Received: May 9, 2022. Revised: June 14, 2022.

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

Parallel Implementation of 3D Model Reconstruction of Monocular Video Frames in a Dynamic Environment

Ghada M. Fathy[1,2] **Hanan A. Hassan[1]** **Walaa M. Sheta[1]** **Fatma Omara[2]**
Emad Nabil[3]*

[1] Informatics Research Institute, City for Scientific Research and Technological Applications, Alexandria, Egypt
[2]Department of Computer Science, Faculty of Computers and Artificial Intelligence, Cairo University, Giza, Egypt
[3]Computer Science Department, Faculty of Computer and Information Systems, Islamic University of Madinah, Madinah, Saudi Arabia
* Corresponding author's Email: emadnabil@iu.edu.sa

Abstract: Real-time three-dimensional (3D) reconstruction has been widely explored in several domains in motion capture, robot navigation, augmented reality, and autonomous driving. It is considered one of the most efficient solutions to overcome several problems, such as occlusion and collision in computer vision and computer graphics. This process is computationally intensive and it considers a bottleneck for real-time realistic interaction applications which need a quick response to achieve action in real-time. The Monocular 3D model reconstruction (M3DMR) is considered one of the possible solutions to build an accurate 3D reconstruction in a dynamic environment. However, the proposed framework needs a high computational power to create a single frame. Graphics processors unit (GPU) architecture is used to improve the computational time of M3DMR. This study discussed how to maximize the benefits of using GPU resources by using several optimization techniques that enable GPU architecture to achieve the best possible performance for the M3DMR. Different multicore heterogeneous systems are used to evaluate the performance of the proposed framework. Experimental results confirm that our parallel implementation of 3D Model Reconstruction of Monocular Video Frames is valid for different GPU architectures. The proposed parallel implementation can execute 28 FBS compared with the serial version that executes one frame in 30 min.

Keywords: Parallel programming, 3D reconstruction, Real-time, GPU, CUDA.

1. Introduction

Dense three-dimensional (3D) scene reconstruction in a dynamic environment is a common challenge in realistic interaction applications. It is considered the most efficient solution for many computer vision and computer graphics problems, such as occlusion and collision. However, regular 3D reconstruction methods are costly, time-consuming, and complex computation. Therefore, they cannot immediately respond when they are used in time-sensitive applications, such as augmented reality and auto self-driving cars.

Recently, with the rapid development of graphics processing unit (GPU) technologies, real-time applications have become increasingly convenient, and 3D reconstruction has emerged as a popular research subject in this context. Traditional methods of 3D model construction are based on image data at different viewing angles for 3D scene reconstruction [1]. However, they have an extended processing sequence and entail high costs. The environment is unknown in specific application scenarios, such as outdoor augmented reality, and 3D scenes should be in real-time. Microsoft developed Kinect fusion based on Kinect, which scans the environment by enabling users to move with the device in hand. Then, it reconstructs a 3D model of the scanned environment based on the scanned data [2]. This technique suffers from many postictal; for example, Kinect is sensitive to distance and limited range. Hence, it can reconstruct the environment only within a small range of 3D scenes.

Different techniques, such as structure from

motion [3] and simultaneous localization and mapping SLAM [4], are used to reconstruct a real environment in real-time, and they are famous techniques used to reconstruct scenes. Real-time SLAM methods are used to combine maps obtained from moving depth sensors. Subsequently, they are used to navigate and map several types of autonomous agents for different applications [4, 5]. Dense SLAM methods based on RGB-D data are widely used to navigate and generate scene construction [6, 7]. Such methods are mainly limited by depth cameras that are mostly based on active sensing. Thus, they cannot have an accurate performance under sunlight, So, reconstruction and mapping in outdoor environments are considered less accurate. Further studies have been conducted to improve the accuracy of depth maps by predicting depth using deep learning techniques [8-10]. Through these techniques, an absolute scale can be detected from examples and consequently predicted from a single image without the need for scene-based assumptions or geometric restrictions [9-11]. Although the predicted depth map can improve the performance of monocular SLAM, it remains inaccurate and performs poorly in a dynamic environment. The authors in [12, 13] have proposed the dense 3D reconstruction of a complex dynamic scene for a single image. In this approach, dynamic scenes are approximated using numerous piecewise planar surfaces, and each one has its rigid motion.

[12] presents a 3D reconstruction technique in a complex dynamic scene using two frames by applying super-pixel over-segmentation to the image. They presented a generically dynamic (hence non-rigid) scene with a piecewise planar and rigid approximation. Also, they reduced the reconstruction problem to a “3D jigsaw puzzle” which takes pieces from an unorganized “soup of super-pixels”.

Another approach is presented using unsupervised learning and point cloud fusion to reconstruct a 3D scene in a dynamic environment (M3DMR) [14]. This approach focuses on applying unsupervised learning to predict the depth map, camera position, and object motion through a sequence of video frames. Then, the sequence of frames is reconstructed via point-cloud fusion. These approaches achieve high accuracy, but reconstruction entails high costs and needs 15,30 minutes for a single frame, respectively.

GPUs are widely used to speed up intensive computation applications in different domains, and they have provided new opportunities for embedded systems that fail to perform computationally intensive tasks in real-time. However, creating a point cloud and generating 3D reconstruction for a whole scene in computing unified device architecture (CUDA) are considered complex tasks. [23, 25] using GPU architecture to improve the processes of 3D reconstruction. The authors [23] used neural networks such as LSTM and GRU to generate a full 3D point cloud from outdoor LiDAR datasets. The main purpose is to use the motion-based neural network that integrates motion features between two consecutive point clouds. NVIDIA GeForce RTX 2080Ti is used for training in testing among different datasets, for example, the KITTI dataset. On the other hand [23] used GPU with MATLAB platform to segment the optical flow field for a full dynamic scene.

Table 1. 3D reconstruction time of the most relevant techniques

Ref.	Scene	Methods	Device	Time (T/F)
[19]	Single Static object	Monocular SLAM	NVIDIA GeForce TITAN X-without CUDA	21 ms
[21]	Single Dynamic object	Markless 3D human motion capture	GeForce RTX 2070 without CUDA	40 ms
[22]	Single Dynamic object	GCN network	Nvidia GeForce RTX 2080Ti -without CUDA	23 ms
[20]	Full Static scene	Online incremental mesh generation	a single CPU thread third-party library OpenCV 2	57.21 ms
[24]	Full Static Scene	visual parallel SLAM on UAV platforms	TX2 embedded development module with CUDA	8.41 s
[23]	Full Dynamic scene	LSTM and GRU networks	NVIDIA GeForce RTX 2080Ti-without CUDA	56 ms
[25]	Full Dynamic scene	segments the optical flow field	MATLAB and GPU	1 m
[12]	Full Dynamic scene	Super-pixel over-segmentation	Intel core i7	15-20 m
[14]	Full Dynamic Scene	Unsupervised learning and point cloud fusion	Tesla V100 and Tesla M10 without CUDA	30 m

