http: //doi.org/10.35784/iapgos.7291 *received: 15.02.2025 | revised: 30.05.2025 | accepted: 19.06.2025 | available online: 27.06.2025*

AN EFFICIENT OMNIDIRECTIONAL IMAGE UNWRAPPING APPROACH

Said Bouhend[1], Chakib Mustapha Anouar Zouaoui[2], Adil Toumouh[1], Nasreddine Taleb[2]
[1]Djillali Liabes University, Computer Science Department, Sidi Bel Abbes, Algeria, [2]Djillali Liabes University, Electronics Department, Sidi Bel Abbes, Algeria

Abstract. *Omnidirectional cameras are widely used almost everywhere especially in robotics and security. These kinds of cameras provide circular images that can be difficult to be interpreted by humans. Hence it must be transformed to be understandable by humans or to be treated with usual systems. This transformation is called unwrapping. The unwrapping may be time consuming and can decrease the performance of the real-time systems. Besides, the unwrapping can affect the quality of obtained images. To overcome these problems, we present an efficient parallel omnidirectional image unwrapping approach based on image partitioning. Experimental results indicate that our unwrapping approach has fast processing and gives better quality of unwrapped panoramic images.*

Keywords: omnidirectional image, image unwrapping, OpenCl, parallel programming

SKUTECZNA METODA ROZPAKOWYWANIA OBRAZU DOOKÓLNEGO

Streszczenie. *Kamery dookólne są szeroko stosowane niemal wszędzie, zwłaszcza w robotyce i bezpieczeństwie. Tego rodzaju kamery dostarczają okrągłe obrazy, które mogą być trudne do zinterpretowania przez ludzi. W związku z tym muszą one zostać przekształcone, aby były zrozumiałe dla ludzi lub mogły być przetwarzane przez zwykłe systemy. Transformacja ta nazywana jest rozpakowywaniem. Rozpakowywanie może być czasochłonne i może obniżyć wydajność systemów czasu rzeczywistego. Ponadto, rozpakowywanie może wpływać na jakość uzyskanych obrazów. Aby przezwyciężyć te problemy, przedstawiamy wydajne równoległe podejście do rozpakowywania obrazów dookólnych oparte na dzieleniu obrazów. Wyniki eksperymentów wskazują, że nasze podejście do rozpakowywania zapewnia szybkie przetwarzanie i lepszą jakość rozpakowanych obrazów panoramicznych*

Słowa kluczowe: obraz dookólny, rozpakowywanie obrazu, OpenCl, programowanie równoległe

Introduction

Nowadays, omnidirectional (OD) cameras have become very important in several areas, where a large field-of-view is needed, like in robotics and security. These kinds of cameras can give a complete view of 360° along one direction. Nevertheless the distorted images obtained by these cameras can be difficult to human vision. In addition the existing systems of computer vision can't deal with this kind of images. Usually, there is a need to create views for human comprehensibility. This task is called unwrapping. Regarding the topic, the unwrapping is an automatic matching between a rectangular image and a catadioptric one as shown in Fig. 1.

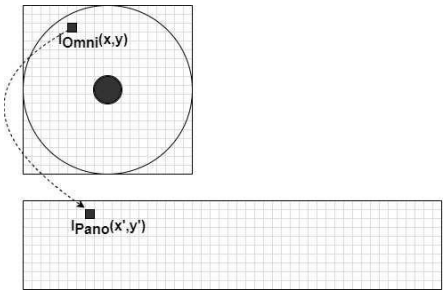

Fig. 1. The unwrapping process

A lot of research has dealt with OD images. According to the manner that these images are treated, omni-camera-based approaches can be categorized into two types, namely the direct use of distorted images and the use of unwrapped ones.

The first type deals with OD images as it is without any transformation. The authors in [14] introduced a new method to extract and match vertical lines between images taken by an omnidirectional camera installed on a wheeled robot. The method proposed in [15] uses optical flow and kernel particle filters to detect and track moving targets in omnidirectional vision. Here, the algorithms of optical flow and kernel particle filter are improved based on the polar coordinates. The authors in [3] have performed an object detection method directly on the omnidirectional images without converting them to panoramic or perspective images. They proposed a modified histogram of oriented gradients (HOG) features adapted to omni-images. Practically, this type of approach can avoid much time wasting in pre-treatment of data sources.

Nevertheless, other researchers have attempted to apply to omnidirectional images standard feature detectors and matching techniques, which have been traditionally employed for perspective images. In [9], the authors have converted the source image to a panoramic view, and then they have developed a method of detecting and tracking moving obstacles from omnidirectional data obtained by a mobile robot. In [10], person identification and tracking have been applied to panoramic images obtained from omni-camera. In [5], the authors have developed a method to detect humans in omnidirectional images based on motion analysis and segmentation. A simple unwrapping method is performed to convert an omnidirectional image into a panoramic image, and two consecutive panoramic images are used for motion analysis. Then the optical flow tracker is applied by tracking each feature in the previous frame to find the corresponding feature in the current frame.

This paper is structured as follows. Section 2 describes briefly some research works related to unwrapping omni-images. Section 3 describes the proposed image unwrapping. The experimental results are reported in Section 4 and followed by our conclusions.

