



UNIVERSIDAD
DE MÁLAGA

Escuela de Ingenierías Industriales

Programa de Doctorado
en Ingeniería Mecatrónica

TESIS DOCTORAL

Novel Parallel Approaches to Efficiently Solve Spatial Problems on Heterogeneous CPU-GPU Systems

Andrés Jesús Sánchez Fernández

Enero 2022

Dirigida por:

Dr. Luis Felipe Romero Gómez

Dr. Gerardo Bandera Burgueño


UNIVERSIDAD
DE MÁLAGA





UNIVERSIDAD
DE MÁLAGA

AUTOR: Andrés Jesús Sánchez Fernández

 <https://orcid.org/0000-0001-6743-3570>

EDITA: Publicaciones y Divulgación Científica. Universidad de Málaga



Esta obra está bajo una licencia de Creative Commons Reconocimiento-NoComercial-SinObraDerivada 4.0 Internacional:

<http://creativecommons.org/licenses/by-nc-nd/4.0/legalcode>

Cualquier parte de esta obra se puede reproducir sin autorización pero con el reconocimiento y atribución de los autores.

No se puede hacer uso comercial de la obra y no se puede alterar, transformar o hacer obras derivadas.

Esta Tesis Doctoral está depositada en el Repositorio Institucional de la Universidad de Málaga (RIUMA): riuma.uma.es





DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD DE LA TESIS PRESENTADA PARA OBTENER EL TÍTULO DE DOCTOR

D./Dña ANDRÉS JESÚS SÁNCHEZ FERNÁNDEZ

Estudiante del programa de doctorado INGENIERÍA MECATRÓNICA de la Universidad de Málaga, autor/a de la tesis, presentada para la obtención del título de doctor por la Universidad de Málaga, titulada: NOVEL PARALLEL APPROACHES TO EFFICIENTLY SOLVE SPATIAL PROBLEMS ON HETEROGENEOUS CPU-GPU SYSTEMS

Realizada bajo la tutorización de LUIS FELIPE ROMERO GÓMEZ y dirección de LUIS FELIPE ROMERO GÓMEZ Y GERARDO BANDERA BURGUEÑO (si tuviera varios directores deberá hacer constar el nombre de todos)

DECLARO QUE:

La tesis presentada es una obra original que no infringe los derechos de propiedad intelectual ni los derechos de propiedad industrial u otros, conforme al ordenamiento jurídico vigente (Real Decreto Legislativo 1/1996, de 12 de abril, por el que se aprueba el texto refundido de la Ley de Propiedad Intelectual, regularizando, aclarando y armonizando las disposiciones legales vigentes sobre la materia), modificado por la Ley 2/2019, de 1 de marzo.

Igualmente asumo, ante a la Universidad de Málaga y ante cualquier otra instancia, la responsabilidad que pudiera derivarse en caso de plagio de contenidos en la tesis presentada, conforme al ordenamiento jurídico vigente.

En Málaga, a 11 de SEPTIEMBRE de 2021

Fdo.: ANDRÉS JESÚS SÁNCHEZ FERNÁNDEZ Doctorando/a	Fdo.: LUIS FELIPE ROMERO GÓMEZ Tutor/a
Fdo.: LUIS FELIPE ROMERO GÓMEZ Y GERARDO BANDERA BURGUEÑO Director/es de tesis	





Autorización para la lectura de la Tesis e Informe de la utilización de las publicaciones que la avalan

Los abajo firmantes declaran, bajo su responsabilidad, que autorizan la lectura de la tesis del doctorando D. Andrés Jesús Sánchez Fernández con DNI tulada “Novel Parallel Approaches to Efficiently Solve Spatial Problems on Heterogeneous CPU-GPU Systems” y que ninguna de las publicaciones que avalan dicha tesis han sido utilizadas en tesis anteriores.

Málaga, a 11 de septiembre de 2021

Luis Felipe Romero Gómez.
Tutor y Director de la tesis.

Gerardo Bandera Burgueño.
Director de la tesis.

Dr. D. Luis Felipe Romero Gómez.
Catedrático del Departamento de
Arquitectura de Computadores.
Universidad de Málaga.

Dr. D. Gerardo Bandera Burgueño.
Profesor Titular del Departamento de
Arquitectura de Computadores.
Universidad de Málaga.

CERTIFICAN:

Que la memoria titulada *Novel Parallel Approaches to Efficiently Solve Spatial Problems on Heterogeneous CPU-GPU Systems* ha sido realizada por D. Andrés Jesús Sánchez Fernández , bajo nuestra dirección en el Departamento de Arquitectura de Computadores de la Universidad de Málaga, y constituye la Tesis que presenta para optar al grado de Doctor en Ingeniería Mecatrónica.

Málaga, enero de 2022



Dr. D. Luis Felipe Romero Gómez.
Tutor y codirector de la tesis.

Dr. D. Gerardo Bandera Burgueño.
Codirector de la tesis.

A mi familia, amigos y compañeros

Agradecimientos

Durante estos cuatro años de duro trabajo he tenido el placer de conocer y convivir con muchas personas que han aportado, en mayor o menor medida, su granito de arena para hacer que hoy me encuentre redactando estas líneas. Solo tengo palabras de agradecimiento por todo el apoyo que he recibido en los malos momentos y todas las alegrías que hemos compartido. Muchas gracias.

En el ámbito académico, quiero dar las gracias a mis directores Felipe y Gerardo por el apoyo, la dedicación, los consejos y el tiempo invertido en mí durante estos años; habéis sido piezas fundamentales en el desarrollo de esta tesis. Además, me habéis enseñado lo que es verdaderamente un trabajo de investigación y todo lo que puede aportar a la sociedad.

También me gustaría agradecer a Óscar, Rafa y Nico la oportunidad que me dieron—hace unos años ya—de poder comenzar el doctorado en el Departamento de Arquitectura de Computadores, sintiéndome siempre uno más del grupo de investigación. De igual forma, me gustaría dar las gracias a todos los profesores del DAC que he conocido durante esta etapa por todo lo que he podido aprender de ellos; a Carmen, por ser la mejor secretaria que existe; y a los técnicos de laboratorio, Paco y Juanjo, por toda la paciencia que tienen con nosotros y por estar siempre con una sonrisa cuando se les necesita. Sois increíbles.

El doctorado además me ha dado la oportunidad de cumplir un sueño que tenía desde niño: dar clases en la universidad. Y no estaba solo, sino en la mejor compañía de la mano de Mario y Fran, dos profesores increíbles con muchísima paciencia. Nunca olvidaré esos exámenes virtuales donde ocurrían las cosas más surrealistas que nos pudiéramos imaginar. Eso sí, me quedo con la espinita de no haber podido enseñar LaTeX a los estudiantes de podología, pero al menos ahora saben un poquito de Word, PowerPoint y Access (o eso creemos).

Fuera de la Universidad de Málaga, quiero dar las gracias a Siham por siempre estar ahí para ayudar y por la gran cantidad de consejos que me ha ido dando

durante estos años para mejorar como investigador. También, cruzando el charco, me gustaría agradecer a Miguel Ángel todo el tiempo que invirtió para sacar adelante el que fue mi primer artículo de revista.

And, of course, I do not forget all the people I met during my stay in Belgium: Dani, Yvan, Joris, Robin, Maxim, Laure, Katrien, Helena, Quentin, Annelies, Sofie, Niels and all the other people I could forget to mention. Special thanks to Dani for being my host when I arrived and who became a good friend over the months. You all have a great working environment that translates into outstanding achievements. Also, I felt at home from the very first day, both in the research group and in the beautiful city of Ghent, where I met many wonderful people that I will always carry with me. I cannot wait to come back.

Tampoco me olvido de mis compañeros, con los que he compartido los mejores momentos de esta etapa. Me gustaría empezar con mis compañeros de promoción: José Carlos e Iván. Los tres empezamos con mucha ilusión este camino tan duro y con muy poca idea de dónde nos metíamos, pero nos fuimos acostumbrando y creciendo, siempre ayudándonos y apoyándonos. Muchas gracias a los dos y mucho ánimo, que ya estamos muy cerca del final y podremos decir dentro de poco que somos doctores.

También querría agradecer especialmente a Bernabé y Cristina, en primer lugar, por haberme dado la oportunidad de poder vivir en Málaga y, sobre todo, por ser como sois y haberme aguantado tan bien durante los tres meses de cuarentena encerrados en el piso. Eso sí que tiene mérito y no el hacer una tesis doctoral. Muchas gracias por todas las risas y buenos momentos.

Pensando en los compañeros de esta etapa, la verdad es que uno se da cuenta de la suerte que ha tenido al estar rodeado de personas tan buenas en todos los aspectos. Me vienen recuerdos muy bonitos como los almuerzos todos juntos en la “sala Comanche”, esos martes de salir a comer por Teatinos, los cumpleaños celebrados con las mejores tartas, las dos cenas de navidad en las que he estado y que por las bromas de Ricardo y Fran casi me voy antes de tiempo y muchos otros momentos alegres que ahora mismo no recuerdo pero que siempre llevaré conmigo. Gracias por toda la ayuda, sois geniales y me gustaría nombraros a todos aquí: Fran, Villegas, Ricardo, Gloria, Denisa, Pedrero, Herruzo, Rubén, Felipe, Paula, Jimmy, Bernabé, José Carlos, Iván y aquellos que haya podido olvidar.

En el ámbito personal, quiero agradecer a Elena y su familia por el apoyo recibido todos estos años. Mi Elena, no hay palabras para agradecerte todo lo que has hecho por mí. Realmente, si estoy hoy escribiendo esto es en gran parte por haber recorrido este camino juntos. Es bonito que, a pesar de no entender

muy bien lo que hacía, ni por qué pasaba tantas horas en el ordenador, siempre has estado a mi lado. Por todos esos días que te he dicho que me tengo que poner con la tesis y me has entendido. Gracias cariño, dentro de nada no volverás a escuchar esas dichosas palabras y seremos doctores los dos. Y no me puedo olvidar de Cooper, mi pequeño compañero de cuatro patas, el mismo que se ha pasado dormido muchas horas sobre mi escritorio mientras escribía esta memoria.

A mis mejores amigos, los de Marbella de toda la vida, con los que he crecido y los que, a su manera, me han ayudado a llegar hasta aquí. Esto va por vosotros: Moya, Chema, Medina y compañía. A los que conocí más tarde, pero sé que estarán ahí siempre: Tabi, Edu, Rafa, María y los demás. A todas aquellas personas que he conocido en Málaga y que han pasado a formar parte de mi vida. A mis compañeros de carrera y viajes: Pedro, Tono, Manu y Diego, por todos los buenos momentos y por los otros muchos que vendrán. A mi tutor de TFG y TFM, Carlos Pérez del Pulgar, por introducirme en el mundo de la investigación y darme la oportunidad de formar parte del proyecto que resultó en la publicación de mi primer artículo científico.

Por supuesto, a mi familia y especialmente, como no, a mis padres y mi hermano. Creo que ellos tienen más ganas que yo de que acabe el doctorado. De verdad, soy incapaz de plasmar con palabras todo lo que habéis hecho por mí, todo el apoyo que me habéis dado siempre, los consejos, la fuerza e inculcarme ese coraje que se necesita para sacar adelante proyectos como este. Es imposible mencionar todo, gracias de corazón.

Y qué decir de mis abuelos, todo lo que he logrado y alguna vez lograré se lo debo a ellos. Por cuidarme cuando era pequeño, hacerme crecer en valores y enseñarme a ser una buena persona, aunque sé que nunca podré llegar a serlo tanto como ellos. Ojalá teneros aquí para que leyeráis estas palabras.

Finalmente, mencionar las fuentes de financiación que han dado pie al desarrollo de esta tesis y a poder realizar la estancia de investigación en el extranjero: el proyecto TIN2016-80920-R del antiguo Ministerio de Economía, Industria y Competitividad (Gobierno de España) y el I Plan Propio de Investigación, Transferencia y Divulgación Científica de la Universidad de Málaga.

Abstract

In recent years, approaches that seek to extract valuable information from large datasets have become particularly relevant in today's society. In this category, we can highlight those problems that comprise data analysis distributed across two-dimensional scenarios called spatial problems. These usually involve processing (i) a series of features distributed across a given plane or (ii) a matrix of values where each cell corresponds to a point on the plane. Therefore, we can see the open-ended and complex nature of spatial problems, but it also leaves room for imagination to be applied in the search for new solutions.

One of the main complications we encounter when dealing with spatial problems is that they are very computationally intensive, typically taking a long time to produce the desired result. This drawback is also an opportunity to use heterogeneous systems to address spatial problems more efficiently. Heterogeneous systems give the developer greater freedom to speed up suitable algorithms by increasing the parallel programming options available, making it possible for different parts of a program to run on the dedicated hardware that suits them best.

Several of the spatial problems that have not been optimised for heterogeneous systems cover very diverse areas that seem vastly different at first sight. However, they are closely related due to common data processing requirements, making them suitable for using dedicated hardware.

In particular, this thesis provides new parallel approaches to tackle the following three crucial spatial problems: latent fingerprint identification, total viewshed computation, and path planning based on maximising visibility in large regions.

Latent fingerprint identification is one of the essential identification procedures in criminal investigations. Addressing this task is difficult as (i) it requires analysing large databases in a short time, and (ii) it is commonly addressed by combining different methods with complex data dependencies, making it challenging to exploit parallelism on heterogeneous CPU-GPU systems. Moreover, most efforts in this context focus on improving the accuracy of the approaches and

neglect reducing the processing time—the most accurate algorithm was designed to process the fingerprints using a single thread.

We developed a new methodology to address the latent fingerprint identification problem called *Asynchronous processing for Latent Fingerprint Identification* (ALFI) that speeds up processing while maintaining high accuracy. ALFI exploits all the resources of CPU-GPU systems using asynchronous processing and fine-coarse parallelism to analyse massive fingerprint databases. We assessed the performance of ALFI on Linux and Windows operating systems using the well-known NIST/FVC databases. Experimental results revealed that ALFI is on average 22x faster than the state-of-the-art identification algorithm, reaching a speed-up of 44.7x for the best-studied case.

In terrain analysis, Digital Elevation Models (DEMs) are relevant datasets used as input to those algorithms that typically sweep the terrain to analyse its main topological features such as visibility, elevation, and slope. The most challenging computation related to this topic is the total viewshed problem. It involves computing the viewshed—the visible area of the terrain—for each of the points in the DEM. The algorithms intended to solve this problem require many memory accesses to 2D arrays, which, despite being regular, lead to poor data locality in memory.

We proposed a methodology called *skewed Digital Elevation Model* (sDEM) that substantially improves the locality of memory accesses and exploits the inherent parallelism of rotational sweep-based algorithms. Particularly, sDEM applies a data relocation technique before accessing the memory and computing the viewshed, thus significantly reducing the execution time. Different implementations are provided for single-core, multi-core, single-GPU, and multi-GPU platforms. We carried out two experiments to compare sDEM with (i) the most commonly used geographic information systems (GIS) software and (ii) the state-of-the-art algorithm for solving the total viewshed problem. In the first experiment, sDEM results on average 8.8x faster than current GIS software, despite considering only a few points because of the limitations of the GIS software. In the second experiment, sDEM is 827.3x faster than the state-of-the-art algorithm considering the best case.

The use of Unmanned Aerial Vehicles (UAVs) with multiple onboard sensors has grown enormously in tasks involving terrain coverage, such as environmental and civil monitoring, disaster management, and forest fire fighting. Many of these tasks require a quick and early response, which makes maximising the land covered from the flight path an essential goal, especially when the area to be monitored is irregular, large, and includes many blind spots. In this regard,

state-of-the-art total viewshed algorithms can help analyse large areas and find new paths providing all-round visibility.

We designed a new heuristic called *Visibility-based Path Planning* (VPP) to solve the path planning problem in large areas based on a thorough visibility analysis. VPP generates flyable paths that provide high visual coverage to monitor forest regions using the onboard camera of a single UAV. For this purpose, the hidden areas of the target territory are identified and taken into account when generating the path. Simulation results showed that VPP covers up to 98.7% of the Montes de Malaga Natural Park and 94.5% of the Sierra de las Nieves National Park, both located in the province of Malaga (Spain). In addition, a real flight test confirmed the high visibility achieved using VPP. Our methodology and analysis can be easily applied to enhance monitoring in other large outdoor areas.

In the end, this work has resulted in three new parallel approaches to efficiently solve three relevant spatial problems, whose common link is the use of heterogeneous systems. The scientific community can apply these approaches to other problems with similar requirements to accelerate data processing.