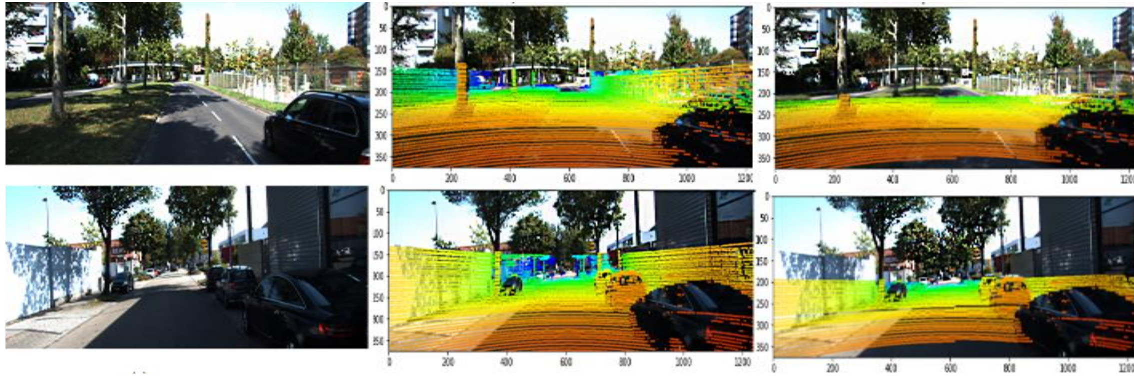

Figure. 1 3D point cloud mapped to 2D image (a) selected input frame (b) Ground truth (c) predicted points [14]

Table 1 summarizes the time of the most relevant implementations of 3D scene reconstruction. In some cases, implementing a particular model may even be impossible because of memory restrictions. CUDA platforms are used among the essential platforms to program GPU devices. CUDA is a parallel computing platform and programming model developed by NVIDIA for general computing [15]. Several types of research work use GPUs to improve the performance of applications and reduce computational time [16, 17]. Ivanavičius et al. [18] proposed a matching algorithm called Cyclops2 for Stereo reconstruction. This algorithm produces a disparity image and provides two rectified grayscale images.

Matching is based on the concept of minimizing the calculation of a weight function using the absolute difference in pixel intensities. The CUDA-parallel programming library is optimized for the embedded NVIDIA Jetson platform is used [18].

This work aims to improve the computational time of the M3DMR framework that is proposed in our previous work [14]. Profile and complexity analyses were performed to select the bottleneck time of serial implementation. In addition, several optimization techniques are used to maximize the benefits of using GPU resources to decrease the total execution time of the reconstruction process and create a 3D point Cloud scene from a sequence of monocular RGB video frames in a dynamic environment. Moreover, to demonstrate the flexibility of the proposed implementation and validate the system, we are experimenting with different multicore heterogeneous systems.

The rest of the paper is organized as follows: Section 2 gives a brief explanation of Monocular 3D model reconstruction (M3DMR), the serial implementation, the profile analysis, and the complexity analysis of the framework. Section 3 introduces GPU parallel computing architecture. section 4 illustrates the performance optimization approaches used in GPU implementation. Section 5 shows experimental results. Section 6 discussion. Finally, section 7 concludes this research and suggests future work.

2. Monocular 3D model reconstruction

2.1 Overview of Monocular 3D model reconstruction (M3DMR)

All M3DMR aims to reconstruct a constant and accurate 3D scene using a point cloud technique for a dynamic environment. The framework environment consists of moving and static background objects.

Fig. 1 shows the output of the M3DMR framework compared with the ground truth.

Framework uses the updating data from a sequence of Monocular RGB video frames instead of the expensive cost of multi-sensor data. M3DMR consider a suitable solution to solve realistic interaction problems, such as occlusion and collision. M3DMR decided into two levels.

Level 1, using an unsupervised learning technique to estimate object motion, scene depth, and camera pose during the online refinement process. Level 2, reconstruct a full 3D scene model through frame-wise point cloud fusion (see Fig. 2). For more details, M3DMR is described in [14].

2.2 Serial implementation of M3DMR

The serial implementation is proposed in [14]. The per-frame point cloud p_i is reconstructed by $\boldsymbol{D_i}$ is the depth for a single frame, the final Motion $\boldsymbol{E_m^F}$ consists of individual moving objects ψM_o and Camera motion $\psi E_{i \to j}$. K is the intrinsic camera calibration matrix.

$$p_i = (E_m^F)^{-1}\,\pi(u, D_i) \qquad (1)$$

Where u denote as homogeneous representation of a pixel $u = (x, y, 1)^T$ and $\pi(u)$ is the back projection from image to camera coordinate,

Received: May 9, 2022. Revised: June 14, 2022.

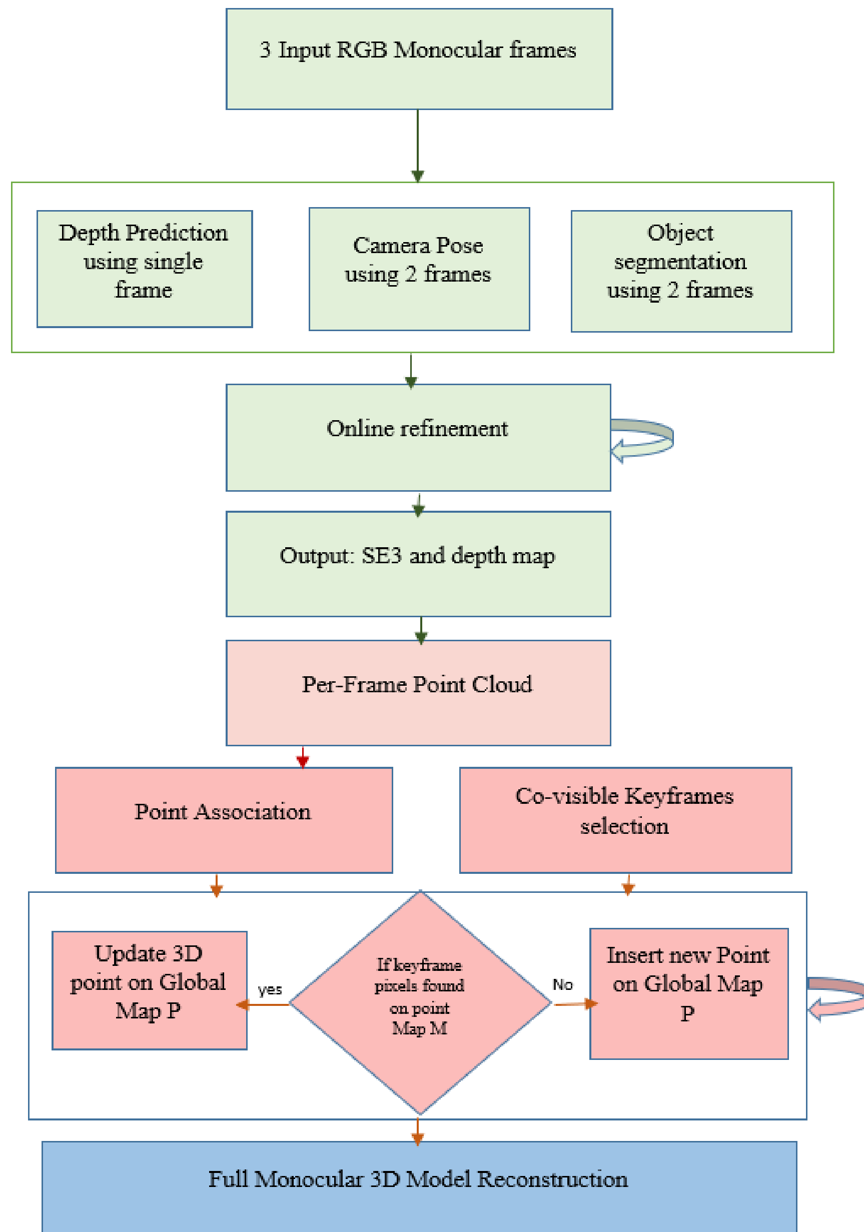

Figure. 2 The M3DMR framework overview

$\pi(u, D_i) = K^{-1} D_i u.$

In point association process calculated as

$$u^k = f\left(K\,(E_m^F)^k \pi(u^i)\right) \tag{2}$$

The co-visible of keyframe $k_1, k_2 \ldots . k_n \in KF$ selected according to the length of monocular video frames on the online refinement process.