1. Background

In the literature many approaches have been proposed to treat omnidirectional images. In fact, the simplest and widely used technique is based on image unwrapping. The unwrapping process creates a perspective image from the distorted one, and existing image processing methods can be applied. Researchers like in [17] propose log-polar mapping for the sampled images, from Cartesian coordinate systems (x; y) to log-polar coordinates (ρ; θ), where in [16], the combination of tangential and radial circle centers can avoid the unwrapped image distortion which cannot be eliminated through a single center parameter. The proposed method can also correct the frame aspect ratio of the unwrapped image and make the measurement results more reliable and accurate. The authors in [11] introduced a generalized unwrapping and rectification method for omnidirectional cameras when the camera parameters are not available.

In reality, the process of unwrapping is time-consuming and can decrease the real-time system performance especially where another image-processing task must be done on the transformed image. This need for accelerating the unwrapping process led many researchers to develop many methods in order to improve the performance. Some approaches like in [6, 11, 12, 17] use the look up tables named pano-mapping tables [7] to store pixel coordinates in order to be used in the transformation task (see Fig. 2).

The pano-mapping table is constructed only once, and it can be used to create panoramic images of the OD one. This is a simple and efficient way to unwrap an omni-image taken by any omni-camera without a need for the physical parameters of the omni-camera. The lookup tables based methods need large memory space in panoramic unwrapping and to reduce it, an eight directions symmetry reuse strategy is proposed in [1]. In [2], the authors propose a method to reduce this large computation load by exploiting the parallelism of graphical processing units (GPUs) based on the Compute Unified Device Architecture (CUDA).Although the unwrapping algorithm based on parallel calculation has the simplest transformation which uses unwrapping of the source image directly, there is no need to large space memory as proposed in [8].This method needs additional memory space and increases page faults. In our approach we work directly on the source without the need for any additional memory space.

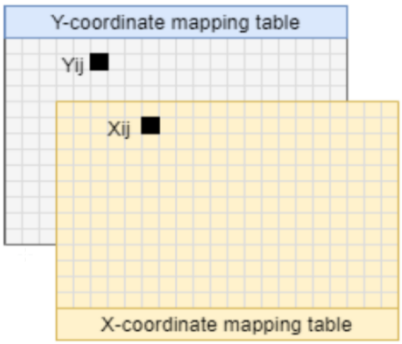

Fig. 2. Pano-mapping table

2. Proposed method

In this section, an efficient method for unwrapping images is presented. The main goal of unwrapping is to obtain a panoramic image from a circular image by applying geometric transformations to the source image. This can be performed sequentially or using parallel techniques. Our approach attempts to reduce as much as possible the device waiting time for more acceleration. Our main contribution is reducing the time of data transfer by partitioning the source image in four equal parts and transferring each part of the device(or devices) by applying the overlapping technique. Furthermore, our contribution is in the quality of the output images using the bi-cubic interpolation.

2.1. Mapping pixels

The procedure of mapping pixels can be done by the use of some geometric transformation. Each pixel of destination images can be obtained by a reverse mapping solution proposed in [8]. This mapping transformation is relevant for identification the panoramic-image pixels coordinates and it is based on its respective OD image pixels coordinates (see Fig. 3), and is expressed by equation 1.

$$\begin{cases} X_{pano} = X_{center} + \frac{1}{\propto} \times (Y_{omni} + r) \times cos(\delta) \\ Y_{pano} = Y_{center} + \frac{1}{\propto} \times (Y_{omni} + r) \times sin(\delta) \\ \delta = (X_{omni} + r) \times \frac{1}{\beta \times R} \end{cases} \quad (1)$$

where (X_{omni}, Y_{omni}) are the coordinates of the pixel on the OD image, and (X_{pano},Y_{pano}) are the respective coordinates of the pixel in the panoramic image, R and r are respectively the radius of internal and external circles of the interest area of OD picture, α and β are the panoramic image sizing parameters, and δ is the angle between line (P, O) which passes through the centre of the OD image and pixel P, and the line of the Cartesian coordinating system.

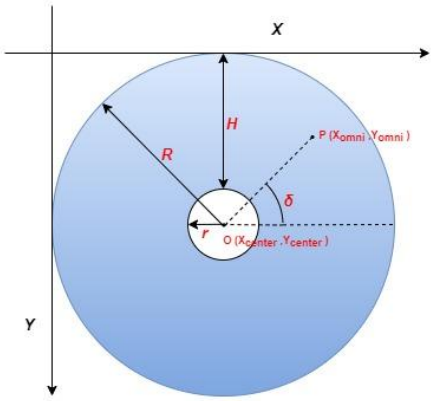

Fig. 3. Omnidirectional image geometry

2.2. Parallel programming

In a sequential approach, each pixel is processed one at a time, often in a nested loop (e.g., iterating over rows and columns of the image).This methods consume too much time and can't be useful for approaches that are used to perform other tasks after the unwrapping step.

In parallel programming, the image pixel mapping refers to the process of leveraging multiple processors or cores to process the individual pixels of an image simultaneously [8], as opposed to processing them one by one in a sequential manner. This approach can dramatically speed up tasks like image processing, especially for large images or complex operations.

Several frameworks and libraries make it easier to implement parallel programming for image pixel mapping, such as: OpenMP which is a popular API for parallel programming on multi-core processors and provides simple constructs for parallel loops, CUDA which is a parallel computing platform and application programming interface (API) model created by NVIDIA, specifically designed for GPUs, and TensorFlow and PyTorch which are learning frameworks that support GPU-based parallel computation, and therefore be leveraged for image processing tasks. OpenCL is a framework for writing programs that can execute across heterogeneous platforms, including CPUs and GPUs and has been used for implementing our approach.