Contents

Agradecimientos	XI
Abstract	XV
Contents	XXVI
List of Figures	XXXI
List of Tables	XXXIV
List of Algorithms	XXXV
List of Abbreviations	XXXVII
1.- Introduction	1
1.1. Motivation	3
1.2. Purposes of this work	5
1.3. Main contributions	6
1.4. Doctoral thesis organisation	7
2.- Background	9
2.1. Parallel computing	9
2.2. General-purpose computing on GPUs	11



2.2.1. Main GPGPU APIs	11
2.2.2. CUDA parallel computing platform	13
2.2.2.1. Hardware and software architectures	14
2.2.2.2. Memory hierarchy	16
2.2.2.3. Concurrent model	16
2.3. Heterogeneous computing	17
2.3.1. Main devices involved	17
2.3.2. Heterogeneous CPU-GPU systems	18
3.- Latent Fingerprint Identification	21
3.1. Introduction	22
3.1.1. Biometric recognition	22
3.1.1.1. Definition	22
3.1.1.2. History	23
3.1.2. Fingerprints	29
3.1.2.1. Formation	29
3.1.2.2. Different patterns	30
3.1.2.3. Types of fingerprints	32
3.1.3. Fingerprint recognition problem	33
3.2. Related work	34
3.2.1. Data enhancement	34
3.2.2. Data preprocessing	35
3.2.3. Acceleration of fingerprint matching	35
3.2.4. Fingerprint representation: MCC as the standard	36
3.2.5. Latent fingerprint identification	38
3.2.5.1. DMC: the state-of-the-art approach	39
3.3. ALFI: Asynchronous processing for Latent Fingerprint Identification	42
3.3.1. Asynchronous data processing	42

3.3.2. Fine-grained parallelism in processing 44

3.3.3. Data configuration 45

3.3.4. Pseudo-codes 47

 3.3.4.1. Control function (host): scheduling 47

 3.3.4.2. Kernel-1 (device): preprocessing angular differences 49

 3.3.4.3. Kernel-2 (device): matching minutiae descriptors . 50

 3.3.4.4. Kernel-3 (device): minutia quality computation . 51

 3.3.4.5. Kernel-4 (device): finding clusters 53

 3.3.4.6. FES function (host): final evaluation stage 54

3.4. Experiments and results 55

 3.4.1. Experimental setup 56

 3.4.2. Databases 58

 3.4.3. Accuracy analysis 59

 3.4.3.1. Identification databases 59

 3.4.3.2. Verification databases 62

 3.4.4. Computational performance analysis 64

 3.4.4.1. Linux 64

 3.4.4.2. Windows 65

 3.4.5. Application timeline 67

 3.4.6. Analysis of the results 68

 3.4.6.1. Accuracy 68

 3.4.6.2. Computational performance 70

4.- Total Viewshed Computation 71

 4.1. Introduction 72

 4.1.1. Earth representation 72

 4.1.1.1. Map projections 73

 4.1.1.2. Coordinate reference systems 76



4.1.2.	Geospatial analysis	78
4.1.3.	Geographic Information Systems	79
4.1.3.1.	History	79
4.1.3.2.	GIS applications	81
4.1.4.	Digital Elevation Models	81
4.1.4.1.	Types	82
4.1.4.2.	DEM generation and applications	83
4.1.5.	Visibility	83
4.1.6.	Viewshed	87
4.1.6.1.	Singular and multiple viewshed	88
4.1.6.2.	Total viewshed	93
4.1.7.	Summary of the main concepts	95
4.2.	Related work	96
4.2.1.	Singular and multiple viewsheds	96
4.2.2.	Total viewshed	97
4.3.	sDEM: skewed Digital Elevation Model	98
4.3.1.	Data reuse	98
4.3.2.	Data relocation	98
4.3.3.	Proposed methodology	101
4.3.4.	Multi-GPU implementation	103
4.3.4.1.	Kernel-1: obtaining the <i>skwDEM</i> structure	103
4.3.4.2.	Kernel-2: viewshed computation on the <i>skwDEM</i>	105
4.3.4.3.	Kernel-3: obtaining the viewshed on the DEM . .	108
4.3.4.4.	Managing multiple GPUs on the host	109
4.4.	Experiments and results	110
4.4.1.	Experimental setup	111
4.4.2.	Comparison with GIS software	112
4.4.3.	Comparison with the state-of-the-art algorithm	114

4.4.3.1. Total viewshed considering a random sector 114

4.4.3.2. Total viewshed based on average values 116

4.4.3.3. Total viewshed map generation using sDEM 117

4.4.4. Multi-GPU implementation profiling 118

4.4.4.1. Application timeline 119

4.4.4.2. Performance-critical kernels 119

4.4.5. Analysis of the results 127

5.- Path Planning Based on Visibility Data 129

5.1. Introduction 130

5.1.1. Forest fire prevention using UAVs 130

5.1.2. Path planning 133

5.1.2.1. Classification of path planning methods 134

5.1.2.2. Global path planning 135

5.1.2.3. The travelling salesman problem 140

5.2. Related work 141

5.2.1. Obstacle avoidance 141

5.2.2. Energy constraints 142

5.2.3. Area coverage 142

5.3. VPP: Visibility-based Path Planning 145

5.3.1. Types of waypoints 146

5.3.2. Main data structures 147

5.3.3. Workflow of the VPP heuristic 148

5.3.4. Description of the four stages 149

5.3.4.1. Stage-1: evaluating the initial locations 149

5.3.4.2. Stage-2: locations of maximum viewshed 150

5.3.4.3. Stage-3: isolated areas 152

5.3.4.4. Stage-4: flight path generation 153



5.4. Viewshed-rejection method for the onboard camera	155
5.4.1. Photographs taken by the onboard camera	155
5.4.2. Additional data structures	156
5.4.3. Finding the best COI for each waypoint	156
5.4.4. Workflow for simulating the camera behaviour	158
5.5. Experiments and results	159
5.5.1. Experimental setup	160
5.5.1.1. Systems and algorithms	160
5.5.1.2. Key features of the UAV	160
5.5.1.3. Memory optimisation	160
5.5.1.4. Main characteristics of the target areas	161
5.5.2. Flight simulation 1: Montes de Malaga	161
5.5.3. Flight simulation 2: Sierra de las Nieves	163
5.5.4. Real flight test: surroundings of Montes de Malaga	164
5.5.5. Analysis of the results	165
6.- Conclusions and Future Work	167
6.1. Conclusions	167
6.2. Future work	172
Appendices	173
A.- Formal definitions	173
A.1. Definitions	173
A.2. Mathematical functions	173
B.- Resumen en español	177
B.1. Introducción	178
B.2. Motivación y Línea de Investigación	180

B.3. Computación en la Actualidad 183

 B.3.1. Principales dispositivos implicados 183

 B.3.2. Sistemas heterogéneos CPU-GPU 184

B.4. ALFI: Identificación de Huellas Dactilares Latentes 186

 B.4.1. DMC: agrupación deformable de minucias 187

 B.4.2. Metodología 188

 B.4.2.1. Procesamiento asíncrono de datos 188

 B.4.2.2. Paralelismo de grano fino en el procesamiento . . 190

 B.4.3. Experimentos y resultados 191

 B.4.3.1. Diseño de los experimentos 191

 B.4.3.2. Bases de datos 193

 B.4.3.3. Análisis de precisión en identificación 194

 B.4.3.4. Análisis de rendimiento computacional 195

B.5. sDEM: Cálculo de la Cuenca Visual Total 199

 B.5.1. Conceptos básicos sobre la visibilidad 200

 B.5.2. Cuenca visual total 201

 B.5.3. Metodología 203

 B.5.3.1. Reutilización de los datos 203

 B.5.3.2. Reorganización de la malla de puntos 204

 B.5.3.3. Cálculo de la visibilidad 205

 B.5.3.4. Implementación multi-GPU 207

 B.5.4. Experimentos y resultados 207

 B.5.4.1. Diseño de los experimentos 208

 B.5.4.2. Comparación con las aplicaciones GIS 209

 B.5.4.3. Comparación con el algoritmo de referencia 210

B.6. VPP: Planificación de Trayectorias con Máxima Visibilidad 214

 B.6.1. Planificación de trayectorias 215

 B.6.2. Metodología 217



B.6.2.1. Caso práctico: Parque Natural Montes de Málaga	217
B.6.2.2. Heurística propuesta para generar la trayectoria	217
B.6.2.3. Método de rechazo de la visibilidad	223
B.6.3. Experimentos y resultados	226
B.6.3.1. Diseño de los experimentos	226
B.6.3.2. Simulación del vuelo del UAV	227
B.7. Conclusiones y Trabajo Futuro	228

Bibliography

235

List of Figures

2.1. CUDA hardware model	14
2.2. CUDA software model	15
2.3. Heterogeneous CPU-GPU system with dedicated GPU	18
3.1. First contract signed using the palm of the hand [41]	24
3.2. Stages of fingerprint formation [49]	29
3.3. Most widely used fingerprint representations	30
3.4. Main five classes of fingerprint patterns [51]	31
3.5. Types of fingerprint images [53]	32
3.6. Cylinder associated with a random minutia [70]	36
3.7. Example of a cylinder section of a random minutia [70]	37
3.8. Bit-based implementation of the cylinder associated with a random minutia [70]	38
3.9. Flowchart of the state-of-the-art DMC-CC algorithm [33]	40
3.10. The operations carried out by ALFI on the host and device	43
3.11. Proposed method for processing fingerprint impressions performed by ALFI on the device	45
3.12. Difference between the local matching and clustering stages	53
3.13. CMC curves of ALFI and DMC	61
3.14. Throughput results of ALFI and DMC-CC on Linux	65
3.15. Throughput results of ALFI and DMC-CC on Windows	66

3.16. Timeline of ALFI	67
3.17. Enlarged images of the ALFI timeline	68
4.1. Map projection families [96]	74
4.2. Geographic coordinate system representation using WGS84 [97] . .	76
4.3. Universal Transverse Mercator CRS [98]	78
4.4. First geospatial analyses from the 19th century [99]	79
4.5. Screenshot of the ArcView extension from ArcGIS	80
4.6. Screenshots of EarthViewer 3D and its successor Google Earth . .	81
4.7. Main types of Digital Elevation Models (DEMs) [104]	82
4.8. First example of the visibility problem between two locations: the POV at ground level and the target point	85
4.9. Second example of the visibility problem between two locations: the POV at height h and the target point	86
4.10. Example of the azimuthal discretization of a given DEM	87
4.11. Singular viewshed problem	88
4.12. Side and zenithal views for a single POV	91
4.13. Multiple viewshed problem	92
4.14. Total viewshed problem	93
4.15. Diagram showing the different viewshed problems related to sin- gular, multiple and total viewsheds	96
4.16. Three examples of Band of Sight (BoS) structures on the plane showing the proposed data relocation for a single sector	99
4.17. Three possible results from the data relocation process considering a single sector	100
4.18. Flowchart of sDEM for total viewshed computation	102
4.19. The <i>DEM</i> and <i>skwDEM</i> structures showing three reference points and their projections for a single sector	105
4.20. The <i>skwDEM</i> and <i>skwVS</i> structures showing the projection of three reference points for a single sector	107

4.21. The *skwVS* and *VS* structures showing three reference points and their projections for a single sector 108

4.22. Computational performance comparison between sDEM and commonly used GIS software to compute multiple viewshed 113

4.23. Speed-up curves and throughput values comparing the different implementations of sDEM with the state-of-the-art algorithm in computing the total viewshed considering a random sector 115

4.24. Speed-up curves and throughput values comparing the different implementations of sDEM with the state-of-the-art algorithm in computing the total viewshed considering average values 116

4.25. Total viewshed map of the Montes de Malaga Natural Park generated using sDEM 118

4.26. Timeline of sDEM 119

4.27. Kernel-1 performance limiter analysis 120

4.28. Kernel-1 compute analysis 121

4.29. Kernel-1 occupancy analysis 121

4.30. Kernel-1 SM utilisation analysis 122

4.31. Kernel-1 profile 122

4.32. Kernel-2 performance limiter analysis 123

4.33. Kernel-2 compute analysis 123

4.34. Kernel-2 occupancy analysis 124

4.35. Kernel-2 SM utilisation analysis 124

4.36. Kernel-2 profile 125

4.37. Kernel-3 performance limiter analysis 125

4.38. Kernel-3 compute analysis 126

4.39. Kernel-3 occupancy analysis 126

4.40. Kernel-3 SM utilisation analysis 126

4.41. Kernel-3 profile 127

5.1. Photograph taken of a drone flying during a fire [134] 132



5.2. Two different path planning strategies depending on the means of transport [142, 143] 134

5.3. Map representations using cell decomposition methods [145] 136

5.4. Map representations using roadmap methods [145] 137

5.5. Map representation using artificial potential field [145] 138

5.6. One example of each of the most well-known types of path planning algorithm [145, 148] 139

5.7. Illustration of the travelling salesman problem (TSP) 140

5.8. Area of the Montes de Malaga Natural Park (Malaga, Spain) . . . 146

5.9. Flowchart of the VPP heuristic 148

5.10. CHAR map at Stage-1 of VPP 150

5.11. CHAR map at Stage-2 of VPP 151

5.12. Isolated areas identified using contour lines at Stage-3 of VPP . . . 153

5.13. Resulting flight path generated using the VPP heuristic in the area of the Montes de Malaga Natural Park 154

5.14. Viewshed computation from a reference point of view (POV) . . . 157

5.15. UAV flight simulation considering the first four waypoints of the path 159

5.16. Flight simulation 1: Montes de Malaga Natural Park 162

5.17. Flight simulation 2: Sierra de las Nieves National Park 163

5.18. Real flight test in the surroundings of the Montes de Malaga Natural Park 164

B.1. Sistema heterogéneo CPU-GPU con GPU dedicada 185

B.2. Flujo de operaciones ejecutadas por ALFI en el huésped y en el dispositivo 189

B.3. Procesamiento de las impresiones dactilares realizado por ALFI en el dispositivo 191

B.4. Resultados de rendimiento de ALFI y DMC-CC en Linux 197

B.5. Resultados de rendimiento de ALFI y DMC-CC en Windows . . . 198

B.6. Principales tipos de problemas de cuencas visuales 201



B.7. Problema de la cuenca visual total	202
B.8. Tres ejemplos de estructuras de banda de visibilidad (BoS) en el plano que muestran la reubicación de datos propuesta	203
B.9. Proceso de reubicación de datos considerando un único sector . . .	205
B.10. Diagrama de flujo de sDEM para calcular la cuenca visual total . .	206
B.11. Comparación de rendimiento computacional entre sDEM y las aplicaciones GIS más conocidas	209
B.12. Curvas de aceleración y resultados de rendimiento de sDEM en comparación con el algoritmo de referencia en el cálculo de la cuenca visual total considerando un sector aleatorio	211
B.13. Curvas de aceleración y resultados de rendimiento de sDEM en comparación con el algoritmo de referencia en el cálculo de la cuenca visual total considerando valores medios	212
B.14. Mapa de la cuenca visual total del Parque Natural de los Montes de Málaga generado con sDEM	214
B.15. Estrategias de planificación de trayectorias en función del medio de transporte [142, 143]	216
B.16. Mapa CHAR al final de la primera etapa de VPP	219
B.17. Mapa CHAR al final de la segunda etapa de VPP	220
B.18. Zonas aisladas identificadas mediante curvas de nivel al final de la tercera etapa de VPP	221
B.19. Trayectoria de vuelo resultante generada mediante la heurística VPP en la zona del Parque Natural de los Montes de Málaga . . .	222
B.20. Perfil del terreno, vista cenital y la estructura <i>RS</i>	224
B.21. Secuencia simplificada de operaciones en la que el UAV visita cada uno de los cuatro primeros puntos de paso de la trayectoria	226
B.22. Cuenca visual acumulada desde la trayectoria (PL-CVS) de la zona del Parque Natural de los Montes de Málaga	228

List of Tables

3.1. Example of the Henry fingerprint classification system [39]	26
3.2. NCIC fingerprint classification system [47]	28
3.3. Parameter descriptions and values used by ALFI	46
3.4. Data structures used by ALFI during execution	47
3.5. Specifications of the host units used in the experiments	57
3.6. Specifications of the device units used in the experiments	57
3.7. Background databases used in the experiments	58
3.8. Rank-1 accuracy of ALFI and DMC	60
3.9. Rank-20 accuracy of ALFI and DMC	60
3.10. Accuracy results of ALFI and DMC on the FVC 2002 databases .	62
3.11. Accuracy results of ALFI and DMC on the FVC 2004 databases .	63
3.12. Accuracy results of ALFI and DMC on the FVC 2006 databases .	63
3.13. Run-time and speed-up results of ALFI and DMC-CC on Linux . .	65
3.14. Run-time and speed-up results of ALFI and DMC-CC on Windows	66
3.15. Accuracy differences in the identification experiment obtained from analysing the rank-1 accuracy	69
3.16. Accuracy differences in the identification experiment obtained from analysing the rank-20 accuracy	69
4.1. DEMs used as input data in the experiments	111
4.2. Analysis of the results of the computational performance compar- ison between sDEM and commonly used GIS software	114