The probabilistic filter is used. Each 3D point in global model P is represented by p_i^n, and the confidence counter C_c is defined as how often the 3D point is observed in co-visible keyframes. The new observation available in the latest frame $\boldsymbol{i}$ according to the following equations:

$$p_i^n = \frac{\left(w_A p_i + w^0 (E_m^F)^{-1} \pi(u^i)\right)}{w_A + w^0} \tag{3}$$

$$C_C^n = \frac{(w_A C_c + w^0 \left\|(E_m^F)^{-1} \pi(u^i) - p_i\right\|)}{w_A + w^0} \tag{4}$$

$$w_A^n = \min(w_A + w^0, W_\varepsilon) \tag{5}$$

Where p_i^n means the newly updated point, w^0 is a constant equal to 1 and W_ε is the truncation threshold equal to 100.

2.3 Profile analysis of serial M3DMR implementation

In the analysis of the time consumption of the

Algorithm 1 represents the pseudo code of 3D model reconstruction process.

Algorithm 1: Generate a 3D Reconstruction for the dynamic scene (Serial implementation)

Input: P ← global Model, hash map contains 3D point cloud, confidence counter, average weight, and point status (Stable, Unstable) with length s
M← point Map, mapping of [x, y] and pointID for u^k with length s
L← the u^k dimension
F ← total number of frames
KF ← total number of Co-visible keyframe

1 **for** $\boldsymbol{i} = 1 \rightarrow F$ do
2 | % project current frame to world coordinates
3 | $p_i = (E_m^F)^{-1}\,\pi(u, D_i)$
4 | **for** *k=1→ KF* **do**
5 | | *% project world coordinates to co-visible KF*
6 | | $p_i \rightarrow u^i \rightarrow u^k$
7 | | $u^k = f\left(K\,(E_m^F)^k \pi\left(u^i\right)\right)$
8 | | **for** j=*1*→ *L* **do** % point association
9 | | | **if** j in M:
10 | | | | % point is visible
11 | | | | % current frame-wise 3D point associated with the 3D global model
12 | | | | % update point info
13 | | | | $p_i^n = \left(w_A p_i + w^0 (E_m^F)^{-1}\,\pi\left(u^i\right)\right) / (w_A + w^0)$
14 | | | | $C_C^n = \left(w_A C_c + w^0\,\left\|(E_m^F)^{-1}\,\pi\left(u^i\right) - p_i\right\|\right) / (w_A + w^0)$
15 | | | | $w_A^n = \min(w_A + w^0, W_\varepsilon)$
16 | | | | **if** C_C^n < stable_threshold
17 | | | | | % point is stable
18 | | | | **else**
19 | | | | | % point unstable
20 | | | **else**
22 | | | | insert p_i to P % with all point info
23 | | | | insert j to M
24 | | | |___
25 | | |______
26 | |____________
27 | % remove unstable points from P
28 |______________________________

Output: accumulated global model P for 3D reconstruction

proposed M3DMR framework, the search on point map (M) (line 9 in Algorithm 1) is repeatedly executed L times for each keyframe KF, where L is equal to the number of keyframe pixels (x, y). The profile analysis reports that almost 97 % of the total frame execution time is consumed in the search on the point map (see Fig. 3). However, only 2 % and 1 % are spent adding and updating, filtering noise processing for a 3D point cloud on the global map P, and creating the initial 3D point cloud and co-visible keyframes, respectively.

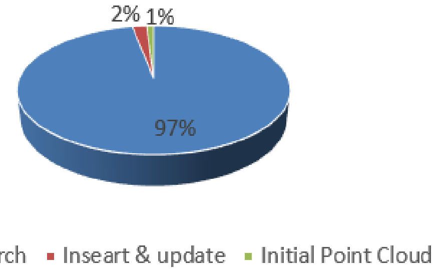

Figure. 3 Profile analysis of serial M3DMR implementation

2.4 Complexity analysis of serial M3DMR

The time complexity of M3DMR consists of several parts. In Fig. 3, a per-frame point cloud is created for each frame at $O(n)$. All pixels of the current frame compared with co-visible keyframes pixels. to generate u^k with a token of $O(n)$. The time complexity of the point association includes a search on the point map and updating or adding point could process. A linear search is performed to check the visibility of pixels in the current frame with all co-visible keyframes and return pointID for updating. The time complexity of the linear search is $O(n)$.

With the system, the linear search is computed for all keyframe pixels, so $O(n^s)$ is necessary. The total time complexity of the M3DMR framework is

$$O(n^2) + O(n^s) = O(n^s) \tag{6}$$

3. GPU parallel computing architecture

NVIDIA launched the concept of CUDA, which manages GPUs for parallel computing. With CUDA, general-purpose programming can be achieved through different languages [27]. The topology of the CUDA platform supports thread-level parallel computation by activating thousands of threads to execute using GPUs. The cost for thread creation and dispatch is negligible in GPUs because they are implemented by hardware rather than software as in a central processing unit (CPU). GPUs are responsible for intensive floating-point computation, such as dense vector and matrix operation. Data exchange occurs between a CPU and GPU through a PCIe bus. GPUs contain streaming multiprocessors (SM) which consist of a large number of single- and double-precision cores. GPUs have different memory units, such as register, shared memory, constant

Received: May 9, 2022. Revised: June 14, 2022.

memory, texture memory, L2 cache, and global memory. A GPU function that executes on a device is called a kernel. Kernels run on a kernel grid composed of many thread blocks. These thread blocks are then executed in parallel and unable to communicate except through a global device memory. By contrast, all threads in one thread block can intercommunicate via shared memory or barrier.

Some concepts about CUDA and GPU that should be considered are listed below [27].

1- Dependency must be avoided as much as possible, and operations in the algorithm should be performed concurrently.
2- In NVIDIA's CUDA architecture, the thread blocks are executed concurrently using device kernels. Each thread block includes a group of warps consisting of 32 threads. Inside warp, threads simultaneously execute the same branch of the algorithm time. To get the best performance, avoid loops, and conditional statements as much as possible.
3- Read and write operations should be in the range of neighbouring memory cells. Otherwise, updates in memory significantly extend the processing time.

4. Parallel implementation of M3DMR

This section describes the design of the parallel implementation of a monocular 3D reconstruction model in a dynamic scene using CUDA and GPU.