When using GPU acceleration device, data transfer refers to the process of moving data between the CPU (central processing unit) and GPU memory to enable efficient parallel computation. At first, data must be transferred from CPU memory to GPU memory before the GPU can process it. Once computation is done, results are often transferred back from GPU memory to CPU memory for further use or analysis (see Fig. 4).

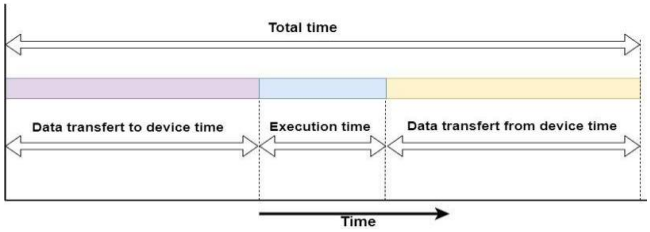

Fig.4. The steps of a job processed by acceleration devices

In parallel programming on GPUs, waiting time often refers to the idle periods where some threads or compute units are not actively working due to imbalanced workloads, inefficient memory access, or synchronization delays. When processing an image, the GPU often needs to read and write large amounts of data. If the images are too large, the data transfer of the original image can cause latency and waiting times. This problem can be solved by a technique named image partitioning, as we will be seen in the next section.

2.3. Image partitioning

Image partitioning in GPU programming is a technique that divides an image into smaller, more manageable sections (or partitions) to be processed concurrently by multiple threads or GPU cores. The source image partitioning is a technique used in several approaches such as in [8, 13]. The problem in using acceleration devices is time wasted in transferring data between principal memories of acceleration devices. This time can be a handicap for the system especially when the computational time is much faster than the data transfer. To fix this problem we use the data partitioning. This method may help to overlap data transfer with computation. In fact, we can split the image into multiple parts. The question is then how to choose the number of partitions.

The choice of the number of partitions of data is important, and it can affect the performance of the system. In this work, the source image is splitted into four parts (see Fig. 5), basing this choice on minimizing the needless data transferred to the device. The needless data is the data transferred to the device and that is not used by the device. In omnidirectional images only pixels inside the circle are used to create the panoramic one. This data is time-consuming when it is transferred to the device.

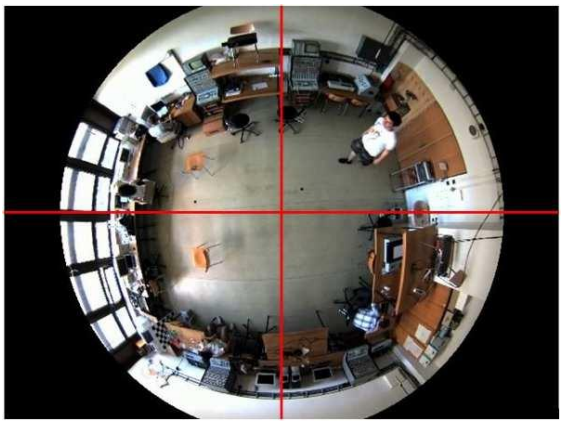

Fig. 5. Image partitioning

In our approach, we attempt to use multiple devices and at the same time minimize the waiting time of the device. As shown in equation 2, the needless data (Sneedless) is calculated by subtracting of the circular surface (SCircle) from the surface of transferred surface. In the flowing, we demonstrate that if we chose more than four parts, we increase the size of the needless data and consequently increase the data transfer time:

- **One(01) part:** In the original image(see Fig. 6) we can calculate the needless data using equation 2.

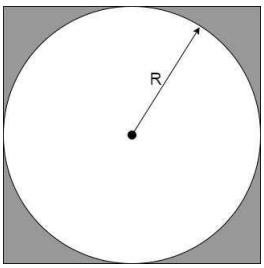

Fig. 6. Image in one piece

$$S_{needless} = S_{square} - S_{circle} \quad (2)$$

where:

$$S_{square} = 4 \times R^2 \quad (3)$$

$$S_{Circle} = \pi \times R^2 \quad (4)$$

by substitution in equation 2:

$$S_{needless} \cong 0.858 \times R^2 \quad (5)$$

The ideal way is to transfer only the pixels that are inside the circle. However, this is not possible due to the nature of the data structure of an image stored in memory. For example, in Figs. 9 and 10, in order to transfer a sector, we will have to copy the rectangle that covers this sector.

- **Two(02) parts:** When dividing the original image to two equal parts (see Fig. 7)witch their surface is calculated as:

$$S_{needless} = (2 * S_{squar_{part_1}}) - S_{circle} \quad (6)$$

where:

$$S_{square\_part\_i} = R^2 \times 2, \text{ i=1,2} \quad (7)$$

by substitution in equation 2:

$$S_{needless} \cong 0.858 \times R^2 \quad (8)$$

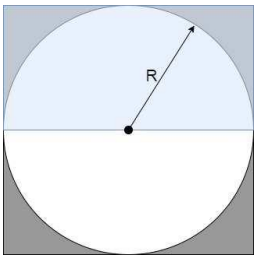

Fig. 7. Image partitioned into two parts

- **Four(04) parts:** When dividing the original image to four parts (see Fig. 8) the needless data is:

$$S_{needless} = 4 * S_{square\_part\_1} - S_{circle} \quad (9)$$

where:

$$S_{square\_part\_i} = R^2, \text{ i=1,2,3,4} \quad (10)$$

by substitution in equation 2:

$$S_{needless} \cong 0.858 \times R^2 \quad (11)$$

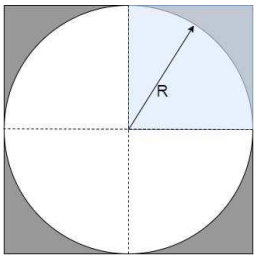

Fig. 8. Image in four pieces

Until here, we can see that the needles data remains the same as that in the two previous partitioning methods. However, the use of the overlapping method can make the difference. Indeed, the experiments (see section 4) shows that splitting the image into two parts is more efficient than using the original image and that partitioning it into four parts provides better results than the two others.