4.3. Run-time and speed-up results of sDEM and TVS in total viewshed computation considering a random sector	115
4.4. Throughput results of sDEM and TVS in total viewshed computation considering a random sector	116
4.5. Run-time and speed-up results of sDEM and TVS in total viewshed computation considering average values	117
4.6. Throughput results of sDEM and TVS in total viewshed computation considering average values	117
5.1. UTM coordinates of the three regions used in the experiments . . .	161
5.2. Elevation statistics of the three regions used in the experiments. .	161
B.1. Especificaciones de las unidades huésped de los experimentos . . .	192
B.2. Especificaciones de los dispositivos utilizados en los experimentos .	192
B.3. Bases de datos ampliadas utilizadas en los experimentos	193
B.4. Precisión de rango-1 de DMC y ALFI	195
B.5. Precisión de rango-20 de DMC y ALFI	195
B.6. Resultados de ALFI y DMC-CC en Linux	196
B.7. Resultados de ALFI y DMC-CC en Windows	198
B.8. DEMs utilizados como datos de entrada en los experimentos	208
B.9. Análisis de los resultados de la comparación de rendimiento computacional entre sDEM y las aplicaciones GIS más conocidas	210
B.10. Tiempo de ejecución y resultados de aceleración de sDEM y TVS en el cálculo de la cuenca visual total para un sector aleatorio . . .	211
B.11. Resultados de rendimiento de sDEM y TVS en el cálculo de la cuenca visual total considerando un sector aleatorio	212
B.12. Tiempo de ejecución y resultados de aceleración de sDEM y TVS en el cálculo de la cuenca visual total considerando valores medios	213
B.13. Resultados de rendimiento de sDEM y TVS en el cálculo de la cuenca visual total considerando valores medios	213
B.14. Análisis estadístico básico del conjunto de datos utilizado en los experimentos	227

List of Algorithms

3.1. Host function that controls the device	48
3.2. Kernel-1 in charge of the LUT_D computation	49
3.3. Kernel-2 in charge of the local matching process	50
3.4. Kernel-3 in charge of computing the minutia quality	51
3.5. Kernel-4 in charge of finding clusters of matching minutiae pairs .	52
3.6. Host function in charge of the final evaluation stage (FES)	55
4.1. General approach for solving the singular viewshed problem	89
4.2. Function that computes the viewshed given a set of points	90
4.3. Function that computes the visibility of an input point T	91
4.4. General approach for solving the total viewshed problem	94
4.5. Kernel-1 in charge of generating the $skwDEM$ structure	104
4.6. Kernel-2 in charge of the $skwVS$ computation on the $skwDEM$. .	106
4.7. Function in charge of the viewshed computation in a row from Kernel-2	106
4.8. Kernel-3 in charge of transforming the $skwVS$ on the $skwDEM$ into the VS structure on the DEM	109
4.9. Host code in charge of managing multiple GPUs	110



List of Abbreviations

ACO	Ant Colony Optimization
AFIS	Automated Fingerprint Identification System
ALFI	Asynchronous processing for Latent Fingerprint Identification
AOV	Angle of View
APF	Artificial Potential Field
API	Application Programming Interface
ASIC	Application-Specific Integrated Circuits
BoS	Band of Sight
CC-SVS	Camera Captured Singular Viewshed
CHAR	Cumulative Hidden Areas
CMC	Cumulative Match Characteristic
COI	Center of Interest
CPU	Central Processing Unit
CRS	Coordinate Reference System
CUDA	Compute Unified Device Architecture
CVT	Centroidal Voronoi Tessellation
DEM	Digital Elevation Model
DMC	Deformable Minutiae Clustering

DMC-CC	DMC – Cylinder-Code
DMC-MT	DMC – M-Triplets
DMC-NMD	DMC – Neighboring Minutiae-based Descriptor
EER	Equal Error Rate
FBI	Federal Bureau of Investigation
FES	Final Evaluation Stage
FLOPS	Floating-point Operations Per Second
FMR	False Match Rate
FPGA	Field-Programmable Gate Array
FVC	Fingerprint Verification Competition
GA	Genetic Algorithm
GIS	Geographic Information System
GPGPU	General-Purpose Computing on GPUs
GPP	Global Path Planning
GPU	Graphics Processing Unit
KMPS	Kilo Matches Per Second
LOS	Lines of Sight
LPP	Local Path Planning
M-TVS	Masked Total Viewshed
MCC	Minutia Cylinder-Code
NCIC	National Crime Information Center
NIST	National Institute of Standards and Technology
NVCC	Nvidia CUDA Compiler
NVVP	Nvidia Visual Profiler
PL-CVS	Point Line Cumulative Viewshed
POV	Point of View

PS-CVS	Point Set Cumulative Viewshed
RRT	Rapidly-exploring Random Tree
RS	Ring-Sectors
sDEM	skewed Digital Elevation Model
SDK	Software Development Kit
SIMD	Single Instruction Multiple Data
SM	Streaming Multiprocessor
SoA	Structure of Arrays
SVS	Singular Viewshed
TIN	Triangulated Irregular Network
TPS	Thin Plate Spline
TSP	Travelling Salesman Problem
TVS	Total Viewshed (algorithm)
UAV	Unmanned Aerial Vehicle
UTM	Universal Transverse Mercator
VPP	Visibility-based Path Planning

1 Introduction

Nowadays, we have vast datasets available in almost every conceivable field thanks to the Internet. In particular, one of these that has experienced significant growth is related to the analysis of data in two-dimensional scenarios, commonly known as *spatial analysis* [1].

The general meaning of spatial analysis states: *a general ability to manipulate spatial data into different forms and extract additional meaning as a result* [2]. More precisely, we can further restrict this definition so that spatial analysis includes any analytical technique that aims to find (i) the spatial distribution of a variable, (ii) the relationship between the spatial distribution of different variables, or (iii) the associations between variables. These results are commonly achieved by analysing every variable's topological, geometric, or geographic properties under study.

The problems arising from this type of analysis are neither clearly defined nor have a straightforward solution. We can start with the idea that spatial problems usually require a specific data structure type where the values included are not isolated, but there is usually a relationship between adjacent values.

Spatial data provide location-related information in which two types of data structures stand out [3]:

- **Vector data:** it represents basic geometrical shapes that include points (single vertices, e.g., buildings), lines (two or more vertices forming an open shape, e.g., rivers), and polygons (three or more vertices forming a closed shape, e.g., city districts).

- **Raster data:** it consists of a matrix of cells arranged in rows and columns, where each cell can contain a discrete or continuous value, such as population density or elevation, respectively. This type of data is typically square and regularly spaced.

In general, vector data is ideal for representing discrete features, such as specific locations, territories, and boundaries. In contrast, raster data is best suited for continuous features, including elevation and precipitation, among others. The most representative example of raster data is a Digital Elevation Model (DEM), which is used mainly by geographic information systems. In a DEM, each cell of the matrix includes the elevation value of the corresponding point in the terrain. It is worth mentioning that a point, which might correspond to the summit of a mountain as an example, will have adjacent cells containing similar but slightly lower elevation values. Note that spatial data is not only restricted to the geographical domain but has a much more open approach related to spatial distribution.

Another characteristic of spatial problems is the repetitive nature of the operations used to analyse the data. In practice, the same operations are typically repeated on the dataset but modifying the target variable. For instance, if we consider the visibility computation problem from a given point, the direction of the line of sight varies between 0 and 360° . Therefore, the data used in spatial analysis comply with the locality principle, also known as *locality of reference*. It states that a processor tends to access the same memory locations over time, i.e., recently used data is more likely to be reused by the program in the near future. In practice, we can distinguish two types of locality:

- **Temporal locality:** recently accessed data have a high probability of being accessed soon.
- **Spatial locality:** data stored close together in memory have a high probability of being accessed shortly. An example of this is the sequential locality that occurs when data is arranged in memory sequentially and accessed the same way, e.g., the linear access to a one-dimensional array.

The principle of temporal locality may apply when analysing spatial data, although it does not have to. However, one can take advantage of spatial locality since data in two-dimensional environments are typically related and stored contiguously in memory—like in the above example of the summit of the mountain and the surrounding points. Accordingly, this fact has allowed us to develop new approaches to accelerate spatial data processing using modern systems.

1.1. Motivation

Many spatial problems seem entirely different and unrelated at first sight; however, some similarities, usually hidden, should be explored to apply to these problems some optimisation approaches that have been already proved valid in other areas.

Based on the broad definition of spatial analysis, two similar scenarios are the pixels of an image and the elevation values in a DEM. The colour of a pixel has a strong relationship with its surroundings, as does the elevation of a point with its neighbours in a given terrain. This circumstance brings new possibilities for finding other solutions to very complex problems, such as fingerprint identification, total viewshed computation and path planning based on maximum visibility. We will describe below some of the most relevant characteristics of these three algorithms and introduce the questions this thesis will answer.

Fingerprint identification is one of the most widely used biometric identification methods nowadays. Each fingerprint has a set of features called minutiae that relate to the points where the ridges of the skin end or split. In practice, each minutia is typically defined by three values within the two-dimensional image of the fingerprint: direction, type, and x and y coordinates. Accordingly, an individual is identified by comparing their fingerprint with each fingerprint included in a specific database—comparing the relative characteristics of the minutiae of both fingerprints.

The identification process is challenging due to the large number of complex relationships defined and checked in each minutiae comparison to determine the level of similarity. The problem is further complicated when identifying **latent fingerprints**. The fingerprints belonging to this group result from unintentionally leaving sweat and/or oil deposits from the fingertip on a particular surface. Thus, they have low quality and need much more processing in general.

The most advanced matching algorithms for either latent or non-latent fingerprints are computationally expensive, time-consuming, and particularly inefficient when working on large databases. Some fingerprint identification algorithms are currently running on hardware accelerators, but there are no similar approaches for latent fingerprint identification. Therefore, one central question arises:

- *Is it possible to develop a new latent fingerprint identification algorithm based on the most accurate identification method so far to run on heterogeneous systems?*

If so, other questions are:

- *Is the resulting latent fingerprint identification algorithm fast enough?*
- *Does it achieve high accuracy in identification?*

Afterwards, we use the experience gained in the two-dimensional problem of fingerprint identification as a basis for finding new solutions to the second spatial problem mentioned: **total viewshed**. This problem consists in obtaining the visible area from each point in the DEM. Both spatial problems have several similarities that we can take advantage of, such as exploiting parallelism based on the partitioning and proximity of the data according to its two-dimensional location.

In geographic information systems, it is common to use algorithms similar to ray-tracing codes when working with digital terrain models. Such algorithms are used to study any variable related to parameters whose magnitude decreases with the square of the distance (e.g., radio signals, sound waves, and human sight). However, these algorithms require performing array accesses that lead to poor exploitation of the spatial locality in memory—even if the memory accesses are regular.

The aim here is to analyse how a complete reorganisation of the data on a per-direction basis leads to a considerable performance improvement, especially in computationally intensive algorithms. We will address the following questions:

- *Is it possible to find a new data structure for storing elevation values that will improve the locality in memory in terrain analysis problems?*

If possible, some other questions that need to be answered are:

- *Will this new data structure substantially increase the performance of the viewshed computation algorithms?*
- *Is it particularly efficient for the total viewshed problem?*

The final part of this thesis aims to bring a new solution to a fundamental problem: **path planning for monitoring purposes**. This problem tries to find the shortest path that provides the maximum visual coverage for a given terrain, which requires complete knowledge of the viewshed and the management and processing of spatially distributed data. An autonomous Unmanned Aerial Vehicle (UAV) will be the device in charge of following the path to monitor the target area. The critical questions of this research are:

- *Can we find a new alternative to address the path planning problem based entirely on maximising the visibility from the path?*
- *Is the algorithm developed to solve the total viewshed problem valid in this case?*
- *Is the direction of the onboard camera during the flight a critical factor to consider?*

1.2. Purposes of this work

The main objective of this thesis is to find new approaches to address some of today's most prominent spatial problems. This main objective can be divided into more specific ones:

1. Bring new ideas and approaches to three different computationally intensive algorithms of spatial analysis using strategies that have been proved valid in other scientific areas. These approaches have not been previously tested for spatial analysis and involve computation on heterogeneous CPU-GPU systems and full GPU processing.
2. Review the state-of-the-art regarding latent fingerprint identification and analyse the parts of the most accurate fingerprint identification algorithm—among those based only on minutiae—currently in use. This algorithm will be optimised to take advantage of the execution on multi-core and many-core accelerators on heterogeneous systems using OpenMP and CUDA.
3. Find and analyse the most up-to-date techniques used in viewshed analysis on DEMs, with a particular focus on the total viewshed problem. We will apply a matrix restructuring technique to take advantage of the high computational power of hardware accelerators, such as GPUs and heterogeneous architectures, where the most efficient system will be used. Moreover, we will deal with the high degree of parallelism existing in the calculation of parallel lines of vision, as well as the homogeneity of the scheduled operations (improving the performance on GPUs) and the data organisation (optimising the use of the memory hierarchy).
4. Identify the latest research in the literature regarding path planning using drones for monitoring purposes. Here, we intend to develop a new methodology for path planning based on maximising visibility from the flight path to address the monitoring of difficult areas considering forest fire prevention

as a case study. This methodology will try to solve the path planning problem based on a careful analysis of the viewshed to obtain a near-optimal path for a single UAV with an onboard camera. The camera's direction will be calculated for each waypoint within the path, thus increasing the amount of terrain covered.

1.3. Main contributions

The contributions of this work are:

1. A new algorithm called Asynchronous processing for Latent Fingerprint Identification (ALFI) that provides faster and accurate latent fingerprint identification over massive databases on heterogeneous systems. ALFI applies fine-grained parallelism at fingerprint descriptor level and asynchronous processing as the basis for achieving an effective CPU-GPU processing pipeline. ALFI reduces idle times in processing and exploits the inherent parallelism of comparing latent fingerprints to fingerprint impressions. To the best of our knowledge, there are no related algorithms in the literature developed for latent fingerprint identification on heterogeneous systems.
2. A new data structure called skewed Digital Elevation Model (sDEM) for faster processing in terrain surface analysis that vastly improves data locality in memory. In particular, this approach applies a data restructuring before processing that fully exploits the intrinsic parallelism of the total viewshed computation, achieving maximum performance through efficient memory accesses. We provide different implementations for single-core, multi-core, single-GPU and multi-GPU platforms, which are compared with the most commonly used geographic information systems and the state-of-the-art algorithm for total viewshed computation.
3. A new heuristic called Visibility-based Path Planning (VPP) that finds a path for a single UAV with high terrain coverage for any given territory. VPP carries out a thorough viewshed analysis and provides the identification of hidden areas in the case study area. Moreover, VPP calculates the direction of the onboard camera for each waypoint within the flight path, aiming to maximise the visual coverage but avoiding visual overlapping between adjacent locations. This methodology achieves the monitoring of the whole region of interest to, e.g., prevent fires at relatively low cost.

1.4. Doctoral thesis organisation

The remainder of this document is organised as follows:

- Chapter 2 addresses the fundamentals of parallel processing that lay the groundwork for a better understanding of this research.
- Chapter 3 presents the ALFI algorithm for CPU-GPU heterogeneous systems, along with the critical aspects of the fingerprint identification problem from the perspective of analysing latent fingerprints.
- Chapter 4 introduces the sDEM methodology intended to address the total viewshed problem for heterogeneous systems, together with the essential concepts of viewshed analysis.
- Chapter 5 explains the VPP heuristic for path planning based on visibility analysis, as well as the key points of this topic.
- Chapter 6 gathers the main conclusions drawn from the achievements of this thesis, the published works that support it, and the future work.

In addition, the essential functions of the ALFI algorithm are described in Appendix A. Finally, the Spanish summary of this thesis is given in Appendix B.

2 Background

The approaches intended to find a spatial relationship by processing large spatial datasets are often associated with very high complexity. This difficulty lies in the wide range of possibilities available regarding tackling the problem. Nevertheless, there is a condition almost fulfilled in any case that we can exploit: values arranged in nearby memory locations have similar characteristics.

This chapter introduces some of the most used techniques to accelerate data processing in computationally demanding algorithms, as with spatial analysis problems. We will address parallel computing, general-purpose computing on graphics processing units, and heterogeneous computing.

Section 2.1 describes the key aspects of parallel computing. Section 2.2 explains the methodology followed in the area of general-purpose computing on GPUs, with a special focus on the NVIDIA CUDA API. Section 2.3 presents the methodology followed in heterogeneous computing, along with a brief introduction of the three objects of study addressed in this thesis.

2.1. Parallel computing

Traditionally, software developers used to write the source code of any program with sequential execution in mind, similar to the way a person thinks. At the time, this proved to be the most straightforward and efficient approach. All programs developed were intended to run on single-core CPU systems and were therefore limited by the speed of the processor. Unfortunately, significant improvements could not be achieved without upgrading the hardware.

The development and subsequent growth of the Internet have enabled business and government organisations to store large amounts of data online. This information requires thorough processing to draw conclusions based on analysing the relationship between variables. In practice, this process would have been unfeasible in terms of time if processed with sequential algorithms such as those used years ago. Nowadays, massive datasets are usually processed using high-performance computers, also known as supercomputers, mainly designed for research purposes.