Fig. 4 shows the serial framework mapped and divided among CPUs and GPUs. The left side (Host) represents the framework parts that run on a CPU, and the right side (Device) includes the parts that run on a GPU.

Fig. 4 presents the same equations of the serial framework used in this study. The parallel design considers the following:

1) The host allocates memory and controls the outer portion of loops and the overflow of the framework.
2) The device (GPU) excites the most intensive computational parts of the framework.

Algorithm 2 describes the overview pseudo-code of CUDA-parallel implementation, which involves three device kernels; parallel linear search, point cloud addition and update, and unstable point removal.

4.1 Parallel implementation complexity

As mentioned in section 2.4, the serial time

Algorithm 2: Generate a 3D Reconstruction for the dynamic scene (CUDA Version)

Input: P ← global Model, arrays contains 3D point cloud, confidence counter, average weight, and point status (Stable, Unstable) with length s
M← point Map, mapping of [x, y] and pointID for u^k with length s
F ← total number of frames
KF ← total number of Co-visible keyframe
$pointID_{index}$ ← the output array of linear search
1 **for** $i = 1 \rightarrow F$ do
2 | % project current frame to world coordinates
3 | $p_i = (E_m^F)^{-1}\pi(u, D_i)$
4 | **for** k=1→ KF **do**
5 | | *% project world coordinates to co-visible KF*
6 | | $p_i \rightarrow u^i \rightarrow u^k$
7 | | $u^k = f\left(K\,(E_m^F)^k \pi(u^i)\right)$
8 | | Memory **transfer** $\boldsymbol{M}$, $\boldsymbol{u^k}$ **P**, $\boldsymbol{pointID_{index}}$ from H-D
9 | | **Kernel1_ParallelLinearSearch** (u^k, M)
10 | | **Kernel2_addAndUpdatePointCloud** (P, M, $pointID_{index}$, u^k)
11 | |___
12 | **Kernel3_RemoveUnstablePoint** (P, M)
13 | **Memory transfer M, P from D-H**
|___

Output: accumulated global model P for 3D reconstruction

complexity is O(n). The parallel implementation doesn't change the big O notation. That is because the parallel simply divides the overall execution time by the number of parallel executions. However, we can evaluate the parallel implementation as the following:

The cost of computational (CoC) is calculated as shown in Eq. (7).

$$CoC = total\ execution\ time\ (e) * total\ number\ of\ processors\ S(n) \quad (7)$$

The serial computation:

$$t_s = e \quad (8)$$

The cost of parallel computation:

$$C_P = t_p * n + C(t) \quad (9)$$

where t_p parallel execution time is given by $t_s/S(n)$ and $C(t)$is the communication overhead.

Received: May 9, 2022. Revised: June 14, 2022.

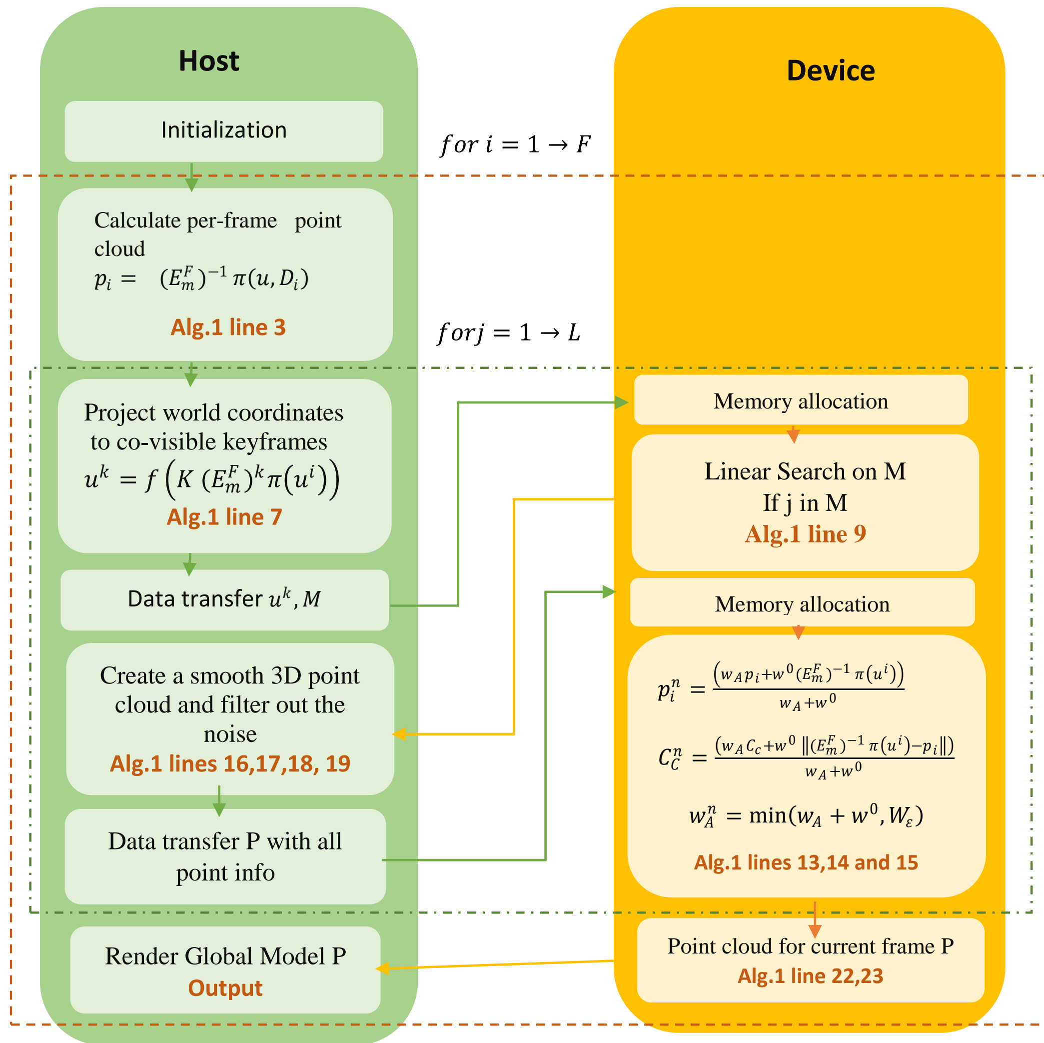

Figure. 4 Parallel design of M3DR in Host (H) and Device (D)

4.2 Kernel 1: Parallel Linear Search

As mentioned before, search on point map is the most time-consuming process (section 2; line 9 Algorithm 1). The linear search has a time complexity of $O(n)$) and is performed for each pixel in the current frame. It is also applied to assign each thread with one pixel in the current frame.

In Fig. 5, the number of threads N_{thr} is equal to the number of pixels in co-visible Keyframes s. Each thread searches the point map (M) via the x and y coordinates. The number of thread block N_{block} is calculated as follows:

$$N_{block} = \left(\frac{N_{thr}}{\mathrm{MAXNUMBER_{thr}}}\right) + \left((N_{thr}\%\mathrm{MAXNUMBER_{thr}})?\,1:0\right) \quad (10)$$

Where $\mathrm{MAXNUMBER_{thr}}$ is the maximum number of active threads. Linear search is considered the most efficient way because of the following aspects. First, the point map is unsorted, and linear search is a suitable solution for unsorted arrays. Second, other approaches need costly pre-processing, such as binary search and HashMap. For example, a system is searched with keyframe pixels x and y, but using these two values during the search needs expensive sorting pre-processing and consumes time. In the proposed parallel implementation, most of the concepts mentioned in section 3.1 are covered, and each thread in a warp is read and written in the range of neighbouring memory cells (Fig. 5).