- **Eight(08) parts:** When dividing the original image to eight parts (see Fig. 9) the needless data is:

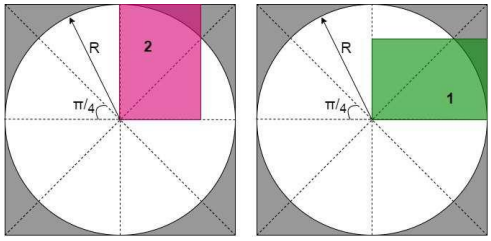

Fig. 9. Image partitioned in eight parts

When dividing the original image into eight parts, we have to transfer the square that cover sectors as shown in Fig. 9 and the needless data can be calculated as follow:

$$S_{square} = \left(S_{square\_part\_1} + S_{square\_part\_2}\right) \times 4 \quad (12)$$

$$S_{square\_part\_1} = S_{square\_part\_2} = R^2 \times \sin\left(\frac{\pi}{4}\right) \quad (13)$$

$$S_{needless} \cong 2.515 \times R^2 \quad (14)$$

- **Twelve(12) parts:** In this case(see Fig. 10), the needless data can be calculated as:

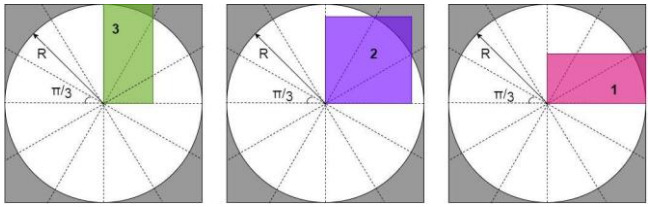

Fig. 10. Image partitioned in twelve parts

$$S_{needless} = (S_{square\_part\_1} + S_{square\_part\_2} + S_{square\_part\_3}) \times 4$$

$$S_{square\_part\_1} = S_{square\_part\_3} = R^2 \times \sin(\frac{\pi}{6}) \quad (16)$$

$$S_{square\_part\_2} = (R \times \cos(\frac{\pi}{6}))^2 \quad (17)$$

$$S_{needless} \cong 3.85 \times R^2 \quad (18)$$

By comparing equations (5), (8) and (11) we can see that the needless data to be transferred to the device is the same when the number of parts is under or equal to 4. However this data increase when the number of parts is higher than 4 as seen in equations (14) and (18).

Consequently, in our approach we adopted to divide the images in 4 parts.

2.4. Interpolation

Based on the affine transformation model, the panoramic image can be unwrapped after pixel replication. However, there is no exact one-to-one correspondence between the calculated pixels and those of the unwrapped image. In order to solve this problem, an interpolation algorithm is used to replicate the pixel values.

The most of the existent approaches use the linear interpolation such as [17] or the bilinear one like in [2, 8, 12, 16]. In our approach we used the bi-cubic interpolation (see Fig. 11) that gives a high output quality to estimate a pixel P in the OD image using its 16 adjacent pixels. The colour value of P is calculated using 16 these pixels according to their distances to P. Due to the need of these 16 pixels and as mentioned above, we have in the current paper adjusted the height and width of source parts to guarantee some smoothing between the destination image parts and giving better image quality and minimizing the loss of image information.

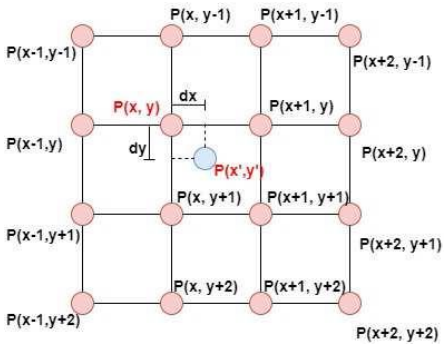

Fig. 11. Bi-cubic interpolation

2.5. Minimizing the loses of data

Minimizing data loss in image partitioning is a critical consideration when processing large images on a GPU. Image partitioning involves dividing an image into smaller sections to process them concurrently. However, this partitioning can introduce potential risks of data loss or inaccuracies, particularly when the image processing algorithms require information from neighbouring pixels. In these cases, careful management of how image sections are divided and processed is essential to ensure no data is lost or corrupted during computation

In our case, partitioning an image may create some loss of data, exactly in the partitioning borders. During the mapping process, the approximated pixels are obtained by applying the interpolation method to each part of the source image. In bi-cubic interpolation, the 16 pixels are needed to approximate the destination pixel and this cannot be done correctly when we want approximately first (last) line or first (last) colon. Our proposed solution is to adjust the width and height of parts according to the interpolation method. In our case, we used the interpolation bi-cubic, so we added 4 pixels for the width and height. This can help to avoid the loss of smoothing (see Fig. 12).