Not every researcher has access to a supercomputer. Even if they do, optimised code must always accompany the powerful hardware to unleash the full potential of the system on which it runs. This last point is why parallel processing techniques emerged, aiming to speed up the analysis of massive datasets while making the best use of the underlying hardware.

In practice, any sequential algorithm analysed and designated as suitable can be parallelised to run on multi-core CPU systems, but with the disadvantage of having to write parallel programs. This type of algorithm follows the *divide-and-conquer* approach that divides the main task into a series of subprocesses that are executed simultaneously, thereby reducing the processing time. The developer must explicitly define the synchronisation of the many subprocesses and accesses to those code areas of shared data.

In general, parallel processing techniques fall into two different groups according to the nature of the application involved:

- **Compute-intensive:** this term describes compute-bound applications, i.e., most of the execution time is spent in computation on the Central Processing Unit (CPU), instead of input/output (I/O) operations. As they typically require small volumes of data, parallel processing involves parallelising the most computationally demanding parts of the algorithm. In more complex platforms, a common approach is to divide the process into separate tasks executed in parallel on different platforms, avoiding serial execution and optimising the available resources. It is known as task parallelism since multiple tasks are simultaneously performed, with each one tackling a particular part of the problem.
- **Data-intensive:** this concept identifies memory-bound applications in which a large amount of data must be processed and, therefore, the application spends most of its execution time moving and manipulating data instead of computing. Here, parallel processing involves dividing the data into multiple chunks that are then processed independently by the same

application running in parallel. At the end of the process, independent results are accumulated to produce the complete data output. This type of application usually scale linearly according to the data size and are made to be easily parallelised.

The above is the ideal classification of the algorithms according to their limitations. Nevertheless, an algorithm may have characteristics from both groups or may seem to belong to one group but actually belongs to the other once analysed. This fact occurs with well-known algorithms from the research areas involving latent fingerprint identification and viewshed computation. They have to handle massive databases (typically divided into several chunks) but are CPU-bound because of the high level of processing required to achieve the results. The optimisation task will be challenging in such cases if the parallel computation on multi-core CPUs is the only option considered. In this regard, the inclusion of GPUs has carried great weight in recent years for parallel computing as they are capable of massive multi-threading in data-intensive computation, among other advantages.

2.2. General-purpose computing on GPUs

In the last two decades, the role of GPUs has completely evolved from managing tasks related to visual processing (e.g., rendering 3D graphics in video games and visual applications) to general data processing, commonly known as *General-Purpose Computing on GPUs* (GPGPU). Such type of system typically includes a CPU that controls one or more GPUs, achieving great success in the industry and spreading all over the world.

2.2.1. Main GPGPU APIs

The foundations of this technological breakthrough began with the release of OpenGL [4] in 1992. It was an operating system independent application programming interface (API) capable of rendering 2D and 3D vector graphics using the GPU. Even though OpenGL did not support general-purpose computing, it became the standard for rendering in the industry and the first step towards this capability.

At the beginning of the 21st century, the first developers using GPUs for general-purpose computing needed to represent their mathematical problems using vertices and pixels to run them on GPUs. It was not until 2006 when NVIDIA

launched the parallel computing platform and API model called *Compute Unified Device Architecture* (CUDA) [5]. CUDA provided general-purpose computing capability for NVIDIA GPUs, enabling researchers and developers to take full advantage of the parallel nature of these devices with less effort and more efficiently than in the past. CUDA is available using C, C++ or Python as programming language.

Nowadays, NVIDIA GPUs are the first choice in the field of research. The benefits of CUDA are [6]:

- New versions of the CUDA toolkit are frequently released.
- Highly advanced debugging and profiling tools available.
- There are many optimised libraries for various areas available, such as linear algebra and image processing.

Conversely, the main drawbacks of CUDA are:

- Memory transfers between GPU and CPU must be managed and located manually. Similarly, the code to run on the GPU must be explicitly programmed.
- CUDA does not support execution on GPUs other than NVIDIA graphics. Therefore, developers can not test the performance on, for example, AMD, Asus, and Intel GPUs. These companies have been involved in manufacturing GPUs for many years but to a lesser extent.

There is another API for GPU processing called *OpenCL* [7], initially released in 2009, which shares some similarities with CUDA: they both provide a specific set of operations for GPU processing. In particular, OpenCL has great potential since it is an open standard for low-level heterogeneous parallel programming with stable support up to the present day by the non-profit Khronos Group. The code developed to run on the CPU requires C/ C++ programming language, whereas the code developed for the GPU requires special API calls. The main advantages of OpenCL include:

- Run-time execution model, which brings abstract memory and portability.
- High portability of the developed code. It can run on very different hardware: CPUs, dedicated GPUs from different vendors (e.g., NVIDIA and AMD), integrated GPUs (the main manufacturer is Intel), and other types of accelerators including Xeon Phi and FPGAs.

The negative points of OpenCL involve:

- Low-level API that makes the programming task more difficult, e.g., a higher number of lines of code for the same task and more time spent on programming.
- Performance is not guaranteed to be maintained when changing the underlying hardware on which the code runs from one manufacturer to another.

The Khronos Group also launched in 2014 a new API called *SYCL* [8]. It is a royalty-free cross-platform abstraction layer based on OpenCL that enables programs to run on various heterogeneous processors by writing the source code using the ISO C++ standard. On the positive side, it seems to offer greater ease of programming as it is based on the ISO C++ standard, but it still has some disadvantages related to OpenCL.

In this work, NVIDIA CUDA toolkit 10.0 with C++ has been considered based on the following reasons:

1. OpenCL generally requires considerably more lines of code than CUDA for the same application [9], which results in more effort for the developer. However, the human factor also has an impact on this aspect.
2. The code generated as a result of the compilation stage using CUDA targets the architecture of the underlying NVIDIA GPU [10]. In contrast, OpenCL cannot achieve such fine-tuning as it is a more general cross-platform tool.
3. Regarding Internet searches, the worldwide popularity of CUDA is significantly higher than that of OpenCL (100 vs 4 in points of interest). Similarly, scientific publications containing the words ‘CUDA GPU’ (221,000 results) far outnumber those with ‘OpenCL GPU’ (33,000). Both statements are supported by the data from Google Trends and Google Scholar tools as of 01 January 2022.
4. Furthermore, from the data shown in the previous point, it can be extrapolated that CUDA has greater support from the scientific community. This fact helps resolve those problems that may arise during the development of the algorithms.

2.2.2. CUDA parallel computing platform

The NVIDIA CUDA framework for GPUs provides a high-level abstraction for software developers to use NVIDIA GPUs for general-purpose computing

using C/C++ and Python programming languages. This API enables applications running on the CPU—also known as *host*—to perform data processing on the NVIDIA GPU—*device*—following a more efficient strategy.

The areas in which GPU processing is widely used are related to emerging scientific and technological areas where matrix processing is the basis of computing, such as molecular analysis [11], weather prediction [12], visibility analysis [13], and biometric recognition [14, 15, 16]. All of these have in common the need to manage and process a massive amount of data, a task that GPUs can speed up significantly.

2.2.2.1. Hardware and software architectures

The hardware side of the NVIDIA CUDA framework [5] (Figure 2.1) consists of a set of Streaming Multiprocessors (SMs), whose number depends on the GPU architecture. Each SM is composed of usually 32 cores, which can run many threads in parallel responsible for executing the functions designed specifically

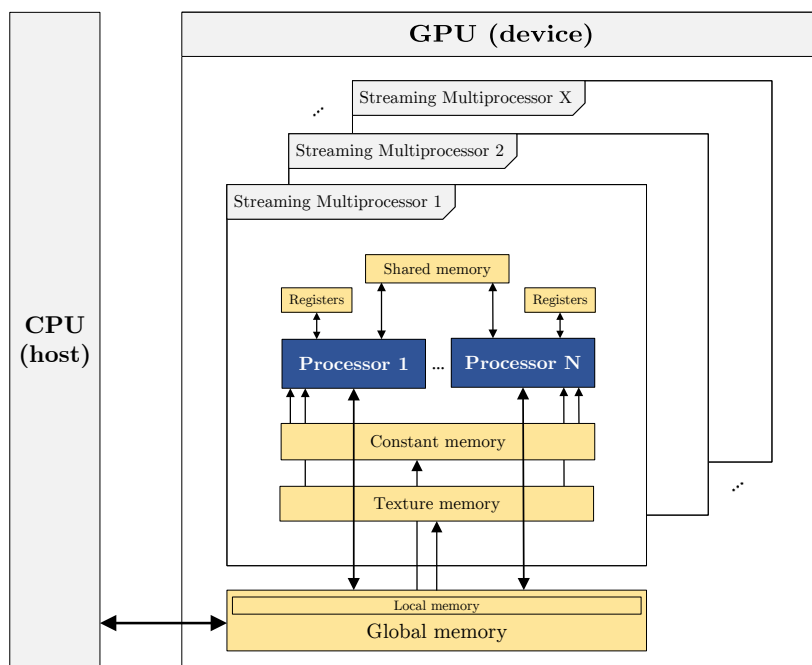


Figure 2.1: CUDA hardware model.

for the device, commonly known as *kernels*. We will explain the CUDA memory model in detail in the next section.

In the CUDA software model (Figure 2.2), threads are grouped into processing structures called warps, each of these containing typically 32 threads. Each thread from a particular warp should be performing Single Instruction Multiple Data (SIMD) operations inside the kernels to achieve maximum performance. The cause of this fact lies in avoiding the thread divergence problem, which occurs when threads from the same warp take different paths after processing a branch instruction, such as if-else and switch statements.

Moreover, threads are also grouped at a higher level into thread blocks, which run on the same SM sharing its resources. These thread blocks run independently since communication between blocks is impossible unless the global memory is involved. Finally, thread blocks are gathered inside a grid.

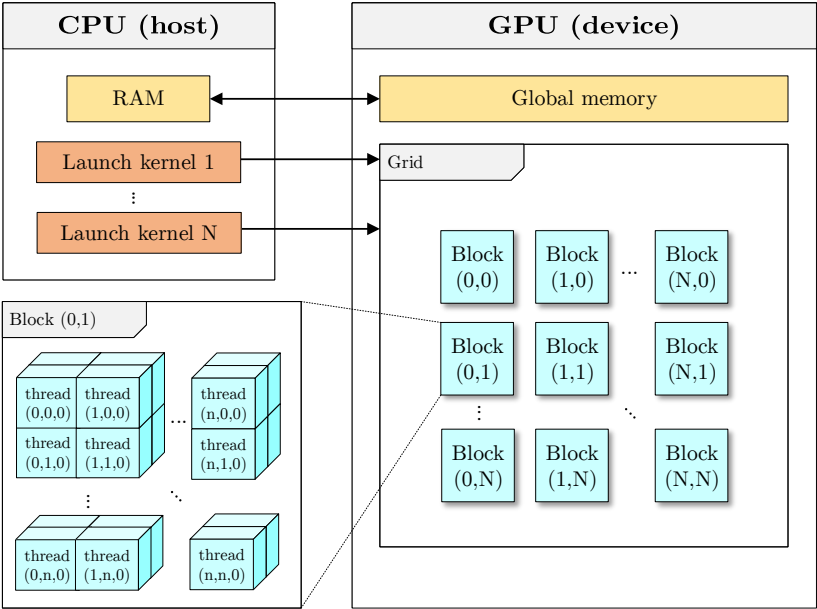


Figure 2.2: CUDA software model.

2.2.2.2. Memory hierarchy

The smallest and fastest memory units are the registers, followed by the local memory, which is much slower as it is included in the device global memory. Both types of memory are private for every thread, and the data stored cannot be shared between them. The next memory level is the shared memory space, where the data is accessible for every thread within the same block, provided that the block runs. Finally, the largest but slowest storage space is the global memory, which all thread blocks can access. Therefore, the global memory enables data sharing between threads, even between those that belong to different blocks. This last memory unit is used for communication purposes with the host unit. Additionally, there is a level 1 cache (L1) for each SM and a level 2 cache (L2) shared by all SMs.

Data allocations in the host memory are pageable by default, so that the device cannot access this data directly. Therefore, when a data transfer from pageable host memory to device memory is invoked, the CUDA driver comes into play. It must first allocate a temporary page-locked, generally known as pinned memory, then copy the host data to this pinned memory and, finally, transfer the data from the pinned memory to the device memory. In order to avoid this inefficient process, data in the host can be directly allocated in pinned memory, improving transfer speeds by preventing the memory from being swapped out.

2.2.2.3. Concurrent model

Concurrent execution is possible using the structures available in CUDA called *streams*, which are a sequence of operations executed in order on the device. Developers can create and utilise non-default streams, performing multiple operations such as concurrent execution of kernels and memory transfers inside different streams. Thus, the use of multiple streams can add new layers of parallelisation to particular applications, offering many more opportunities for optimisation. For example, data transfers between host and device can be overlapped with (i) computation on the host, (ii) computation on the device, and (iii) other data transfers between the host and device.

Note that synchronisation between different operations is necessary for successful concurrent execution on the device, and the structures called *events* can be helpful for this purpose. These events are synchronisation markers provided by CUDA that indicate whether a particular operation has concluded inside a specific stream. In practice, we can use an event to block the device or host until a given task finishes inside a particular stream.

2.3. Heterogeneous computing

Inherently-sequential algorithms (e.g., the n-body problem in physics) are impossible to parallelise efficiently and are therefore most suitable for being executed on single-core CPUs running at high frequencies. Conversely, embarrassingly parallel algorithms, such as Monte Carlo analysis, finite difference methods, and 2D/3D model rendering, take advantage of being executed on multi-core CPU systems, or even more on hardware accelerators with many cores running at lower frequencies. In general, most algorithms trying to solve real-world problems can be divided into subtasks, requiring serial or parallel execution and benefiting from concurrent execution on different devices.

2.3.1. Main devices involved

In the past, the processor was the leading actor in computing science with the first single-core CPU systems, where increasing clock speeds maximised performance. The clock speed of the systems progressively increased until the frequency could not keep up, and that is when multi-core CPUs came into play. These units achieve more performance with lower energy consumption, but with the drawbacks of spending much power in non-computational units (logic and cache, for example) and the need of having to write parallel code to take full advantage of the underlying hardware.

In parallel and following a similar approach, hardware accelerators started to gain importance in computing. These units are specialised pieces of hardware with greater efficiency than the general-purpose CPUs to carry out computationally intensive tasks. The cores are specifically designed to maximise performance by spending much power on performing calculations. These cores have fewer transistors running at lower frequencies, with the counterpart of sacrificing functionality: the impossibility of running operating systems and performing logic operations efficiently, among others. Some of the most commonly used accelerators are application-specific integrated circuits (ASICs), field-programmable gate arrays (FPGAs) and GPUs. Each of these devices offers different trade-offs between efficiency, configuration and development cost [17].

Things have changed since the first processors appeared, and the future of high-performance computing seems to be heading towards processing in heterogeneous computers where the task of processing is carried out by the CPU in combination with hardware accelerators to exploit all available resources.

A heterogeneous system contains different types of computational units, including (i) one or more general-purpose processors that run an operating system and execute the program, in addition to distributing the data among (ii) every available hardware accelerator that assists the processor to speed up the computation. In practice, each accelerator is best suited to a particular type of task that runs more efficiently on it than on the CPU. Thus, execution speed improves by introducing specialised hardware.

2.3.2. Heterogeneous CPU-GPU systems

The most common strategy in heterogeneous computing is combining general computing on the CPU—where greater control of the implementation is possible—with specialised fast processing on the GPU, thus reducing processing time when analysing large datasets. We can classify heterogeneous CPU-GPU systems into two groups, according to the layout of the units: both in the same chip or independent pieces of hardware (dedicated hardware). The first of these alternatives is widely used for small and everyday devices, such as smartphones, laptops and desktop computers for office. Processors that integrate CPU and GPU in the same chip are generally less powerful due to space and power constraints. On the other hand, dedicated hardware is typically installed in more powerful desktops and computing systems (Figure 2.3).

The use of heterogeneous systems has a considerable impact on the difficulty of the programming task, which becomes more challenging as the number of

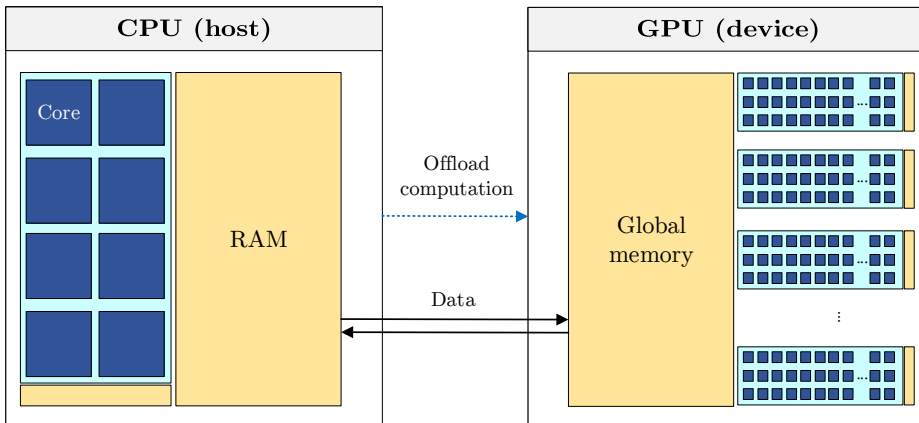


Figure 2.3: Heterogeneous CPU-GPU system with dedicated GPU.

devices considered increases. As mentioned in Section 2.1, parallel programming increases complexity due to the synchronisation required in the areas of code that access shared data. This problem, usually related to multi-core CPUs, also extends to heterogeneous systems to an even greater extent. However, the overall benefits of heterogeneous computing, which include faster processing speed, high capacity to handle large datasets, and lower cost, outweigh the disadvantages. Nowadays, many applications benefit from being redesigned to run on heterogeneous systems achieving significant reductions in time. Some tools such as OpenCL, SYCL, or CUDA help in the task of optimising existing algorithms or developing new ones.

