDog. 2022. https://perfdog.qq.com/.
[35] Todd B Pittman, YH Shih, DV Strekalov, and Alexander V Sergienko. 1995. Optical imaging by means of two-photon quantum entanglement. *Physical Review A* 52, 5 (1995), R3429.
[36] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. 2011. ORB: An efficient alternative to SIFT or SURF. In *2011 International conference on computer vision.* Ieee, 2564–2571.
[37] Thomas Schöps, Torsten Sattler, Christian Häne, and Marc Pollefeys. 2017. Large-scale outdoor 3D reconstruction on a mobile device. *Computer Vision and Image Understanding* 157 (2017), 151–166.
[38] Akihito Seki and Marc Pollefeys. 2017. Sgm-nets: Semi-global matching with neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 231–240.
[39] Naai-Jung Shih, Pei-Huang Diao, Yi-Ting Qiu, and Tzu-Yu Chen. 2020. Situated ar simulations of a lantern festival using a smartphone and lidar-based 3d models. *Applied Sciences* 11, 1 (2020), 12.
[40] Prarthana Shrstha, Mauro Barbieri, and Hans Weda. 2007. Synchronization of multi-camera video recordings based on audio. In *Proceedings of the 15th ACM international conference on Multimedia.* 545–548.
[41] Robert Spangenberg, Tobias Langner, and Raúl Rojas. 2013. Weighted semi-global matching and center-symmetric census transform for robust driver assistance. In *International Conference on Computer Analysis of Images and Patterns.* Springer, 34–41.
[42] John E Stone, David Gohara, and Guochun Shi. 2010. OpenCL: A parallel programming standard for heterogeneous computing systems. *Computing in science & engineering* 12, 3 (2010), 66.
[43] Bjarne Stroustrup. 2013. *The C++ programming language.* Pearson Education.
[44] Vladimir Tankovich, Christian Hane, Yinda Zhang, Adarsh Kowdle, Sean Fanello, and Sofien Bouaziz. 2021. Hitnet: Hierarchical iterative tile refinement network for real-time stereo matching. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 14362–14372.
[45] Stefano Tavani, Andrea Billi, Amerigo Corradetti, Marco Mercuri, Alessandro Bosman, Marco Cuffaro, Thomas Seers, and Eugenio Carminati. 2022. Smartphone assisted fieldwork: Towards the digital transition of geoscience fieldwork using LiDAR-equipped iPhones. *Earth-Science Reviews* (2022), 103969.
[46] Yuan Tian, Yuxin Ma, Shuxue Quan, and Yi Xu. 2019. Occlusion and collision aware smartphone AR using time-of-flight camera. In *International Symposium on Visual Computing.* Springer, 141–153.
[47] Beau Tippetts, Dah Jye Lee, Kirt Lillywhite, and James Archibald. 2016. Review of stereo vision algorithms and their suitability for resource-limited systems. *Journal of Real-Time Image Processing* 11, 1 (2016), 5–25.
[48] Alessio Tonioni, Fabio Tosi, Matteo Poggi, Stefano Mattoccia, and Luigi Di Stefano. 2019. Real-time self-adaptive deep stereo. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*
[49] Julien Valentin, Adarsh Kowdle, Jonathan T Barron, Neal Wadhwa, Max Dzitsiuk, Michael Schoenberg, Vivek Verma, Ambrus Csaszar, Eric Turner, Ivan Dryanovski, et al. 2018. Depth from motion for smartphone AR. *ACM Transactions on Graphics (ToG)* 37, 6 (2018), 1–19.
[50] VR and AR market size 2024. 2021. https://www.statista.com/statistics/591181/global-augmented-virtual-reality-market-size/.
[51] Bill Waggener, William N Waggener, and William M Waggener. 1995. *Pulse code modulation techniques.* Springer Science & Business Media.
[52] Yan Wang, Zihang Lai, Gao Huang, Brian H. Wang, Laurens Van Der Maaten, Mark Campbell, and Kilian Q Weinberger. 2018. Anytime Stereo Image Depth Estimation on Mobile Devices. *arXiv preprint arXiv:1810.11408* (2018).
[53] Jingao Xu, Guoxuan Chi, Zheng Yang, Danyang Li, Qian Zhang, Qiang Ma, and Xin Miao. 2021. FollowUpAR: enabling follow-up effects in mobile AR applications. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services.* 1–13.
[54] Xingbin Yang, Liyang Zhou, Hanqing Jiang, Zhongliang Tang, Yuanbo Wang, Hujun Bao, and Guofeng Zhang. 2020. Mobile3DRecon: real-time monocular 3D reconstruction on a mobile phone. *IEEE Transactions on Visualization and Computer Graphics* 26, 12 (2020), 3446–3456.
[55] Juheon Yi and Youngki Lee. 2020. Heimdall: mobile gpu coordination platform for augmented reality applications. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking.* 1–14.
[56] Changqian Yu, Bin Xiao, Changxin Gao, Lu Yuan, Lei Zhang, Nong Sang, and Jingdong Wang. 2021. Lite-HRNet: A Lightweight High-Resolution Network. In *CVPR.*
[57] Jinrui Zhang, Deyu Zhang, Xiaohui Xu, Fucheng Jia, Yunxin Liu, Xuanzhe Liu, Ju Ren, and Yaoxue Zhang. 2020. MobiPose: Real-time multi-person pose estimation

on mobile devices. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*. 136–149.

[58] Zhengyou Zhang. 1999. Flexible camera calibration by viewing a plane from unknown orientations. In *Proceedings of the seventh ieee international conference on computer vision*, Vol. 1. Ieee, 666–673.