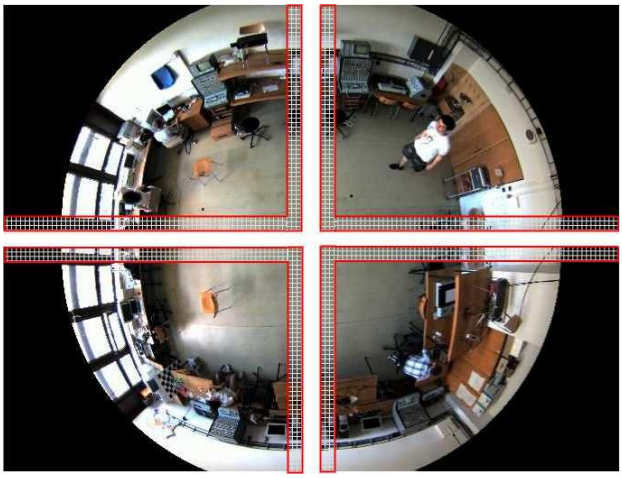

Fig. 12. The four (04) parts of original image enlarged

2.6. Overlapping process

The overlapping technique (see Fig. 13) in GPUs refers to the practice of simultaneously executing multiple operations to maximize hardware utilization and improve overall performance. This is achieved by overlapping computation with data transfer or by interleaving the execution of different tasks. Without overlapping, these tasks would be done one after the other:

1) First, data are transferred to the GPU.
2) Then, the device executes the kernel.
3) After computation, the results are transferred back to the host.

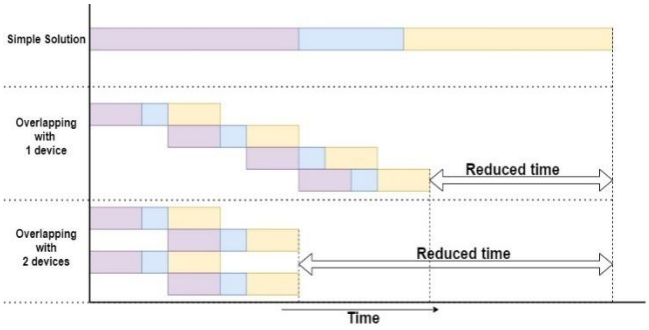

Fig. 13. Overlapping process

The overlapping technique allows the GPU to transfer data and run computations simultaneously by using asynchronous operations. OpenCL provides a feature called command queues that allow us to run operations asynchronously. In our approach, this means, each part of the original image is treated in separate queues, allowing them to run at the same time if resources allow.

Three steps are needed to Achieve Overlapping Data Transfer and Computation in OpenCL:

- First: Create four Command Queues: One for one part of the source image.
- Next: Use Asynchronous Memory Transfer: OpenCL supports asynchronous memory transfer functions like clEnqueueWriteBuffer and clEnqueueReadBuffer which

do not block the program's execution, allowing other tasks (like running kernels) to proceed while data is being transferred.

- Finally: Run the Kernel: Once the data is transferred (or while it's still being transferred), we can enqueue the kernel to be executed.

2.7. Sample OpenCL code

The code in Fig. 14 uses a command queue containing the steps of a job performed by device: read, write and execute. clEnqueueWriteBuffer is a function used to transfer data to the device asynchronously, this is possible by using the flag CL_FALSE, which means that the transfer is non-blocking. Similarly, clEnqueueReadBuffer can transfer data back from the device to the host asynchronously. The function clEnqueueNDRangeKernel runs the kernel on the GPU and while the kernel is running; the GPU can still transfer data in the background if there is no dependency between the data and computation. To synchronize all operations, the function clFinish waits for all tasks in the queue to finish before proceeding. This ensures that the program doesn't end until all work is done.

```
// Step 1: Create command queues
cl_command_queue queue1 =
clCreateCommandQueue(context, device, 0, &err);
cl_command_queue queue2 =
clCreateCommandQueue(context, device, 0, &err);

// Step 2: Asynchronously transfer data to the device
clEnqueueWriteBuffer(queue1, buffer, CL_FALSE, 0,
bufferSize, hostData, 0, NULL, NULL);

// Step 3: Run the kernel on the GPU (while data is
being transferred)
clEnqueueNDRangeKernel(queue2, kernel, 1, NULL,
globalWorkSize, localWorkSize, 0, NULL, NULL);

// Step 4: Asynchronously transfer data back to the
host (while computation is happening)
clEnqueueReadBuffer(queue1, buffer, CL_FALSE, 0,
bufferSize, hostData, 0, NULL, NULL);

// Step 5: Wait for all operations to complete
clFinish(queue1);
clFinish(queue2);
```

Fig. 14. Example of OPENCL code of overlapping process

For maximum performance, multiple devices may be used to perform the process of unwrapping. In the current approach, we used two devices to boost acceleration. After the image is split, the parts must be transferred to the devices to perform the unwrapping process. By applying the overlapping technique, we hide the memory transfer time between the host and the device as much as possible. Here, the data is transferred between the host and the device as shown in Fig. 13, while some of the regions are in GPU being executed.

2.8. Algorithm

In this section, we describe the algorithm underlying our approach. Our approach can be applied to one or multiple devices. In the case of a single device (see Fig. 15), one part is first transferred to the device, after which the kernel execution is initiated. Once the device completes processing, the first part, the second part is transferred to the device while the first part is simultaneously transferred back to the host. Similarly, parts 3 and 4 are processed in the same manner as parts 1 and 2.