Heterogeneous computing with GPU has experienced major growth in three areas that are of great importance to the scientific community: Artificial Intelligence [18, 19], Machine Learning [20], and Deep Learning [21, 22]. These three concepts are closely related, the first being the one that encompasses the other two concepts. Moreover, other applications, which belong to very different areas, involve time series analysis [23, 24], medical data processing [25, 26, 27], extreme learning machine [28, 29], numerical integration [30], and urban growth simulation [31]. Additionally, those problems involving spatial analysis also meet the requirements to be executed on heterogeneous systems as they are data-intensive applications that manage large datasets.

This work focuses on exploiting the intrinsic parallelism of three spatial algorithms belonging to different leading research areas: latent fingerprint identification, viewshed analysis, and path planning. These three algorithms seem completely different at first glance, but they have to analyse large datasets of spatially related samples. Latent fingerprint identification has to analyse a vast amount of data that can be easily divided into chunks, with the problem of avoiding the complex dependencies that make parallelisation impossible using independent tasks. The other two problems, related to viewshed computation and path planning, have to analyse less data, but it is much more challenging to split into independent chunks.

The following three sections will present different data processing solutions to tackle these spatial analysis problems, where the inclusion of GPUs in the processing stages is considered, either on its own or in combination with the CPU in heterogeneous systems.

3 Latent Fingerprint Identification

One of the essential procedures used in criminal investigations is to identify a suspect based on partial fingerprints left on the surface of an object at a crime scene. This process is called *latent fingerprint identification* and addressing it is challenging as (i) it requires analysing massive databases in reasonable periods and (ii) suspect identification is usually achieved by combining different methods with very complex data-dependencies [32]. These two facts make fully exploiting heterogeneous CPU-GPU systems very complex. Most efforts in this context focus on improving the accuracy of the approaches and neglect reducing the processing time. As a matter of fact, the most accurate approach was designed for one single thread [33].

This chapter introduces a new distributed and parallel computing methodology for latent fingerprint identification called *Asynchronous processing for Latent Fingerprint Identification* (ALFI). We designed it to fully exploit all the hardware resources of heterogeneous systems with CPU and GPU cores working at full capacity. This is achieved by combining an efficient asynchronous CPU-GPU communication approach with fine-grained parallelism at fingerprint descriptor level. In practice, ALFI can analyse large databases faster than the state-of-the-art algorithm [33] while providing very similar precision results. Therefore, local authorities could use ALFI to reduce processing times using the hardware available on almost any computer today.

The main contributions of this chapter are as follows:

- A new methodology called ALFI (Asynchronous processing for Latent Fingerprint Identification) is designed for a faster and accurate latent fingerprint identification on heterogeneous systems.
- Fine-grained parallelism at fingerprint descriptor level is proposed as a basis for achieving an effective CPU-GPU processing pipeline.
- There are no related algorithms of latent fingerprint identification in the literature specifically developed for heterogeneous systems.

Section 3.1 introduces the biometric recognition concept with a particular focus on the use of fingerprints. Section 3.2 describes the state-of-the-art regarding fingerprint identification. Section 3.3 presents the ALFI methodology for heterogeneous CPU-GPU systems. Section 3.4 evaluates the proposed algorithm in terms of accuracy and computational performance, comparing it with the state-of-the-art approaches. In addition, the essential functions used throughout this chapter are given in Appendix A.

3.1. Introduction

The field of biometric recognition is extensive, with many researchers involved. Here is an overview of the history highlighting the significant discoveries and all the necessary previous knowledge.

First of all, this section introduces the biometric recognition concept and the most relevant historical events with a particular focus on fingerprints. Secondly, we explain how fingerprints are formed and their main characteristics. Finally, the fingerprint recognition problem and its main difficulties are addressed.

3.1.1. Biometric recognition

3.1.1.1. Definition

With the development of new technologies such as the Internet, mobile phones, and computers, keeping all information in a secure environment has become a challenging task and a priority in anyone's life. Nowadays, data is commonly protected under the user identity so that only the corresponding individual can access the information.

Many different methods have been developed to protect the information of a personal nature, where the most popular are passwords and patterns, but equally, they can be the most insecure. Any person possessing the password or pattern could access a hypothetical system without significant problems, pretending to be someone else and stealing the data. This issue is one of the reasons why biometric recognition—the use of some parts of the human body to identify the user—has become very popular in recent years [34].

The term *biometric* is formed by combining the Greek words *bio* (life) and *metrics* (to measure). Therefore, biometric recognition refers to the process of identifying an individual based on some of the main physiological and behavioural characteristics. These include fingerprints, the shape of the face, colour and size of the iris, the retina, the shape of the hand, the shape of the ear, and the style and trend of walking.

Biometric recognition is not something new as it is intrinsic to human behaviour as a society. Since the beginning of civilisation, faces have been used unconsciously to distinguish relatives from strangers on a daily basis [35]. The difficulty of this basic task has grown enormously with the increase in population we have experienced, making manual identification mechanisms unmanageable. Over the last decades, automated biometric systems have become available because of the significant advances in computer processing [36]. Fingerprint processing stands out among them as it is widely used in daily identification tasks due to its uniqueness and ease of use. This well-known problem called *fingerprint recognition* has been studied since the late 19th century.

3.1.1.2. History

There are many examples of biometric recognition throughout history. The following are some of the most relevant events in that area—mainly focused on fingerprints.

Starting from the prehistoric period, palmprints have been found in the walls of many caves throughout the world [37]. Similarly, fingerprints appeared on the walls of tombs in ancient Egypt [38]. In this particular civilisation, physical descriptions were also used to differentiate trusted traders of known reputation from those new in the local market. In ancient China, there is strong evidence that fingerprints were used as a signature in Babylonian business transactions, recorded on clay tablets dating back to 300 B.C. [39]. The Chinese were the first culture to use friction ridge impressions for identification. Another example of biometric recognition is given in the Persian book *Jaamehol-Tawarikh* from the

14th century, which contains some comments about the practice of identifying individuals based on their fingerprints [40].

In the 19th century, the first appearance of fingerprints on official documents was in the records of the Civil Service of India in 1858 [41]. Sir William James Herschel (January 9, 1833; Slough, England) recorded captures of hand images, also known as palmprints, on the back of each contract with local workers, aiming to identify them in the future. Initially, palmprints were required, but the prints of the right index and middle finger were only used over time. This event is considered the first system that involved the capture of hand and finger images uniformly taken for identification purposes.

The palms of the hands were the commercial signature of the local businessman Rajyadhar Konai, which he used to sign Herschel's contracts (Figure 3.1) when working together on a road construction project. They both had a common interest in using this form of signature as Herschel came to claim: "To frighten Konai out of all thought of repudiating his signature hereafter." The conviction lied in the fact that through increasing personal contact with the document, this one was seen as a more binding contract than if the locals had simply signed it.

Nevertheless, the relevant finding came as Herschel's collection grew. He realised that those fingerprint impressions could also be used for identification purposes as they were discriminatory. After this and throughout his life, Her-

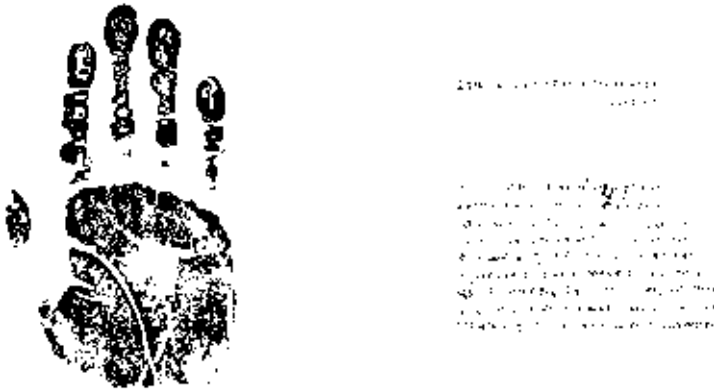


Figure 3.1: Famous Herschel's contract signed by the local businessman Rajyadhar Konai using the palm of the hand [41].

schel claimed that fingerprints were unique and permanent and, to support this statement, he documented his fingerprints over his lifetime to prove permanence. All these findings were reflected in his book *The Origin of Finger-Printing* published in 1916 [41]. Herschel is considered one of the first Europeans to recognise fingerprints as a valuable identification source.

During the last years of the 19th century, criminal investigation, as we know it, emerged with the first system of criminal investigation, developed in 1883 by Alphonse Bertillon (April 22, 1853; Paris, France) [42]. This system, commonly known as Bertillonage, was based on anthropometry, which studies the measurement and proportions of specific human body parts. In practice, the information of an individual included in criminal records was organised as follows: (i) descriptive data such as weight, height, and eye colour; (ii) body marks such as tattoos, scars and deformities; and (iii) body measurements. The more police institutions adopted this system, the more apparent its problems became. The main problem was related to measurements taken by different officers: they could be different enough when identifying the same person or similar enough to identify two individuals as the same person.

During the same period, the celebrated English scientist Sir Francis Galton (February 16, 1822; Birmingham, England), cousin of Charles Darwin, stated that through using Bertillonage as a method of criminal identification, the rate of false identifications would rise to unacceptable values worldwide [39]. As a result of his dislike of anthropometry, Galton researched personal individualisation using fingerprints, which culminated in the creation of the first fingerprint classification system [42], included in his famous book *Finger Prints* published in 1892 [43]. In this book, Galton claimed that fingerprints were unique and permanent and described a new classification system that used the alphabet to enumerate the three main fingerprint patterns: L for a loop, W for a whorl, and A for an arch. An example of this classification method using every finger in both hands of an individual is as follows: AWLWLLWLWL. Therefore, Galton is considered one of the pioneers in fingerprint identification and the first to provide a functional classification of fingerprint patterns.

On the other hand, Sir Edward Richard Henry (July 26, 1850; Shadwell, England) developed in the early 1890s a more comprehensive classification method with the help of Galton, in which fingerprints are classified based on their physiological characteristics [39]. By 1900, this classification method became very popular throughout the world, and such was its success that Scotland Yard decided to adopt it and abandon the Bertillonage system.

The Henry Classification System, as it came to be known, also served as a basis for the classification system used for many years by the Federal Bureau of Investigation (FBI) in the US. An example of this classification method is shown in Table 3.1 and Equation 3.1. It should be noted that this classification can only be applied in the case of having available all the fingerprints from both hands. It works as follows: a primary number (from 1 to 10) is assigned to each finger according to its position considering both hands, beginning from the right thumb and ending with the left little finger. Afterwards, a secondary number (power of two up to 16) is assigned only to those fingers that contain a whorl pattern (circular or spiral). In contrast, a zero value is assigned to those fingers with a non-whorl pattern. Not every finger with a whorl pattern has the same value as it is a function of the position. Finally, the result is a fraction where the sum of each even-numbered finger value is in the numerator, and the sum of each odd-numbered finger value is in the denominator, considering that a value of 1 is added to each sum with the maximum value of 32 on either side of the fraction.

Table 3.1: Example of the Henry fingerprint classification system [39]. Fingerprint patterns are denoted as *L* for a loop, *W* for a whorl, and *A* for an arch.

	Right hand					Left hand				
	Thumb	Index	Middle	Ring	Pinky	Thumb	Index	Middle	Ring	Pinky
Primary No.	1	2	3	4	5	6	7	8	9	10
Secondary No.	16	16	8	8	4	4	2	2	1	1
Pattern	L	L	W	A	L	W	L	W	L	L
Class (<i>C</i>)	-	-	8	-	-	4	-	2	-	-

$$C = \frac{1 + (\text{sum of each even finger value})}{1 + (\text{sum of each odd finger value})} = \frac{1 + (4 + 2)}{1 + (8)} = \frac{7}{9} \quad (3.1)$$

In the early years of the 20th century, the use of fingerprints for identification purposes began to grow. In 1924, the FBI created the first Identification Division to establish a public database of fingerprints obtained from police agencies all over the US to be used in searches for criminal investigations [44]. The fingerprint records, which initially included 810,188 files, formed the central repository of the FBI in the following years. As fingerprint records continued to grow, the task of manual searching became increasingly difficult and time-consuming.

In 1934, the FBI intended to develop the automation of known fingerprints involving sorting machines and punch cards. Each punch card contained the information of the class of a known fingerprint so that they could be sorted by

the machine—which previously contained punched cards related to the different classifications. Then, the card-sorting machine could extract the resulting cards matching the target classification with the aim that examiners could check them manually. Even though this method was revolutionary for the time, it was determined unsuccessful by the FBI and abandoned soon after it began [45].

In the 1950s, everything changed with the invention of the computer, so law enforcement took up research in the field of automating the identification process [42]. After creating the original fingerprint collection of 810,188 samples, hundreds of thousands of new records were added every year. Continuing this progression, the FBI's criminal database came to include approximately the data from 15 million individuals, which amounts to a total of 150 million fingerprints by the early 1960s. Even though the systems based on punch cards reduced the number of fingerprints to examine, human work was still necessary to analyse each fingerprint from the candidate list, reaching the point of being unable to cope with the daily work. In 1963, the FBI resumed the automation task of the criminal fingerprint database after initial research had shown it to be feasible.

In 1965, in parallel with the development of the automatic system for fingerprint processing that started a few years earlier, the New York State Information and Identification System researched the classification of fingerprints using local structures such as minutiae [42]. These local structures correspond to singular points located at the endings and bifurcations of the fingerprint ridges [44]. Thus, the classification task involved the manual recording of fingerprint minutiae on different overlays so that this data could be used to develop a novel minutiae extraction software. Unfortunately, budgetary restrictions caused the cancellation of the program shortly after the state hired a company to develop the minutiae coding system. Nevertheless, the results of this research were used later.

By the late 1960s, the FBI hired the National Institute of Standards and Technology (NIST) to build a prototype for reading and matching fingerprints, considered the first attempt to develop an Automated Fingerprint Identification System (AFIS) [46]. This prototype was finally delivered to the Identification Division of the FBI in 1972. Engineers first studied Henry's system to classify fingerprints, but it proved very time-consuming. Therefore, Henry's system was replaced by the new classification code developed by the National Crime Information Center (NCIC). This NCIC alphanumeric classification system (Table 3.2), which takes the same name as the centre, is pattern-specific to each fingerprint without considering the combination of fingers, thus differing from Henry's system and making the computational load lighter in processing. A typical NCIC fingerprint classification (NCIC-FPC) might appear as follows: FPC/ 20 dI PO PM 08 PO 17 19 PM CI.

Table 3.2: *NCIC fingerprint classification system [47].*

Pattern type	Pattern subgroup	NCIC symbols
Arch	Plain	AA
	Tented	TT
Loop	Ulnar	0 ~ 49 (ridge count)
	Radial	51 ~ 99 (ridge count + 50)
Plain whorl	Inner	PI
	Meet	PM
	Outer	PO
Central pocket loop whorl	Inner	CI
	Meet	CM
	Outer	CO
Double loop whorl	Inner	dI
	Meet	dM
	Outer	dO
Accidental whorl	Inner	XI
	Meet	XM
	Outer	XO
Completely scarred or mutilated pattern		SR
Missing or amputated		XX

Eventually, after thoroughly studying the manual methods used by human examiners, NIST engineers discovered that minutiae were crucial for achieving a proper identification method. Based on that, the last phase of the prototype development program focused on designing a new approach to automatically measuring, extracting, and matching minutiae from two different fingerprints to look for similarities. It was based on the conclusion that if two fingerprints had similar minutiae, they were determined to be identical.

The final identification system was called Finder, which was delivered to the FBI in the mid-1970s giving rise to many other similar systems. AFIS has spread worldwide since the 1980s until today, not only being used in law enforcement agencies but also in most of the current devices that we use daily, such as computers and smartphones.

3.1.2. Fingerprints

3.1.2.1. Formation

A fingerprint, as defined by the dictionary, is *the pattern of curved lines on the end of a finger or thumb that is different in every person, or a mark left by this pattern*. Fingerprints are entirely formed around the seventh month of fetal development, and the distribution of ridges does not change over time unless the individual suffers accidents such as cuts or burns on the fingertips (Figure 3.2) [48]. Both uniqueness and permanence make fingerprints widely used in current biometric recognition systems.