The searching process is considered independent, so each thread is responsible for checking one pixel and does not need to wait for other threads. The pseudocode of parallel linear search is presented in Algorithm 3, kernel 1.

4.3 Kernel 2: Add and update point cloud in global map

After the execution time of the frames via a parallel search is reduced, the framework time should

Received: May 9, 2022. Revised: June 14, 2022.

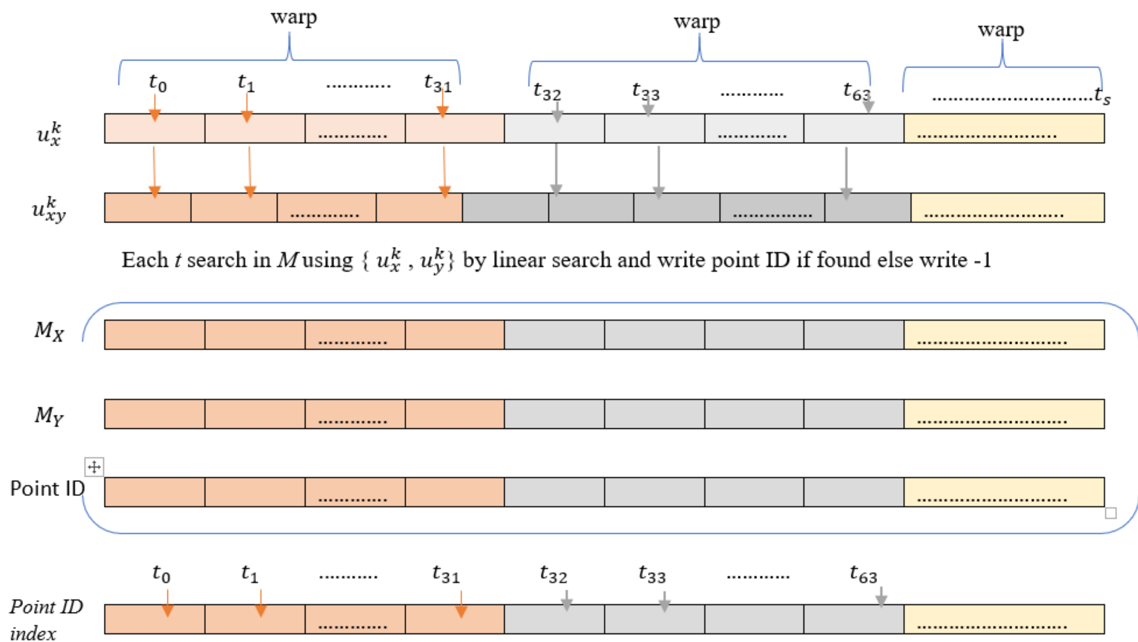

Figure. 5 Parallel design for searching on M using linear search

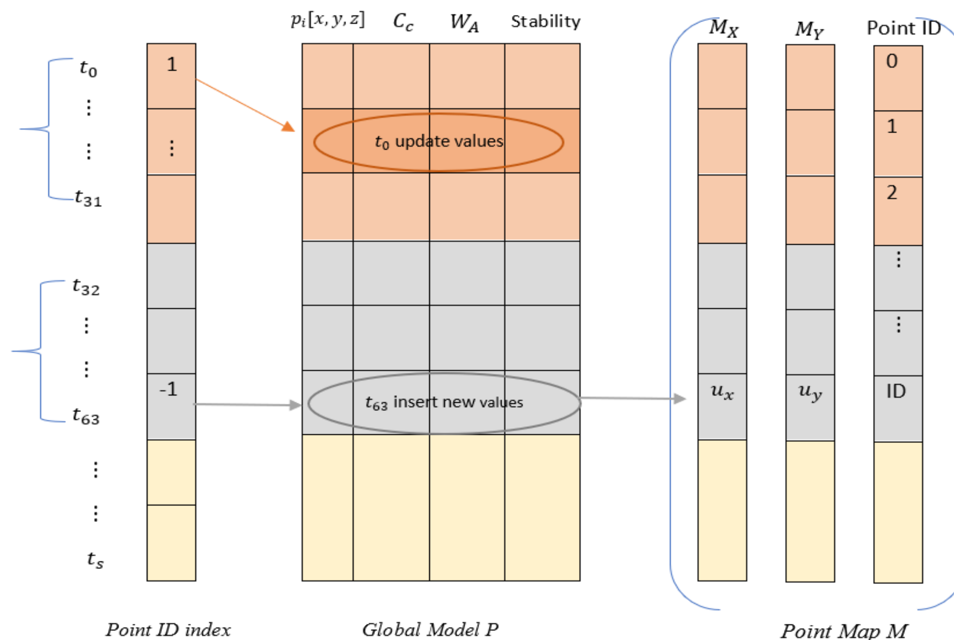

Figure. 6 Add and update global model

be improved further so that it can be accepted for real-time applications. Adding or updating the 3D point cloud is chosen as the second phase in parallel implementation. The global model consists of a list of 3D point clouds with correlating characteristics, such as confidence counter, average weight, and stability flag (Fig. 6).

The 3D point cloud p_i is updated in case the search output is not equal to -1, indicating that the mapped key frame point u^k is found on point map M. If p_i is equal to -1, the estimated 3D point is added to the global model P, and $u^k[x,y]$ is added to M.

The number of active threads, N_{thr}, and the number of thread block N_{block} are calculated as the same as in kernel 1, and each thread is responsible for adding or updating one point. Memory is transferred between a host and a GPU device once for each frame in video sequence frames to reduce the overhead of memory traffic. The process of updating and adding each frame into P is explained in Algorithm 4, Kernel 2.

Algorithm 3 Kernel 1: Parallel linear search

Input: M← point Map, mapping of [M_x, M_y] and pointID for u^k with length s
u^k (u_x^k, u_y^k) ← *project world coordinates to co-visible keyframe*
L← u^k *length*

1 $tid = blockDim.x * blockIdx.x + threadIdx.x$
2 ***if*** $tid > L$ do
3 $return$
4 **for** $\boldsymbol{i} = 0 \rightarrow L$:
5 ***if*** $u_x^k[tid] = M_x[tid]$ && $u_y^k[tid] = M_y[tid]$:
6 $pointID_{index}[tid] = i$
7 Break
8 ***else***:
9 $pointID_{index}[tid] = -1$

Output: $pointID_{index}$: *the index of point in M or* -1

4.4 Kernel 3: Filtering out the noise (remove unstable points)

The system starts to filter the noise through a probabilistic noise filter after a visibility check between the current frame, and all the selected keyframes are completed by removing the point record containing a −1 value, which indicates that this point is unstable. In Algorithm 5, kernel 3 checks if the point state flag is set on the update process (kernel 2) and removes the point record from the global and point maps by setting them to −1.