Using two devices provides better performance, as demonstrated in the experiment section. In the previous case, each device processed all four jobs. With two devices, however, each device processes two jobs(see Fig. 16), and each job consists of two parts. Theoretically, this approach reduces the execution time to the duration required to process two parts.

```
// Create the context
Context = CreateContext(Device);
// Create the kernel
Kernel  = CreateKernel(Context);
// Divide source image into 4 parts
InputParts[] =
PartitionFrameInto4Parts(SourceImage);
// Copy the 1st part to device
CopyDataToDevice(InputParts[1], Device);
// Execute the mapping process for the 1st part
// on the device
ExecuteKernelOnDevice(Device,Kernel);
// In the same time:
//   - Copy the 2nd part to the device
//   - Copy the 1st mapped part from the device
//   to memory
CopyDataToDevice(InputParts[2]; Device)
OutputParts[1] = CopyDataFromDevice(Device)
// Execute the mapping process for the 2nd part
// on the device
ExecuteKernelOnDevice(Device,Kernel);
// In the same time:
//   - Copy the 3rd part to the device
//   - Copy the 2nd mapped part from the device
//   to memory
CopyDataToDevice(InputParts[3]; Device)
OutputParts[2] = CopyDataFromDevice(Device)
// Execute the mapping process for the 3nd part
// on the device
ExecuteKernelOnDevice(Device,Kernel);
// In the same time:
//   - Copy the 4th part to the device
//   - Copy the 3rd mapped part from the device
//   to memory
CopyDataToDevice(InputParts[4]; Device)
OutputParts[3] = CopyDataFromDevice(Device)
// Execute the mapping process for the 4th part
// on the device
ExecuteKernelOnDevice(Device,Kernel);
// Copy the 4th mapped part from the device
// to memory
OutputParts[4] = CopyDataFromDevice(Device)
// concatenate the four mapped parts
NewFrame = ConcatenateParts(OutputParts)
```

Fig. 15. Unwrapping algorithm with one device

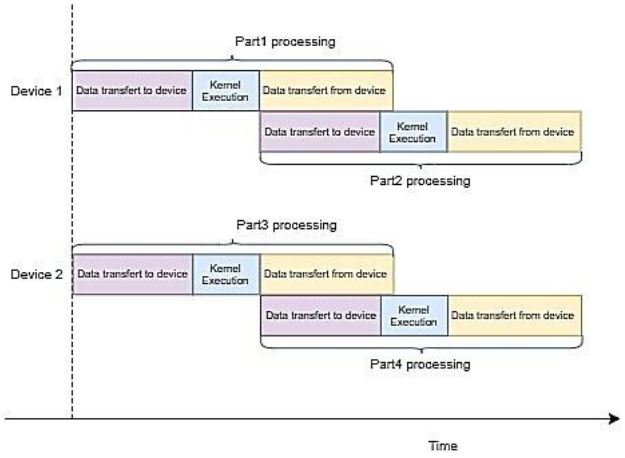

Fig. 16. Overlapping process using tow device

The algorithm used with two devices (see Fig. 17) is the same as that used for a single device, except that each device processes two parts of the original image. The transfer of the first and third parts is initiated simultaneously. Similarly, the execution of Kernels 1 and 2 occurs in parallel. Next, the transfer of parts 2 and 4 to the devices and the transfer of the mapped parts back to the host are initiated simultaneously, which may significantly reduce waiting time. Finally, the four parts are concatenated to form the resulting panoramic image.

```
// Create the two contexts
Context1 = CreateContext(Device1)
Context2 = CreateContext(Device2)
// Create the two Kernels
Kernel1 = CreateKernel(Context1)
Kernel2 = CreateKernel(Context2)
// Divide the source image into for parts:
InputParts = PartitionFrameInto4Parts(InputImage)
// in the same time
//  - Copy the 1st part to the device 1
//  - Copy the 3rd part to the device 2
CopyDataToDevice(InputParts[1]; Device1)
CopyDataToDevice(InputParts[3]; Device2)
// in the same time
//  - Executing mapping process of the 1st part
//  on the device 1
//  - Executing mapping process of the 3rd part
//  on the device 2
ExecuteKernelOnDevice(Device1,Kernel1)
ExecuteKernelOnDevice(Device2,Kernel2)
// In the same time:
//   - Copy the 2nd part to the device
//   - Copy the 1st mapped part from the device
//   to memory
//   - Copy the 4th part to the device
//   - Copy the 3rd mapped part from the device
//   to memory
CopyDataToDevice(InputParts[2]; Device1)
OutputParts[1] = CopyDataFromDevice(Device1)
CopyDataToDevice(InputParts[4]; Device2)
OutputParts[3] = CopyDataFromDevice(Device2)
// in the same time
//  - Executing mapping process of the 2nd part
//  on the device 1
//  - Executing mapping process of the 4th part
//  on the device 2
ExecuteKernelOnDevice(Device1,Kernel1)
ExecuteKernelOnDevice(Device2,Kernel2)
// In the same time:
//   - Copy the 2nd mapped part from the device 1
//   to memory
//   - Copy the 4th mapped part from the device 2
//   to memory
OutputParts[2] = CopyDataFromDevice(Device1)
OutputParts[4] = CopyDataFromDevice(Device2)
// Concatenate the four mapped parts
NewFrame = ConcatenateParts(OutputParts)
```

Fig. 17. Unwrapping algorithm with tow devices

3. Experiments and results

In this experiment, we used a laptop with I7 processor, 8 Gb of memory with two acceleration devices and rung Windows 8.1 64 bits as shown in table 1.