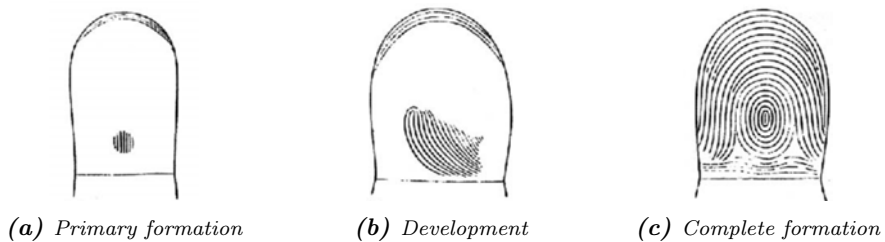


Figure 3.2: Stages of fingerprint formation [49].

Fingerprints are part of an individual's phenotype resulting from the interaction between a specific genotype (determined by the genes) and the environment. In particular, the relief of the fingertip appears when the skin begins to differentiate, which takes place with the growth of the volar pads of the fetal limbs. The cell differentiation process is greatly affected by changes in the fetus's position within the uterus and the flow of the amniotic fluid around it, even though these changes are minor. Hence, slightly different microenvironments are created from one individual to another where the cells on the fingertip grow.

The innumerable variables that affect cellular differentiation make it virtually impossible for two fingerprints from two different individuals to be the same. This is particularly the case with twins, where most of the physical characteristics are the same, but not the minutiae that form their fingerprints [50]. In addition, one might think from all these facts that fingerprints from twins are only a product of chance. However, it is not the case as the cell differentiation process is based on the same genes and, therefore, do not follow random but similar patterns.

3.1.2.2. Different patterns

When different scales are used to analyse a target fingerprint (Figure 3.3a), the main fingerprint patterns can be classified into two different levels [48]:

- **Global level:** the distribution of the ridges and valleys forms different patterns, where the important elements are called *singular points* and include cores and deltas (Figure 3.3b). The core is the structure located at the centre of several curved ridges. For example, in whorl fingerprints, the core is found in the middle of the spiral; in loop patterns, the core is in the top area of the innermost loop. On the other hand, the delta is the point that lies within a triangular structure formed by several ridges, where two parallel lines diverge around the loop or whorl. One delta can be found in loop patterns, whereas whorls have two or more. Singular points are useful in fingerprint classification and indexing problems, but they are not good enough to achieve accurate identification. Fingerprint shape, ridges frequency, and orientation field are other known patterns.
- **Local level:** there are many different types of local ridge characteristics, called *minutiae*, non-uniformly distributed across the fingerprint (Figure 3.3c). There are approximately 150 different types, such as islands, dots, terminations, crossovers, and spurs. However, the extraction of these structures greatly depends on the acquisition conditions of the images and the type of fingerprint; therefore, most of them cannot be observed clearly

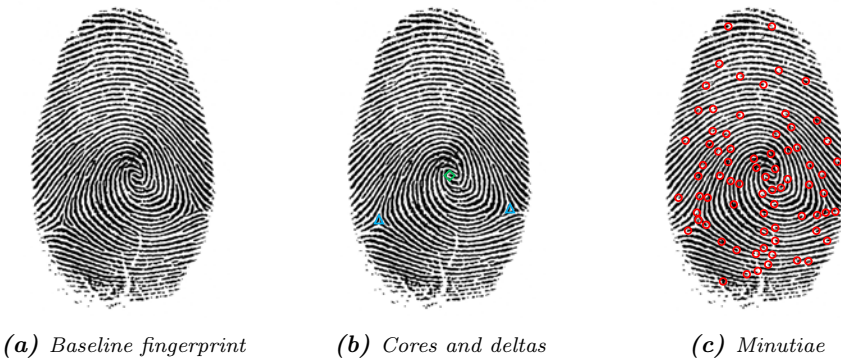


Figure 3.3: Baseline fingerprint and the two most widely used fingerprint representations. In (b), the cores and deltas structures belonging to the global level are marked with green squares and blue triangles, respectively. In (c), the minutiae structures belonging to the local level are marked with red rings.

enough. In practice, the most commonly used minutiae types are ridge endings and bifurcations. The ridge ending corresponds to the point of the ridge where it abruptly terminates. Conversely, the ridge bifurcation refers to the point where a single ridge diverges into two different branches. Minutiae-based representation and matching are known to be generally robust for use in identification; nevertheless, reliable automatic minutiae extraction can be problematic in extremely low-quality images, such as the ones obtained from latent fingerprints.

From a larger scale, Galton grouped fingerprints into three categories according to the different global patterns. Later, Henry included two more to form the five-class system still used by most authors in the literature (Figure 3.4) [52]. There are many more than these five classes, but the variations are usually generated from these. The five classes of fingerprints and their frequencies within the population are as follows:

- **Arch** (3.7%): the lines of the pattern form arches that flow smoothly from one side of the fingertip to the other, along with a slight upward twist near the centre of the pattern. This type is also known as plain arch. This class has no singular points.
- **Left loop** (33.8%): at least one ridge of the loop enters from the left side of the pattern towards its centre (core), then this ridge returns to the left side. There is usually more than one ridge forming the loop. This class has one core and one delta, where the core is located to the left of the delta.
- **Right loop** (31.7%): this is similar to the left loop class, but the ridges enter from the right side of the pattern. This type has one core and one delta, where the core is located to the right of the delta.

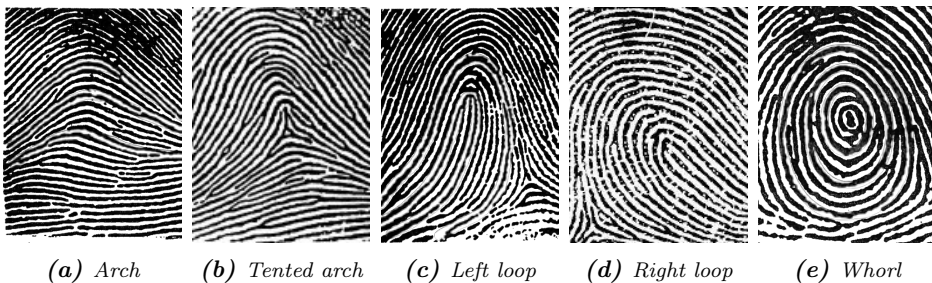


Figure 3.4: Main five classes of fingerprint patterns defined by Sir Edward Henry in the early 20th century and still used in most biometric classification researches [51].

- **Tented arch** (2.9%): this is similar to the plain arch class but with the difference of having raised ridges flowing in the same direction across the pattern. This class has one core and one delta.
- **Whorl** (27.9%): at least one ridge forms a circle, oval, or spiral (typically in the centre of the pattern), and there must be at least two deltas. Therefore, this class has one or two cores and two deltas.

3.1.2.3. Types of fingerprints

The type of fingerprint is another aspect to consider when developing a matching algorithm. As shown in Figure 3.5, fingerprints can be classified into three different types according to the conditions under which they are acquired [48]:

- **Rolled**: fingerprints obtained by rolling the finger from one side to the other, hence getting more information, but also introducing deformations in the resulting image. This type is characterised by good image quality due to a voluntary acquisition process performed under controlled conditions.
- **Plain**: fingerprints produced just by pressing the finger onto a surface. This type is also characterised by having a good image quality.
- **Latent**: fingerprints unintentionally left on a specific surface by deposits of sweat and/or oil from the fingertip. This type of fingerprint is usually not visible to the naked eye and requires additional processing to be detected. Most common acquisition techniques include dusting with fine

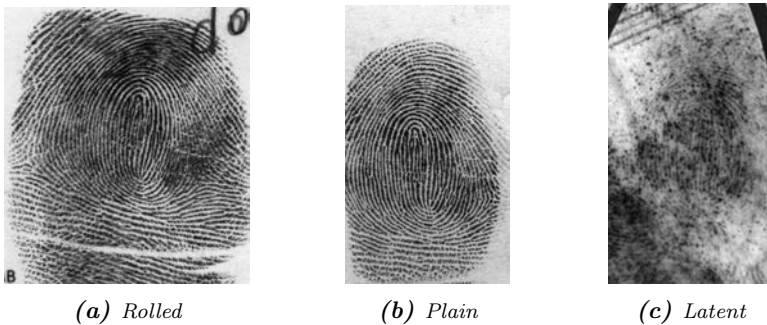


Figure 3.5: Types of fingerprint images according to different acquisition processes [53].

powder and using chemicals. Fingerprints obtained by any of these procedures may result in incomplete and inaccurate information per fingerprint, which introduces errors to the matching process [54]. However, their utility in criminal investigations and the inherent challenge of processing lower quality and deformed images are just a few of the compelling reasons to process them [55].

3.1.3. Fingerprint recognition problem

Fingerprint recognition is the automated process of identifying or verifying an individual's identity using fingerprints. This problem can be divided into two groups according to the number of comparisons needed to get a match [48]:

- **Verification:** it only requires one comparison to check if the target fingerprint matches with another stored previously; this is a 1:1 comparison, also known as fingerprint authentication. Typically, the user provides a username or card before using the fingerprint scanner. Then, the system analyses the input data and compares it with those associated with the user. If both fingerprints match, the individual is allowed to access the system.
- **Identification:** it is related to the problem of identifying an individual among those whose data is included in a specific database; this is a 1: N comparison, where N is the number of samples in the database. This number also coincides with the number of comparisons performed in a procedure commonly known as the matching process. Thus, in this type of problem, the system needs to search the entire fingerprint database until a match is found.

The second problem is the most challenging one in terms of computational cost and complexity [56]. In the literature, four approaches deal with large-scale databases containing plain/rolled fingerprints: classification, indexing, hardware improvement, and distributed and parallel computing.

Classification reduces the number of fingerprints to match by only comparing similar types. Indexing involves characterising and comparing each fingerprint according to a numerical vector that summarises its main features. Hardware improvement addresses the upgrade of existing hardware for faster processing. Finally, distributed and parallel computing aims to balance the computational load in processing between all available resources. In practice, processing latent

fingerprints requires handling partially or poorly data so that only the distribution technique is suitable.

Focusing on latent fingerprints, the difficulty in processing this type of fingerprint remains very high nowadays. The current trend seems to be in the direction of developing specific algorithms for latent fingerprint matching so that they are suited to their particular processing needs [54, 57]. Since there is very little information available per latent fingerprint, the focus is on finding and assessing relationships among the fingerprint descriptors. This fact creates data dependencies between different processing stages, and complex methodologies are required to manage them, making the use of parallel techniques difficult.

Another main disadvantage related to latent fingerprint identification algorithms lies in their inability to handle massive databases, in the order of millions of fingerprints, in the time required by law enforcement authorities. The latent fingerprint identification algorithm that provides the best trade-off between computational cost and precision is based on the Deformable Minutiae Clustering (DMC) method [33]; however, this algorithm was designed for one single thread.

3.2. Related work

Relevant research in fingerprint recognition falls into five categories related to (i) data enhancement, (ii) data preprocessing, (iii) acceleration of fingerprint matching, (iv) fingerprint representation, and (v) latent fingerprint identification.

3.2.1. Data enhancement

Most works in fingerprint data enhancement focus on designing new preprocessing techniques to improve the data acquired from a fingerprint or verify its authenticity. For instance, the orientation of the sample can produce bad accuracy results, so most relevant approaches find a correct orientation field model [58]. This parameter can be built even in the presence of noise and distortion [59] or using a trained convolutional neural network [60].

On the other hand, security and fault tolerance in current identification systems are essential issues to prevent attacks. This problem is usually addressed by analysing whether a particular fingerprint sample stems from a live subject or an artificial replica [61]. Although this problem remains difficult in terms of robustness, effectiveness, and efficiency, several studies are still proposing hardware and software-based approaches [62, 63].

3.2.2. Data preprocessing

Real-world fingerprint databases typically contain in the order of millions of fingerprints. Several studies reduce the computational cost by performing classification, indexing, hardware improvement, or parallel computing [48].

The most studied is classification, which filters large-scale databases by separating fingerprints into different categories based on their shapes. Only those belonging to the same class as the input sample will be processed in the following steps. It increases processing speed and allows massive databases to be handled [64, 52]. However, latent fingerprints usually correspond to partially or poorly acquired data, making these preprocessing tasks almost impossible.

3.2.3. Acceleration of fingerprint matching

The use of Graphics Processing Units (GPUs) in biometric recognition algorithms has increased in recent years. Several studies focus on this particular approach for databases with good quality fingerprints.

The authors in [14] proposed an optimised GPU fingerprint matching system based on MCC, which accelerates the comparison method up to 100.8x over the sequential CPU implementation. The proposal presented in [15] yields a speed-up of 1946x and 207x, considering the ratio between the kilo matches per second (KMPS) values and with respect to the non-optimised baseline and the one optimised with SIMD sequential CPU implementations, respectively. The work described in [16] accelerates a well-known fingerprint matching algorithm [65], achieving superior performance results in contrast to the multi-threaded CPU implementations [56]. The proposal in [66] speeds up the comparison method and implements a novel strategy in the consolidation stage that is shown to enhance accuracy.

All mentioned works that are specifically developed to be executed on GPUs share a common objective: speeding up the evaluation of massive databases by increasing the number of fingerprints processed per second (throughput). Nevertheless, these implementations need to be developed considering the underlying architecture and must be relatively simple to run effectively on GPUs [67], thereby reducing accuracy in most cases. Furthermore, state-of-the-art GPU-based algorithms do not exploit the power of the CPU in processing, which would lead to better run-time results.

3.2.4. Fingerprint representation: MCC as the standard

There exists a large number of studies on the representation of fingerprints. Early works analyse fingerprints considering core and delta parameters or ridge flow methods [68, 69], whereas current approaches consider minutiae [32]. These structures are the basis of the well-known Minutia Cylinder-Code (MCC) descriptor [70], widely used in the recent literature because of its high accuracy at relatively low computational cost [32, 71].

As introduced in Section 3.1.2.2, minutiae are local structures related to specific points in the discontinuities of the fingerprint ridges, such as endings and bifurcations. As a rule, each minutia is typically characterised by x and y coordinates, θ direction ($[0, 2\pi]$), type, and quality. One of the novelties introduced by Cappelli *et al.* [70] regarding the MCC descriptor is the use of coordinates and directions to represent minutiae in processing.

In MCC, each minutia m is associated with a 3D local structure called *cylinder* defined by radius R and height 2π ($[-\pi, \pi]$) and whose base is centred at the minutia location (Figure 3.6). Each cylinder is in charge of holding the spatial

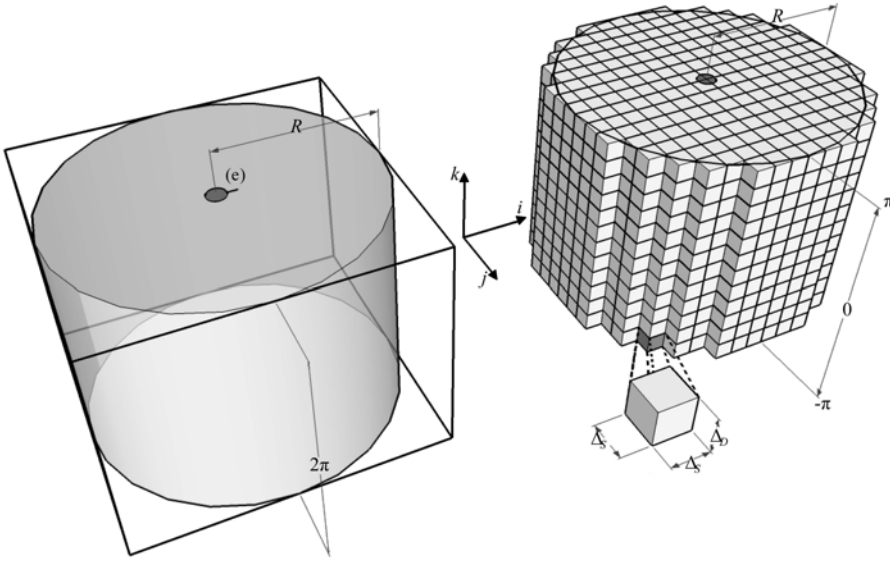


Figure 3.6: Cylinder associated with a random minutia. The cylinder and the cuboid enclosing it are drawn on the left, together with the resulting discretization of the cylinder on the right [70].

and directional relationship between the reference minutia and the ones within its neighbourhood, considering it a fixed-radius circumference. The cylinder is enclosed inside a cuboid whose base is aligned following the minutia direction θ ; likewise, the cuboid is discretised into cells. Each cell, whose position inside the cylinder can be uniquely identified by three indices (i, j, k) , contains a numerical value corresponding to the likelihood of finding minutiae near the cell with a directional difference, with respect to minutia m , below a previously established threshold. This value is obtained by adding the contributions of all the minutiae close to the cell. Each contribution depends on the Euclidean distance between the target minutia and minutia m following a Gaussian distribution (Figure 3.7). The cylinder structure has the following characteristics:

- Invariant for translation and rotation because it uses differences in distance and angle between minutiae instead of specific values.
- Robust against skin distortion, which is small at local level.
- Minor errors during the feature extraction process since the contribution of each minutia is based on the Gaussian function.
- Fixed-size structure given by the number of cells.