5 Experiments

5.1 Experimental configuration

Two different multicore heterogeneous systems are used to evaluate the performance. The first one is equipped with PowerEdge Dell 740XD (2× Intel Xeon Gold 6248 2.5G, 20C/40T, 10.4GTs, 27.5 M Cache, Turbo, HT [150W] DDR4-2933+384G RAM) with Tesla V100 GPU [31]. The second system is a virtual machine on cloud computing with NIVIDIA Tesla M10 GPU card. Table 2 illustrates the comparison between two GPU card specs.

The framework is implemented by CUDA 10.1, Python, and Pycuda library. The experiments are conducted using the KITTI dataset [29].

Algorithm 4_Kernel 2: Add and update point cloud

Input: P ← global Model, arrays contains 3D point cloud p, confidence counter C_c, average weight W_A, and point_state (Stable, Unstable) with length s
M← point Map, mapping of [x, y] and pointID for u^k with length s
L← u^k length
$pointID_{index}$ ← the output of linear search with length L

1 $tid = blockDim.x * blockIdx.x + threadIdx.x$
2 ***if*** $tid > L$ do
3 $return$
4 ***if*** $pointID_{index}[tid]! = -1$
5 $p_i^n[tid] = \frac{\left(w_A p_i + w^0 (E_m^F)^{-1} \pi(u^i)\right)}{(w_A + w^0)}$ % *update 3D Point Cloud*
6 $C_C^n[tid] = \frac{(w_A C_c + w^0 \, \|(E_m^F)^{-1} \pi(u^i) - p_i\|)}{w_A + w^0}$ *%update Confidence*
7 $w_A^n[tid] = w_A^n = \min(w_A + w^0, W_\varepsilon)$ *%update Average Weight*
8 Check point_state () % *check state according to confidence* and set it 0 or 1
9 ***else***:
10 $p_i[tid] = insertToGlobelModel()$
11 $C_C^n[tid] = intial_confidence$
12 $w_A^n = initial_weight$
13 *point_state*=0

Output: $M \rightarrow point\ Map$, $P \rightarrow Golbal\ Model$

Algorithm 5_Kernel 3: Remove unstable point

Input: P ← global Model, arrays contains 3D point cloud p, confidence counter C_c, average weight W_A, and point_state (Stable, Unstable) with length s
M← point Map, mapping of [x, y] and pointID for u^k with length s

1 $tid = blockDim.x * blockIdx.x + threadIdx.x$
2 ***if*** $tid > s$ do
3 $return$
4 ***if*** $point_state = 0$
5 Set P {$p_i^n[tid]$, $C_C^n[tid]$, $w_A^n[tid]$, $point_{state}$} $= -1$
6 Set $M_x[tid], M_y[tid] = -1$

Output: $M \rightarrow point\ Map$, $P \rightarrow Golbal\ Model$

Table 2. Specs comparison between Tesla V100 and Tesla M10 [30]

	NVIDIA Tesla V100 PCIe 32 GB	**NVIDIA Tesla M10**
Technical info		
Boost clock speed	1380 MHz	1306 MHz
Core clock speed	1230 MHz	1033 MHz
Floating-point performance	14,131 gflops	4x 1,672 gflops
Manufacturing process technology	12 nm	28 nm
Pipelines	5120	4x 640
Texture fill rate	441.6 GTexel /s	4x 52.24 GTexel / s billion / sec
Thermal Design Power (TDP)	250 Watt	225 Watt
Transistor count	21,100 million	1,870 million
Memory		
Maximum RAM amount	32 GB	4x 8 GB
Memory bandwidth	897.0 GB / s	4x 83.2 GB / s
Memory bus width	4096 Bit	4x 128 Bit
Memory clock speed	1752 MHz	5200 MHz
Memory type	HBM2	GDDR5

5.2 Experimental results

Experiment 1:

In Experiment 1, the execution time per frame is evaluated using different multicore heterogeneous systems. Tesla V100 on a Linux operating system and Tesla M10 on a virtualized Windows 10 machine are used. This experiment confirms that the proposed parallel implementation can reconstruct several frames in real-time through different configurations (high and low).

In Fig.7, the evaluation was conducted using a different number of threads per block from 32 to 1024 threads. Each GPU device has its architecture and limitations. In Fig. 7, the best performance of a parallel monocular 3D model reconstruction framework was using 256 thread blocks on Tesla V100. By contrast, the best performance was recorded in grad M10 with 128 thread blocks. Based on the execution time in Figs. 7, 8 illustrates the capacity of two devices to reconstruct frames in 1 s.

Experiment 2:

Experiment 2 is performed to measure the effect of using a different number of key frames in accuracy and execution time. As mentioned in [15], five key frames are used and obtained an acceptable result. However, we cannot evaluate the framework with more than five key frames because of the long execution time of serial implementation. In this experiment, the framework is tested on different key frames scattered among video frames, starting from 2 to 9 key frames.

The average root mean square error (RMSE) and the execution time for a single frame are calculated. In Fig. 9, the RMSE is slightly affected by increasing number of keyframes, and eight keyframes record the lowest error in 20 video frames. However, adding more keyframes increases frame processing and extends the execution time. Based on Fig. 9, accuracy is inversely related to execution time; whenever the number of keyframes increases, the RMSE decreases, and execution time prolongs. Using five keyframes is considered the middle of accuracy and execution time.

6 Discussion

The work in this paper presents the CUDA-parallel implementation of monocular 3D reconstruction, which has been proposed in our previous study [14]. The framework aims to generate a smooth and accurate 3D point cloud from a sequence of monocular video frames containing existent moving objects. The framework is split into two stages using an unsupervised learning technique to estimate depth from a single RGB frame, camera position, and object motion. A complete 3D reconstruction scene is created at the second stage through frame-wise point cloud fusion. The first stage is performed in real-time, whereas the model runs at 30 FPS on advanced GPUs [10]. Most of the time is consumed at the second stage, which is executed in an extended time that reaches 15 min per frame, so it is unacceptable for real-time interactive applications.

The main objective of this research is to improve time consumption to generate a 3D point cloud by using the CUDA platform on different advanced GPUs. The parallel implementation consists of three main kernels. The first kernel is used to search the point map M using a linear search algorithm. In the second kernel, the point cloud is added and updated in the global map. In the third kernel, an unstable point is removed from maps. Table 3 illustrates the comparison between the average frame time in serial implementation through a parallel linear search (kernel 1) and the final execution time after kernels 2 and 3 are applied using Tesla V100 GPU and M10 device.

Received: May 9, 2022. Revised: June 14, 2022.