Table 1. Hardware specification

designation	information
processor	I7 CU6500
Memory	8Gb
Acc. device 1	Intel HD 520
Acc. device 2	AMD 6800
Platform	OpenCl
Os	Microsoft Windows 8.1

We used the Bomni dataset [14] for our experiments. Bomni-DB comprises videos recorded in a room with two omnidirectional cameras, where the bounding boxes and actions of individuals are annotated. The omnidirectional cameras are positioned at the top and side of the room. In this study, we utilized the top camera (see Fig. 18).

In order to compare between different types of interpolation, we developed three kernels each one for one type of interpolation: clMappingNearest, clMappingLienear and clMappingBicubic. The following is the header of the kernel:

```
__kernel void clMappingBicubic(
    __global const unsigned char *srcImage,
    __global unsigned char *dstImage,
    __global int *width,
    __global int *hieght,
    __global int *part_number)
```

Fig. 18. Omnidirectional image from BomniDataset

The dataset used in our experiments consists of videos with 907 frames, each with a resolution of 640×480 pixels. Each frame is treated individually as a single image. Initially, the source image is divided into four parts, with additional pixels added along the borders. These enlarged border pixels ensure that bicubic interpolation produces accurate values. Next, two devices are initialized, each receiving two parts of the divided image. The devices execute their tasks simultaneously. Finally, the resulting panoramic parts are concatenated to form the final panoramic image. Fig. 19 provides a sample of a panoramic image obtained from a source image. The resolution of the panoramic image is 224×1568 pixels, demonstrating high quality and smoothness.

Fig. 19. Panoramic image obtained by unwarpping OD image

The basic approach used to evaluate our work is the image unwarping without splitting image and without using the overlapping technique (see table 2).

Table 2. Basic approach without splitting the image

interpolation	AMD 8600M (1 job)	Intel HD520 (1 job)
Nearest	25,92 fps	28.34 fps
Linear	21.59 fps	25.91 fps
Bi-cubic	17.44 fps	20.15 fps

To provide further comparison, we applied our approach using a single device, and the results are presented in table 3. A last, comparison with table 2 shows that our approach achieves high performances for unwrapping omnidirectional (OD) images, processing more than 351 frames per second. The Nearest interpolation can particularly be useful for systems requiring low image quality. However, with the bi-cubic interpolation, our approach delivers better performances while providing enhanced image quality.

Table 3. Experiment results for image resolution 640×480

Interpolation	AMD 8600M (4 jobs)	Intel HD520 (4 jobs)	**Intel HD520 (2 jobs) AMD 8600M(2 jobs)**
Nearest	89.94 fps	187.12 fps	**351.58 fps**
Linear	72.18 fps	156.93 fps	**302.15 fps**
Bi-cubic	44.57 fps	102.08 fps	**186.91 fps**

We also tested our solution with videos of different sizes. Since the BOMNI dataset provides only one size, we created new videos by resizing the source videos. We then applied our approach to these resized videos, obtaining the results shown in tables 4 and 5.

Table 4. Experiment results for image resolution 960×720

Interpolation	AMD 8600M (4 jobs)	Intel HD520 (4 jobs)	**Intel HD520 (2 jobs) AMD 8600M(2 jobs)**
Nearest	79.49 fps	137.76 fps	**312.11 fps**
Linear	67.65 fps	102.11 fps	**265.75 fps**
Bi-cubic	35.06 fps	57.39 fps	**139.44 fps**

Table 5. Experiment results for image resolution 1280×960

Interpolation	AMD 8600M (4 jobs)	Intel HD520 (4 jobs)	**Intel HD520 (2 jobs) AMD 8600M(2 jobs)**
Nearest	53.75 fps	88.93 fps	**214.24** fps
Linear	42.69 fps	67.39 fps	**169.99** fps
Bi-cubic	20.89 fps	35.55 fps	**83.56** fps

Tables 3, 4, and 5 demonstrate that our approach is both more efficient and more effective across various image resolutions. Additionally, as shown in Fig. 19, the quality of the resulting image is improved, with smoothness being well-preserved. This improvement is attributed to the use of bi-cubic interpolation and our solution, which involves adding pixels during the splitting step to prevent data loss.

4. Conclusion

Parallel programming for image pixel mapping enhances performance by dividing the task of pixel manipulation into smaller, concurrent jobs, thereby optimizing the time required to process large or complex images. Image partitioning is an effective technique in GPU programming as it facilitates efficient parallelization and memory management. By dividing an image into smaller partitions, the GPU can process different sections concurrently, improving overall performance and ensuring optimal utilization of GPU resources. The proposed parallel method for omnidirectional (OD) image unwrapping on GPUs demonstrates that combining data partitioning with overlapping across multiple devices enhances the unwrapping process. This approach is beneficial for parallelizing works that aims to adapt existing image processing methods for applied to omnidirectional images. Additionally, our image partitioning technique reduces the risk of data loss and prevents data corruption when using bi-cubic interpolation. In future work, we aim to introduce further optimization techniques, such as leveraging local memory and the possibility to use more devices.

Acknowledgments

The research being reported in this publication was supported by the Algerian Directorate General for Scientific Research and Technological Development (DGRSDT).