All these advantages, added to the use of statistical variables to obtain the value of each cell, make MCC suitable for a bit-based implementation where each

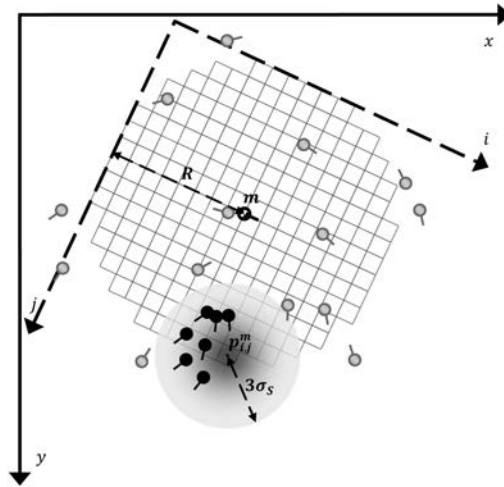


Figure 3.7: Example of a cylinder section associated with a random minutia m [70].

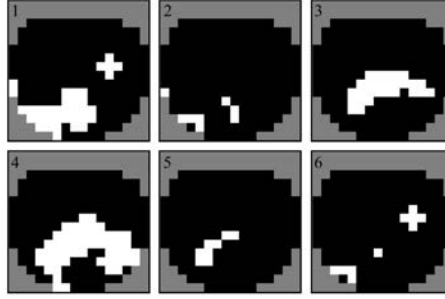


Figure 3.8: Bit-based implementation of the cylinder associated with a random minutia where six sections are shown, each of these corresponds to a specific angular value in the range $[-\pi, \pi]$. Black is related to the binary value 0, white is 1, and grey is invalid. The white cells represent the presence of minutiae, in that location, with an angular difference similar to the angular value of the cylinder section [70].

cell has one of the following three states: 1, 0, or *invalid* (Figure 3.8). The binary value 1 is given if the likelihood is above a preset threshold, 0 otherwise, and *invalid* if the target cell lies outside the cylinder region. With this approach, the process of matching two fingerprints is reduced to matching only their cylinders using MCC, in a much less computationally expensive process.

3.2.5. Latent fingerprint identification

Many studies analyse the performance of general identification algorithms in processing latent fingerprint databases. The achieved results revealed poor performance owing to the low quality of the input data [72], thus opening the way to the development of new algorithms designed to this aim.

Early works proposed several solutions for handling typical deformations that affect the matching procedure in latent fingerprint identification. For instance, regarding the minutiae matching process, several approaches are usually considered: the use of a minutia-based descriptor [73] or a combination of this structure and an orientation field descriptor of the fingerprint [53]. In the latter case, a global matching operation is performed by selecting the five best minutiae pairs to find new sets. For each found cluster, a matching score is computed, and after that the maximum value is chosen as the similarity score between the latent and rolled fingerprints. On the other hand, the proposal presented in [74] uses a different approach that combines local minutiae descriptors and fingerprint alignment through the Hough Transform to improve the fingerprint matching performance.

Another main characteristic of latent fingerprints lies in the presence of noise after the feature extraction. For this reason, researchers in [75] developed a method to improve the latent matching accuracy by incorporating feedback from some exemplars—rolled or plain fingerprints—to refine the feature extraction.

The most complete and accurate latent fingerprint identification algorithm, among those which are based solely on minutiae structures, finds deformable clusters of matching minutiae pairs in local regions by performing multiple alignments [33]. Overlapped clusters are merged to find consolidated matching minutiae pairs that are after that used to build a Thin Plate Spline (TPS) model [76]. Through this last step, new minutiae pairs, which might not have been found due to deformations in previous steps, can contribute to the global score.

3.2.5.1. DMC: the state-of-the-art approach

The Deformable Minutiae Clustering algorithm using the MCC descriptor (DMC-CC) [33] was developed by merging the following four well-known independent methods along with a final similarity computation stage:

- The MCC descriptor is used as input data of the local matching processing to find the first group of minutiae pairs. As described in Section 3.2.4, this descriptor is based on 3D data structures built from minutiae positions and angles after merging local structures [65].
- The Minutiae Discrimination method [77] calculates the quality value of each minutia in the latent fingerprint and fingerprint impression based on the neighbouring minutiae.
- The DMC method [33, 78] finds clusters of minutiae pairs, along with a weight value for each one, from the initial set of matching minutiae pairs. After merging the clusters, a final set of minutiae pairs is obtained and used to compute an initial similarity score between fingerprints.
- The Thin Plate Spline (TPS) method [76] is applied to avoid data loss due to fingerprint deformation and find new matching minutiae pairs. These pairs could have been discarded in previous steps due to the deformation effects and may improve the previously calculated similarity value.
- The last step is called Similarity Computation, where different statistical outcomes are obtained.

Given the above, the specific steps required by the DMC-CC algorithm (Figure 3.9) are described in detail below:

1. *MCC processing.* Let L and T be the minutiae sets of the latent fingerprint and a particular fingerprint impression from a database, respectively. Each minutia $q \in L$ is compared with all minutiae $p \in T$ based on their minutiae descriptors. Similar minutiae are selected as matched minutiae pairs (q, p) and included inside a new set A , which is after that, sorted in descending order according to their similarity values. Then, a new array M is filled with no more than $\max\{|L|, |T|\}$ local matching minutiae pairs from A so that the repetition of minutiae within different pairs is reduced.
2. *Minutiae Discrimination.* The quality value is computed for each minutia $q \in L$ and $p \in T$ relying on the neighbouring minutiae. This method is based on the fact that the less the minutia quality, the more minutiae there are around it [77]. Therefore, two sets containing all minutiae quality values from both fingerprints are obtained.

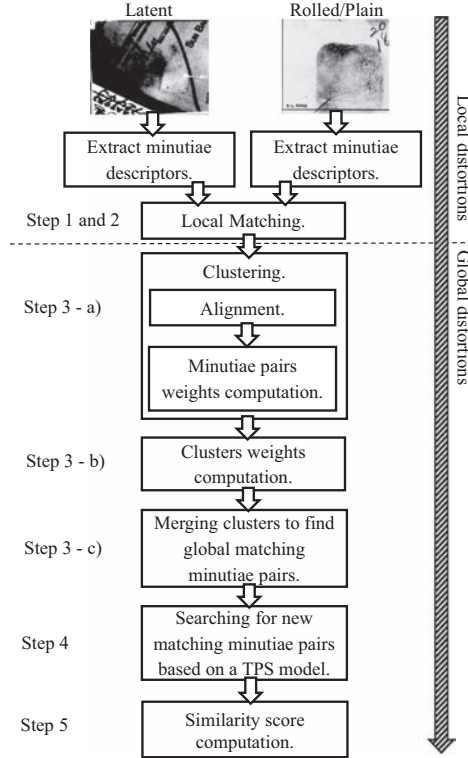


Figure 3.9: Flowchart of the state-of-the-art DMC-CC algorithm [33].

3. *Deformable Minutiae Clustering.*

- a) Each minutiae pair $(q, p) \in M$ is used to align fingerprints and find a cluster of matching minutiae pairs. Let \mathcal{C}_s be the set of found clusters of matching minutiae pairs. Each $(q_h, p_h) \in M, h = 1 \dots |M|$ is used in this step to work as the centroid of its cluster, denoted as B_h . For each $(q_g, p_g) \in M, g = 1 \dots |M|$ compute if q_g matches with p_g when aligning using current (q_h, p_h) and if this condition is fulfilled, update $B_h = B_h \cup (q_g, p_g)$.
 - b) Sort B_h in descending order according to their new similarity values obtained in the previous step. Let C_h be defined as the cluster that will contain a reduced number of minutiae pairs from sorted B_h to decrease the repetition of minutiae within different pairs. A weight $w_{p_h}^{q_h}$ for each minutiae pair is computed depending on the number of minutiae pairs inside its respective cluster C_h and the number of minutiae in the latent and fingerprint impressions. Every admissible cluster is then added to the actual set $\mathcal{C}_s = \mathcal{C}_s \cup \{C_h\}$ that will be used in the following steps.
 - c) The final weight w_{C_h} for each cluster $C_h \in \mathcal{C}_s, h = 1 \dots |\mathcal{C}_s|$ is obtained by accumulating every weight $w_{p_k}^{q_k}$ of the minutiae pairs $(q_k, p_k) \in C_h, k = 1 \dots |C_h|$. Then, \mathcal{C}_s is sorted in descending order based on their cluster weights and, thereafter, all clusters are merged according to several design parameters to find a preliminary set of global matching minutiae pairs (M').
4. *Thin Plate Spline.* From the previous set of minutiae pairs, a TPS model is built in order to correct any deformations the fingerprint image may have. By using this method, new minutiae pairs are found and included in a set called M^* . The weights of these minutiae pairs found with this method are calculated in a similar way as the one presented in Steps 3a-3c.
 5. *Similarity Computation.* The matching score between the latent fingerprint and the fingerprint impression is obtained by accumulating the weights of every minutiae pair inside both M' and M^* sets.

The DMC-CC algorithm presents several unavoidable and complex dependencies that force it to execute its steps sequentially. This fact causes a significant performance loss when computing on multi-core and heterogeneous systems.

3.3. ALFI: Asynchronous processing for Latent Fingerprint Identification

This section describes the new methodology specifically designed to address the latent fingerprint identification problem. The proposal is called *Asynchronous processing for Latent Fingerprint Identification* (ALFI) and exploits the intrinsic parallelism of the latent fingerprint identification procedure, which has not been addressed in recent literature. This methodology is developed considering the technical features of CPU (host) and GPU (device) to take the maximum advantage of these high-performance devices.

First, the fundamentals of ALFI with regards to the asynchronous processing method are described in this section. This is followed by an explanation of the fine-grained parallelism used. Then, the data structures used in ALFI are introduced. Lastly, different pseudo-codes are presented, which are related to (i) the host function in control of the device, (ii) the different kernels running on the device, and (iii) the host function in charge of the final evaluation stage.

3.3.1. Asynchronous data processing

ALFI is inspired by the DMC-CC algorithm (previously described in Section 3.2.5.1) but based on a complete redesign to achieve faster processing and correct performance on heterogeneous systems. We developed new methods for Steps 1-3a to be executed through different kernels because of their suitability to be processed on the device. Steps 3b-5 are modified to take the device results as input so that the host can execute them to balance the computational load between the host and device. These last steps computed on the host will be referred to as the multi-threaded *final evaluation stage* (FES) from now on. The host also coordinates the launch of all the operations to be executed on the device.

Let L and T_{DB} be the latent fingerprint used as a case study and the large-scale fingerprint database of impressions, respectively. First, since the data from the database cannot be entirely stored in the device memory at one stroke, this must be divided into several batches of equal size if possible. Each impression fingerprint from a particular batch $T \in T_{DB}$ is compared with the latent fingerprint L on the host after several steps are completed first on the device. These steps include host-to-device data transfer $H2D$, processing kernels K and device-to-host data transfer $D2H$ operations. ALFI efficiently overlaps and synchronises these operations and the ones performed on the host by using synchronisation events, forming an effective CPU-GPU processing pipeline avoiding idle times.

The behaviour of ALFI is shown in Figure 3.10 for the particular case of T_{DB} divided into four batches of fingerprints $T_i \in T_{DB}, i = 1 \dots 4$, for the sake of simplicity. First, the allocation of the latent fingerprint $A(|L|)$ and the entire database $A(|T_{DB}|)$ are performed in the host H . Allocating the database is possible in the host but not in the device since the memory space typically available in the first one is far larger than the available space in the second.

Pinned memory is used in the host memory since this method prevents the memory space from being swapped out, improving the speed of data transfers between host and device units. Regarding the memory management in the device D , the allocation of the latent fingerprint $A(|L|)$ is carried out at start-up. The rest of the available memory space is divided into two large spaces. Both areas, denoted as $A(n_m)$ each, will be filled in with two different batches of fingerprints to overlap data transfer and computation. Likewise, each area (including the batch within it) is managed by one of the two non-default CUDA streams str_0 and str_1 from the device.

An example of the flow of operations of ALFI could be the following: the two memory areas will be filled in with T_1 and T_2 batches at start-up and managed by str_0 and str_1 , respectively. In the following iterations, T_3 will be stored in

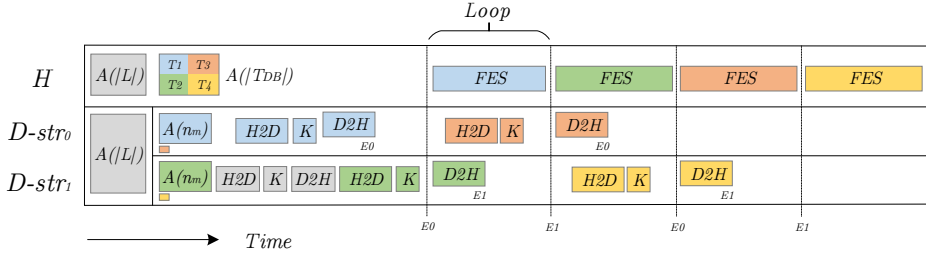


Figure 3.10: The operations carried out by ALFI on the host (H) and the device using two CUDA streams ($D-str_0$, $D-str_1$) over time for the particular case of one latent fingerprint L and four batches T_1-4 , resulting from splitting the database T_{DB} . These batches are allocated (A) in both device and host memory spaces where n_m is the number of fingerprints per stream in the device memory. Each operation, either on the host or device, is performed over a particular batch of fingerprints specified by the colour displayed. CUDA streams str_0 and str_1 and their corresponding synchronisation events E_0 and E_1 coordinate the requested operations. The operations performed on the device involve host-to-device ($H2D$) and device-to-host ($D2H$) data transfers and the computation carried out in kernels (K). These are overlapped with the multi-threaded final evaluation stage (FES) performed on the host so that while the host is processing a particular batch, the device is processing the next batch in the task queue.

the first memory space for the stream str_0 while the processing of T_2 is taking place in the stream str_1 and after the $D2H$ operation containing the results of processing T_1 is finished (E_0 event). Similarly, T_4 will be stored in the second memory space for str_1 while the processing of T_3 is taking place in str_0 and after the $D2H$ operation containing T_2 results is finished (E_1 event). This way, the data is always stored in the device and available before processing, reducing idle times. All these operations happen while the host is executing the FES function over its corresponding batch in the task queue.

3.3.2. Fine-grained parallelism in processing

Once the data is correctly allocated in the device memory, four different kernels K are launched to process batches of fingerprints on the device. Local minutiae matching is performed in $K_{1,2}$ with a fixed number of found matching minutiae pairs—where K_1 is only executed once at the beginning of the process. The quality score for each minutia inside the latent fingerprint and the batch of fingerprints is related to two executions of K_3 , with slight variations for the latent fingerprint. Finally, the alignment of the minutiae pairs to find clusters of these structures, which will add up to the overall total, is carried out in K_4 for a specific batch of fingerprints.

The computation performed in the previously mentioned kernels follows a similar strategy, except for K_1 since it implements a modified version of the algorithm described in [15]. Detailed descriptions of these kernels will be presented in Section 3.3.4.

Inside each kernel, ALFI states that each thread is in charge of processing a specific minutia from a batch of fingerprints, according to its thread identification number tid (Figure 3.11). For instance, considering a batch of fingerprints T containing m minutiae and a kernel K , the thread with $tid = s$ will pick and analyse the similarity of the minutia with index s with the ones in the latent fingerprint. This thread will carry out all the requested operations considering the fingerprint limits—starting s and ending e minutia indexes—to which the chosen minutia belongs.

After processing the four kernels on the device, a set of partial outcomes (vT) will be generated for each processed batch of fingerprints. This result is then transferred to the host and used as input to perform the multi-threaded FES step, obtaining the similarity scores between the latent fingerprint and the fingerprint impressions.

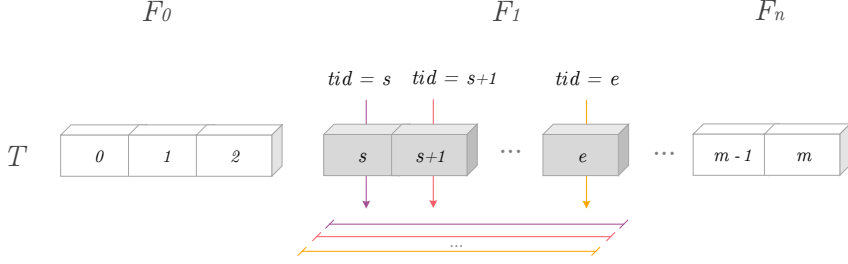


Figure 3.11: Proposed method for processing fingerprint impressions in a CUDA stream on the device. Each thread tid performs all the required operations in a kernel K over its corresponding minutia from a fingerprint F_i inside the T batch of fingerprints. Parameter descriptions are shown in Table 3.3.