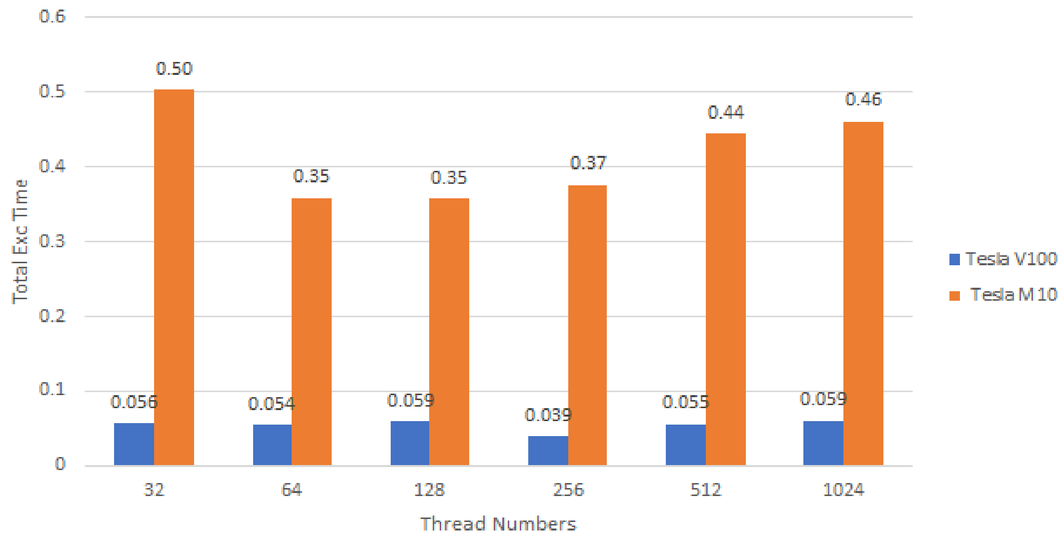

Figure. 7 The effect of using different number of threads blocks on total execution time

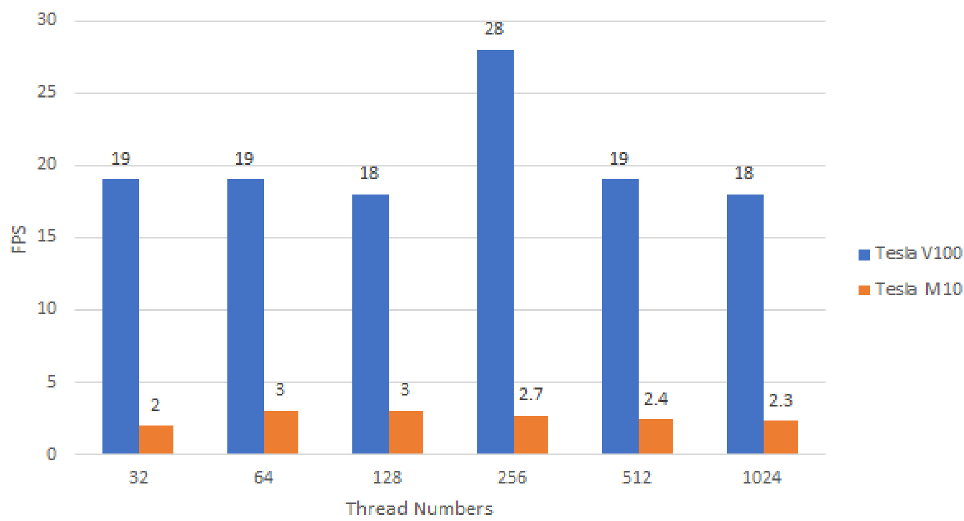

Figure. 8 The effect of using different number of threads blocks on FPS

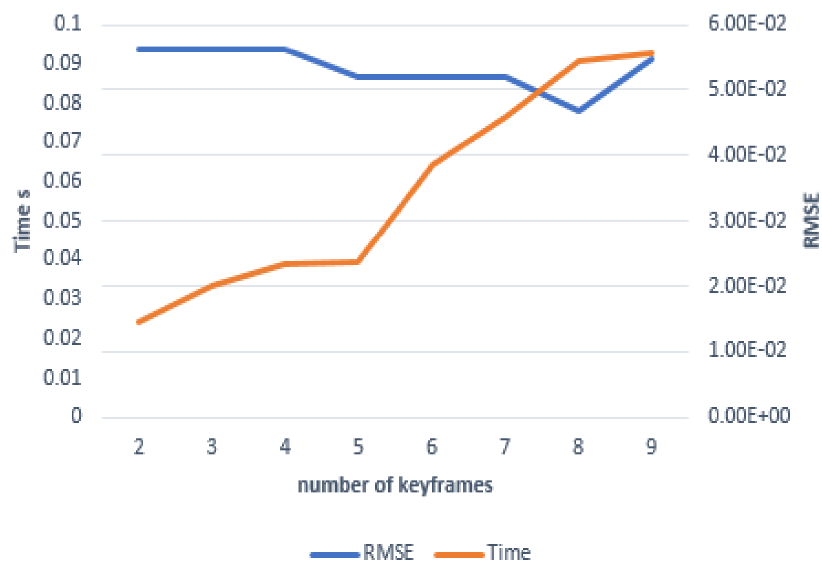

Figure. 9 RMSE Vs Time using different keyframes

Table 3. Frame time in serial and GPU parallel implementation

GPU Device	Serial	Apply kernel 1	Apply kernel 2 and 3
Tesla V100	30 minutes	31 second	0.039 second
Tesla M10	30 minutes	48 second	0.35 second

In Table 3, the proposed GPU parallel implementation of the monocular 3D model reconstruction framework significantly improves the execution time for a single frame. After kernel 1 is applied, the speed increases by 58× serial implementation on Tesla V100. NVIDIA Tesla V100 is the most advanced GPU accelerator built to expedite AI, HPC, and graphics. Powered by NVIDIA Volta, the latest GPU architecture, namely, Tesla V100 offers the performance of up to 100 CPUs in a single GPU. However, it is expensive and not popularly used. Therefore, we evaluate our implementation on different GPU devices, which are more popular and flexible to virtualize the specs as needed. The parallel framework on the Tesla M10 device obtains 37.5× speed compared with the serial implementation. After kernel 1 is applied, time is still inapplicable to real-time applications. Kernels 2 and 3 improve the total execution time with an average of 0.039 seconds per frame and 0.35 seconds per frame using Tesla V100 and Tesla M10 respectively. So, it is acceptable for real-time application. The strength of the proposed parallel implementation is the flexibility to be executed on different GPU devices with different specifications and still maintains the time speedup.

7 Conclusion and future work

In this study, a CUDA-parallel implementation was illustrated to improve the total execution time of point cloud creation for the RGB frame in the monocular 3D reconstruction framework. The results showed that the proposed framework enhances the execution time which would be acceptable for real-time applications. The serial implementation was subjected to performance analysis to evaluate the most time-consuming parts. As a result, three parts were chosen to complete parallelization, and the point map M was searched using a linear search technique. The point cloud in the global map was added and updated, and the unstable point was removed from maps. Two multicore heterogeneous systems were used to evaluate performance. The proposed parallel design was applied to successfully execute the framework in real-time. The parallel design was separated into two phases. First, each pixel was assigned to one thread to compute the linear search. The speedup reached 58× for every frame. Second, the point cloud was updated or inserted into the global map. The total time of parallel implementation was 0.03 seconds instead of 30 min of serial implementation.

In future studies, the proposed framework could be used for real-time interaction applications, such as increased reality and application performance evaluation. Moreover, parallel implementation was adopted for mobile phones composed of a GPU device.

Funding

This research is funded by the Deanship of Scientific Research, Islamic University of Madinah, Madinah, Saudi Arabia.

Conflicts of interest

The authors declare no conflict of interest.