References

[1] Chen L. D., Zhang M. J., Xiong Z. H.: Series–parallel pipeline architecture for high-resolution catadioptric panoramic unwrapping. IET image processing 4(5), 2010, 403–412.
[2] Chong N. S., Wong M. D., Kho Y. H.: Accelerated catadioptric omnidirectional view image unwrapping processing using GPU parallelisation. Journal of Real-Time Image Processing 12, 2016, 55–69.
[3] Cinaroglu I., Bastanlar I.: A direct approach for object detection with catadioptric omnidirectional cameras. Signal, Image and Video Processing 10(2), 2016, 413–420.
[4] Demiröz B. E., et al.: Feature-based tracking on a multi-omnidirectional camera dataset. 5th International Symposium on Communications, Control and Signal Processing. IEEE, 2012, 1–5
[5] Hariyono J., Hoang V. D., Jo K. H.: Human detection from omnidirectional camera using feature tracking and motion segmentation. Intelligent Information and Database Systems: 7th Asian Conference, ACIIDS 2015, Bali, Indonesia, March 23-25, 2015, Proceedings, Part II 7. Springer International Publishing, 2015, 329–338
[6] Jeng S. W., Tsai W. H.: Using pano-mapping tables for unwarping of omni-images into panoramic and perspective-view images. IET Image Processing 1(2), 2007, 149–155.
[7] Jeng S. W, Tsai W. H.: A unified approach to unwarping of omni-images into panoramic and perspective-view images using pano-mapping tables. IEEE International Conference on Systems, Man and Cybernetics 3, 2006, 2401–2406.
[8] Kepucka E., Gurcan I., Temizel A.: Fast Omnidirectional Image Unwrapping on GPU. Euromicro International Conference on Parallel, Distributed and Network-Based Computing (WIP), 2012.
[9] Koyasu H., Miura J., Shirai Y.: Real-time omnidirectional stereo for obstacle detection and tracking in dynamic environments. IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the Next Millennium 1, 2001, 31–36.
[10] Kristou M., Ohya A., Yuta S. I.: Target person identification and following based on omnidirectional camera and LRF data fusion. Ro-Man. IEEE, 2011, 419–424.
[11] Lazarenko V. P., Yarishev S., Korotaev V.: The algorithm for generation of panoramic images for omnidirectional cameras. Proc. SPIE 9530, 2015, 165–173.
[12] Lei J., et al.: Unwrapping and stereo rectification for omnidirectional images. Journal of Zhejiang University-Science A 10(8), 2009, 1125–1139.
[13] Requena M. J. M., Moscato P., Ujaldón M.: Efficient data partitioning for the GPU computation of moment functions. Journal of Parallel and Distributed Computing 74(1), 2014, 1994–2004.
[14] Scaramuzza D., et al.: Robust feature extraction and matching for omnidirectional images. Field and Service Robotics: Results of the 6th International Conference. Springer, Berlin Heidelberg 2008, 71–81.
[15] Shu-Ying Y, WeiMin G., Cheng Z.: Tracking unknown moving targets on omnidirectional vision. Vision Research 49(3), 2009, 362–367.
[16] Wang X., et al.: Corrected unwrapping method based on the tangential and radial distortion centre for the panoramic annular image. IET Image Processing 9(2), 2015, 127–133.
[17] Xuepeng H.: A panoramic unwrapping approach based on log-polar mapping. 2nd International Conference on Intelligent Information Processing. 2017, 1–4.

M.Sc. Eng. Said Bouhend
e-mail: bouhendsaid@gmail.com

He received an engineer degree and a magister's degree in computer science from Djillali Liabes University, Sidi Bel Abbes, Algeria. He is a currently a computer science teacher while working on his doctoral thesis at Djillali Liabes University.
His research interests include image/video processing and parallel computing.

https: //orcid.org/0000-0002-5234-1849

Ph.D. Eng. Chakib Mustapha Anouar Zouaoui
e-mail: chakib@ipatdz.info

He received an engineer degree and a magister's degree in electrical engineering, as well as a Ph.D. degree in parallel and distributed computing, all from Djillali Liabes University, Sidi Bel Abbes, Algeria. He is currently an assistant professor in the Department of Electronics, Faculty of Electrical Engineering, Djillali Liabes University. As a member of the "Communication Networks, Architecture, and Multimedia" Laboratory, his principal research interests are in the fields of compilation, parallel and distributed computing, operating systems, and networking. He is also an International Expert and Solutions Architect in networking, security, and data centers.

https: //orcid.org/0000-0001-5176-4623

Ph.D. Eng. Adil Toumouh
e-mail: toumouh@gmail.com

He received an engineer degree, a magister's degree and a doctorate degree, all in computer science, from Djillali Liabes University, Sidi Bel Abbes, Algeria. He is currently an associate professor in the Computer Science Department at the Faculty of Exact Sciences, Djillali Liabes University. His research interests include artificial intelligence, knowledge engineering, text mining, and natural language processing (NLP).

https: //orcid.org/0009-0001-2604-6627

Prof. Nasreddine Taleb
e-mail: ne_taleb@yahoo.com

He received an M.Sc. degree in computer engineering from Boston University, an Elect-Eng. degree from Northeastern University, and a Ph.D. degree in electrical engineering from Djillali Liabes University, Sidi Bel Abbes, Algeria. He is currently a professor at the faculty of electrical engineering at Djillali Liabes University, where he has been teaching since 1990 and where he is also the director of the "Communication Networks, Architecture, and Multimedia" research laboratory. His principal research interests are in the fields of digital signal and image processing, image analysis, medical and satellite image applications, and advanced architectures for implementation of DSP/DIP applications. Pr. Taleb is an IEEE senior member.

https: //orcid.org/0000-0002-9688-3834