Author contributions

Conceptualization, Ghada, Hanan and Emad; methodology, Ghada, Hanan, Fatma, Walaa and Emad; implementation, Ghada; validation, Ghada, Hanan, and Fatma Omara; formal analysis, Walaa and Fatma; investigation, Ghada, Emad; writing—original draft preparation, Ghada; writing—review and editing, Fatma, Walaa, Hanan, and Emad; supervision, Fatma and Walaa.

References

[1] R. Koch and J. M. Frahm, "Visual-Geometric Scene Reconstruction from Image Streams," In: *Proc. of the Vision Modeling and Visualization Conference*, pp. 367-374, 2001.

[2] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, and A. Fitzgibbon, "Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera", In: *Proc. of the 24th Annual ACM Symposium on User Interface Software and Technology*, pp. 559–568, 2011.

[3] P. Nyimbili, H. Demirel, D. Z. Seker, and T. Erden, "Structure from motion (sfm)-approaches and applications", In: *Proc. of the International Scientific Conf. on Applied Sciences*, Antalya, Turkey, pp. 27–30, 2016.

[4] K. Tateno, F. Tombari, I. Laina, and N. Navab, "Cnn- slam: Real-time dense monocular slam with learned depth prediction", In: *Proc. of the IEEE Conf. On Computer Vision and Pattern*

Recognition, pp. 6243–6252, 2017.

[5] Y. Saito, R. Hachiuma, M. Yamaguchi, and H. Saito, "In- plane rotation-aware monocular depth estimation using slam", In: *Proc. of International Workshop on Frontiers of Computer Vision*, pp. 305–317, 2020.

[6] M. Keller, D. Lefloch, M. Lambers, S. Izadi, T. Weyrich, and A. Kolb, "Real-time 3d reconstruction in dynamic scenes using point-based fusion", In: *Proc. of 2013 International Conf. on 3D Vision-3DV*, pp. 1–8, 2013.

[7] K. Chen, Yu. Lai, and Shi. Hu, "3d indoor scene modelling from rgb-d data: a survey", *Computational Visual Media*, Vol. 1, No.4, pp. 267–278, 2015.

[8] F. Liu, C. Shen, G. Lin, and I. Reid, "Learning depth from single monocular images using deep convolutional neural fields", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 38, No. 10, pp. 2024–2039, 2015.

[9] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab, "Deeper depth prediction with fully convolutional residual networks", In: *Proc. of 2016 Fourth International Conf. On 3D Vision (3DV)*, pp. 239–248, 2016.

[10] V. Casser, S. Pirk, R. Mahjourian, and A. Angelova, "Depth prediction without the sensors: Leveraging structure for unsuper- vised learning from monocular videos", In: *Proc. of the AAAI Conf. on Artificial Intelligence*, Vol. 33, pp. 8001–8008, 2019.

[11] D. Eigen and R. Fergus, "Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture", In: *Proc. of the IEEE International Conf. on Computer vision*, pp. 2650–2658, 2015.

[12] S. Kumar, Y. Dai, and H. Li, "Superpixel soup: Monocular dense 3d reconstruction of a complex dynamic scene", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 43, No. 5, pp. 1705–1717, 2019.

[13] S. Kumar, Y. Dai, and H. Li, "Monocular dense 3d reconstruction of a complex dynamic scene from two perspective frames", In: *Proc. of the IEEE International Conf. on Computer Vision*, pp. 4649–4657, 2017.

[14] G. Fathy, H. Hassan, W. Sheta, F. Omara, and E. Nabil, "A novel no-sensors 3d model reconstruction from monocular video frames for a dynamic environment", *PeerJ Computer Science*, Vol. 7, p. e529,2021.

[15] R. Pickles, "White paper-next generation graphics gpu shader and compute libraries", In: *Proc. of 2020 AIAA/IEEE 39th Digital Avionics Systems Conf. (DASC)*, pp. 1–6, 2020.

[16] H. Hassan, G. Fathy, Z. Fayez, and W. Sheta, "Exploring the parallel capabilities of gpu: Berlekamp-massey algorithm case study", *Cluster Computing*, Vol. 23, No. 2, pp. 1007–1024, 2020.

[17] G. Fathy, H. Hassan, S. Rahwan, and W. Sheta, "Parallel implementation of multiple kernel self-organizing maps for spectral unmixing", *Journal of Real-Time Image Processing*, Vol. 17, N.5, pp. 1267–1284, 2020.

[18] A. Ivanavicˇius, H. Simonavicˇius, J. Gelšvartas, A. Lauraitis, R. Maskeliu, P. Cimmperman, and P. Serafi-navicˇius, "Real-time cuda-based stereo matching using cyclops2 algorithm", *EURASIP Journal on Image and Video Processing*, No. 1, pp.1–15, 2018.

[19] J. Wang, H. Liu, L. Cong, Z. Xiahou, and L. Wang, "Cnn-monofusion: online monocular dense reconstruction using learned depth from single view", In: *Proc. of IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*, IEEE, pp. 57–62, 2018.

[20] X. Yang, L. Zhou, H. Jiang, Z. Tang, Y. Wang, H. Bao, and G. Zhang, "Mobile3drecon: real-time monocular 3d reconstruction on a mobile phone", *IEEE Transactions on Visual-Ization and Computer Graphics*, Vol. 26, No. 12, pp. 3446–3456, 2020.

[21] S. Shimada, V. Golyanik, W. Xu, and C. Theobalt, "Physcap: Physically plausible monocular 3d motion capture in real time", *ACM Transactions on Graphics (TOG)*, Vol. 39, No. 6, pp.1–16, 2020.

[22] H. Peng, C. Xian, and Y. Zhang, "3d hand mesh reconstruction from a monocular rgb image", *The Visual Computer*, Vol. 36, No. 10, pp. 2227–2239, 2020.

[23] F. Lu, G. Chen, Z. Li, L. Zhang, Y. Liu, S. Qu, and A. Knoll, "Monet: Motion-based point cloud prediction network", *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[24] F. Huang, H. Yang, X. Tan, S. Peng, J. Tao, and S. Peng, " Fast reconstruction of 3d point cloud model using visual slam on embedded uav development platform", *Remote Sensing*, Vol. 12, No. 20, pp. 3308, 2020.

[25] R. Ranftl, V. Vineet, Q. Chen, and V. Koltun, "Dense monocular depth estimation in complex dynamic scenes", In: *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 4058–4066, 2016.

[26] CUDA Nvidia. Cuda .Online http://www.nvidia.com/object/cuda_home_new,

15, 2006.

[27] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, “Vision meets robotics: The kitti dataset”, *The International Journal of Robotics Research*, Vol. 32, No. 11, pp. 1231–1237, 2013.

[28] M. Han and T. Kanade, “Multiple motion scene reconstruction with uncalibrated cameras”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 25, No. 7, pp. 884-894, 2003.

[29] R. M. Artal, J. Montiel, and J. Tardos, “Orb-slam: a versatile and accurate monocular slam system”, *IEEE Transactions on Robotics*, Vol. 31, No. 5, pp. 1147–1163, 2015.

[30] askgeek. NIVIDIA. Online : https://askgeek.io/en/gpus/vs/NVIDIA_Tesla-V100-PCIe-32-GB-vs-NVIDIA_Tesla-M10

[31] T. NVIDIA. Nvidia tesla v100 gpu architecture, 2017.