3.3.3. Data configuration

ALFI needs several data structures to efficiently handle fingerprint processing. Every structure and parameter required by the ALFI methodology is given in Tables 3.3 and 3.4. Their descriptions follow:

- *ClusterCount* is a vector that contains the number of minutiae pairs inside the corresponding cluster from the *ClusterMtiaK* matrix.
- *ClusterMtiaK* is a matrix that contains the latent minutiae indexes found while performing alignments for each minutiae pair, which is formed by the one in T and its partner stored in *MaxMtiaL*, working as the centroid of the clusters.
- L is a structure of arrays (SoA) that includes the information related to the latent fingerprint. It is built in a similar way as the T_{DB} structure, but without the use of the index parameter since only one fingerprint is stored.
- LUT_D is a look-up table that includes all allowed latent minutiae indexes for any quantized angle $\hat{\gamma} = 0...z$ so that $LUT_D[\hat{\gamma}] = \{q_h \in L[d_\theta(\hat{\theta}_h, \hat{\gamma}) < \hat{\lambda}]\}$, where d_θ , in this particular case, represents the minimum angular difference between two quantized angles (Equation A.1 in Appendix).
- LUT_S is a look-up table that contains the first minutia index for each fingerprint in the database.

- *MatchingValue* is a vector that includes the similarity value between each minutia in T and its found partner stored in *MaxMtiaL*.
- *MaxMtiaL* is a vector used to store the most similar latent minutiae indexes from L for each minutia in T .
- *QualityL* and *QualityT* are vectors that include the quality value for each minutia in L and T , respectively.
- *Similarity* is a vector that includes the final matching score between latent and fingerprint impressions.

Table 3.3: Parameter descriptions and values used by ALFI. The values replaced by hyphen symbols indicate that they are either dependent on the database used in the experiments or design choices specified in the results section.

Parameter	Description	Value
α	The highest number of minutiae in a fingerprint from the database	-
λ	Max. angular difference between minutiae	$\pi/4$
ξ	Special value used to point the end of an array	-1
b	Bit-vector length of each minutia cylinder	1280
C_b	The number of blocks in the device unit	$32 \cdot C_s$
C_s	The number of SMs available in the device unit	-
C_t	The number of threads per block in the device unit	1024
H_θ	Threshold for angular minutiae similarity	$\pi/6$
H_e	Threshold for distance minutiae similarity	16
H_m	The number of minutiae inside the neighbourhood	3
$H_{q1,q2}$	Thresholds used for computing minutiae quality	18, 42
l	The number of minutiae in the latent fingerprint	-
m_d	The number of minutiae in the fingerprint database	-
m	The number of minutiae in the i -th batch of fps.	-
$N_{D,S}$	Sections and cells in every minutia cylinder	5, 16
n_d	Total number of fps. in the fingerprint database	-
n_m	The number of fps. per stream in the device memory	-
n	The number of fps. in the batch of fingerprints	-
tid	Thread identification number	-
z	Total number of quantized angles	256

Table 3.4: Data structures used by ALFI during execution on the host and device units. Parameter descriptions and their values are shown in Table 3.3.

Name	Layout	Memory transfer	Device access
<i>ClusterCount</i>	Array[m]	D2H	W
<i>ClusterMtiaK</i>	Matrix[$m \cdot \alpha$]	D2H	W
<i>L</i>	SoA[l]	H2D	R
<i>LUT_D</i>	Matrix[$z \cdot (l + 1)$]	-	R/W
<i>LUT_S</i>	Array[$n + 1$]	H2D	R
<i>MatchingValue</i>	Array[m]	D2H	R/W
<i>MaxMtiaL</i>	Array[m]	D2H	R/W
<i>QualityL</i>	Array[l]	D2H	W
<i>QualityT</i>	Array[m]	D2H	W
<i>Similarity</i>	Array[n_d]	-	-
<i>T_{DB}</i>	SoA[m_d]	-	-
<i>T</i>	SoA[m]	H2D	R

- *T_{DB}* is a SoA that contains the data of the fingerprint database in an optimal way for processing. In particular, every minutia data inside the fingerprint database is distributed across several vectors according to its different attributes, along with the k index of the fingerprint to which it belongs. In addition, *T_{DB}* will be split into several batches *T* for processing and the content of each one can be accessed on the device just by indexing with pointers.

3.3.4. Pseudo-codes

3.3.4.1. Control function (host): scheduling

At the beginning of the process, the tasks of the host involve launching, controlling, and coordinating all further operations to be executed on the device, as presented in Algorithm 3.1. In this code, the parameter r represents the ratio between the number of fingerprints in the database and the size of the fingerprint batches, indicating how many times the loop is performed. i and k are auxiliary variables used as indexes for the execution of operations and coordination events.

Algorithm 3.1 Host function that controls the device. A coordination event E is set if the call appears to the right of the operation. th_B is threads per block, and B is block; any other parameter is given in Table 3.3. The comments indicate the queued operations for the case of a single iteration of the main loop.

```

1:  $r = n_d/n$                                 ▷ fingerprints in database / fingerprints in batch
2:  $i = 2$  and  $k = 1$ 
3:  $A(|L|)$  and  $A(|T_{DB}|)$  in pinned host memory    ▷ reserve memory space in host
4: Split  $T_{DB}$  into  $T_h$ ,  $h = 1 \dots r$                 ▷ database divided into  $r$  batches
5:  $A(|L|)$  and  $A(2 \cdot n_m)$  in device memory        ▷ reserve memory space in device
6:  $str_1 \leftarrow \text{do } H2D(L)$                                 ▷ latent to device
7:  $str_1 \leftarrow \text{launch } K_1(z \text{ } th_B, 1 \text{ } B)$                 ▷ latent preprocessing
8:  $str_1 \leftarrow \text{launch } K_3(128 \text{ } th_B, 1 \text{ } B)$                 ▷ quality of latent
9:  $str_1 \leftarrow \text{do } D2H(QualityL)$                     ▷ quality of latent to host
10:  $str_0 \leftarrow \text{do } H2D(T_1)$                                 ▷ batch-1 to device
11:  $str_0 \leftarrow \text{launch } K_2(C_t/2 \text{ } th_B, C_b \text{ } Bs)$                 ▷ local matching
12:  $str_0 \leftarrow \text{launch } K_{3,4}(C_t \text{ } th_B, C_b \text{ } Bs)$                 ▷ quality and clusters
13:  $str_0 \leftarrow \text{do } D2H(vT_1) : E_0$                     ▷ partial results of batch-1 to host
14:  $str_1 \leftarrow \text{do } H2D(T_2)$                                 ▷ batch-2 to device
15:  $str_1 \leftarrow \text{launch } K_2(C_t/2 \text{ } th_B, C_b \text{ } Bs)$                 ▷ local matching
16:  $str_1 \leftarrow \text{launch } K_{3,4}(C_t \text{ } th_B, C_b \text{ } Bs)$                 ▷ quality and clusters
17: for  $iter = 1, r - 2$  do
18:    $str_k \leftarrow \text{do } D2H(vT_i) : E_k$                 ▷ partial results of batch-2 and then 3 to host
19:   Update  $i = i + 1$  and  $k = 1 - k$ 
20:    $str_k \leftarrow \text{do } H2D(T_i)$                                 ▷ batch-3 and then 4 to device
21:    $str_k \leftarrow \text{launch } K_{2,3,4}$                 ▷ kernels with same previous config. each
22:   Perform if  $E_k : FES(T_{i-2}, vT_{i-2})$                 ▷ batch-1 and then 2 processing on host
23: end for
24:  $str_k \leftarrow \text{do } D2H(vT_i) : E_k$                     ▷ partial results of batch-4 to host
25: Update  $k = 1 - k$ 
26: Perform if  $E_k : FES(T_{i-1}, vT_{i-1})$                 ▷ batch-3 processing on host
27: Update  $k = 1 - k$ 
28: Perform if  $E_k : FES(T_i, vT_i)$                     ▷ batch-4 processing on host

```

The operations intended to be executed on the device unit are queued and dispatched sequentially inside each stream *str*, but operations in different streams can be overlapped. On the other hand, the functions intended to run on the host unit are executed after checking the status of the corresponding coordination event—which indicates the availability of the input data. Once all the intended operations are queued, and data is successfully transferred to the device, the following four kernels are launched sequentially. We set the execution configurations of these kernels after carrying out several tests to obtain the optimal combination that minimises processing time [79].

3.3.4.2. Kernel-1 (device): preprocessing angular differences

This kernel filters out the least similar minutiae according to the similarity of the angular direction, as presented in Algorithm 3.2. This idea lies in the fact that if, for example, a minutia is chosen in the fingerprint impression with x-direction, we will only have to match it against those minutiae from the latent fingerprint with a direction close to x. Therefore, the preprocessing of the latent fingerprint involves grouping its minutiae according to their directions to be used consequently in Kernel-2, avoiding checking the condition in processing, hence saving time. To this purpose, the LUT_D look-up table is used, where each row corresponds to a quantized angle $\hat{\gamma} = 0 \dots z$ that coincides with the range $[0, 2\pi]$ and will contain all the minutiae indexes from the latent fingerprint that meet the condition [15]. That is, if the minimal angular difference (Equation A.1 in

Algorithm 3.2 *Kernel-1 in charge of the LUT_D computation.*

```

1:  $\hat{\lambda} = (z \cdot \lambda) / (2\pi)$                                 ▷ quantized angular difference threshold.
2:  $\hat{\gamma} = tid$                                            ▷ each thread handles a quantized angle
3:  $i = 0$ 
4: for each  $q_h \in L, h = 0 \dots |L|$  do                ▷ iteration over latent minutiae
5:    $\hat{\theta}_h = (z \cdot \theta_h) / (2\pi)$                     ▷ quantized minutia direction
6:   if  $d_\theta(\hat{\theta}_h, \hat{\gamma}) < \hat{\lambda}$  then                    ▷ check quantized angular difference
7:      $LUT_D[\hat{\gamma}][i] = h$                                 ▷ include latent minutia index
8:     Update  $i = i + 1$ 
9:   end if
10: end for
11:  $LUT_D[\hat{\gamma}][i] = \xi$                                 ▷ indicate end of array
```

Appendix A) between the minutia direction and the corresponding $\hat{\gamma} = tid$ is below the quantized λ threshold. Regarding the execution of this kernel, only one block of threads with z threads is launched—one thread per quantized angle.

3.3.4.3. Kernel-2 (device): matching minutiae descriptors

This kernel finds a first set of matching minutiae pairs using the operations shown in Algorithm 3.3. Each thread manages a particular minutia from the fingerprint impression $p_t \in T, t = tid$ with a quantized direction $\hat{\theta}_t$ (Definitions

Algorithm 3.3 *Kernel-2 in charge of the local matching process.*

```

1: for each  $q_h \in L, h = 0 \dots |L|$  do
2:   Store  $\nu_h$  in shared memory           $\triangleright$  latent minutiae cylinders in shared memory
3: end for
4: while  $tid < m$  do                                 $\triangleright$  prevent segmentation fault
5:   Set  $maxSim$  to  $max\{float\}$                          $\triangleright$  store max. similarity value
6:    $maxIx = \xi$                                           $\triangleright$  empty latent minutia index
7:    $T[tid] \leftarrow p_t$                                 $\triangleright$  impression minutia
8:   Store  $\nu_t$  in local memory                         $\triangleright$  impression minutia cylinder in local memory
9:    $\hat{\theta}_t = (z \cdot \theta_t) / (2\pi)$                      $\triangleright$  quantized angle of impression minutia
10:   $i = 0$ 
11:  while  $LUT_D[\hat{\theta}_t][i] \neq \xi$  do           $\triangleright$  go through the  $(\hat{\theta}_t)$ -th row of the look-up table
12:     $k = LUT_D[\hat{\theta}_t][i]$                              $\triangleright$  minutia index in latent
13:     $sim = \sigma(q_k, p_t)$                            $\triangleright$  similarity function
14:    if  $sim > maxSim$  then
15:       $maxSim = sim$                                    $\triangleright$  local storage of max. similarity value
16:       $maxIx = k$                                         $\triangleright$  local storage of minutia index in latent
17:      Update  $i = i + 1$ 
18:    end if
19:  end while
20:   $MatchingValue[tid] = maxSim$                          $\triangleright$  save similarity value
21:   $MaxMtiaL[tid] = maxIx$                               $\triangleright$  save latent minutia index
22:  Update  $tid = tid + C_t \cdot C_b$ 
23: end while

```

1 and 2 in Appendix A) and compares it with each minutia similar in direction from the latent fingerprint $q_h \in LUT_D[\hat{\theta}_t]$ (resulting from the execution of Kernel-1). Finally, the index and the matching value of the most similar minutia from the latent fingerprint are stored in two global arrays inside the particular cell corresponding to each minutia from the database of fingerprint impressions. This selection is based on the function described in Equation A.4 in Appendix A.

The cylinders within the latent fingerprint are stored in shared memory per block to be loaded faster in processing, the same way the target minutia cylinder handled by the thread is stored in thread-local memory. Regarding the execution configuration, this kernel is launched using $C_t/2$ threads per block not to exceed the maximum register size and optimise the available resources.

3.3.4.4. Kernel-3 (device): minutia quality computation

This kernel computes a quality value for each processed minutia as given in Algorithm 3.4. In practice, each thread takes a particular minutia $p_t \in T, t = tid$ and obtains a quality value as a function of the distance between this one and the surrounding minutiae that form its neighbourhood [77]. First, nearest H_m

Algorithm 3.4 *Kernel-3 in charge of computing the minutia quality.*

```

1: while  $tid < m$  do                                ▷ prevent segmentation fault
2:   Set array of  $H_m$  length  $D = \{0\}$                 ▷ initialise array of distances
3:    $T[tid] \leftarrow p_t$                                 ▷ impression minutia
4:    $s = LUT_S[t]$                                        ▷ starting position of the  $t$ -th impression
5:    $e = LUT_S[t + 1] - 1$                                ▷ ending position of the  $t$ -th impression
6:   for each  $p_h \in T, h = s \dots e$  do                ▷ iteration over all minutiae from s to e
7:      $d = d_e(p_h, p_t)$                                 ▷ Euclidean distance
8:     if  $(h \neq t) \wedge (d < distance)$  then
9:       Update  $D$  with  $d$                                 ▷ include  $d$  in sorted  $D$ 
10:    end if
11:  end for
12:  Compute  $\bar{d}$  from the values in  $D$                     ▷ average Euclidean distance
13:   $QualityT[tid] = \rho(\bar{d})$                             ▷ save minutia quality
14:  Update  $tid = tid + C_t \cdot C_b$ 
15: end while

```

minutiae are found using the Euclidean distance and included in the D array, which holds the distance values in ascending order. Then, the average distance value (\bar{d}) is used to obtain the final quality score for each minutia according to the piecewise function ρ , which depends on the H_{q1} and H_{q2} thresholds determined by experience (Equations A.3 and A.5 in Appendix A).

Likewise, this kernel also computes a quality value for each minutia in the latent fingerprint. In this case, the kernel is launched once at the beginning of execution using C_t threads per block to optimise available resources.

Algorithm 3.5 Kernel-4 in charge of finding clusters of matching minutiae pairs. Figure 3.12b can be helpful for the correct understanding of this kernel.

```

1: while  $tid < m$  do                                     ▷ prevent segmentation fault
2:    $s = LUT_S[t]$                                          ▷ starting position of the  $t$ -th impression
3:    $e = LUT_S[t + 1] - 1$                                  ▷ ending position of the  $t$ -th impression
4:    $T[tid] \leftarrow p_t$                                    ▷ impression minutia
5:    $h = \text{maxMtiaL}[t]$                                    ▷ read similar minutia from latent
6:   if  $h \neq \xi$  then  $f_1 = 1$  else  $f_1 = 0$  and  $h = 0$     ▷ avoid divergence
7:    $\text{ClusterMtiaK}[t][0] = t$ 
8:    $i = 1$ 
9:   for each  $p_k \in T, k = s \dots e$  do                 ▷ iterate over known minutiae pairs
10:     $r = \text{maxMtiaL}[k]$                                    ▷ read similar minutia from latent
11:    if  $r \neq \xi$  then  $f_2 = 1$  else  $f_2 = 0$  and  $r = 0$     ▷ avoid divergence
12:     $q'_r = \psi(q_r, q_h, p_t)$                            ▷ new projected minutia
13:     $\text{sim} = \sigma_e(q'_r, p_k) \cdot \sigma_\theta(q'_r, p_k) \cdot \sigma_t(q'_r, p_k)$  ▷ check new similarity
14:    if  $(f_1 \cdot f_2 \cdot \text{sim}) > 0$  then                 ▷ avoid divergence
15:       $\text{ClusterMtiaK}[t][i] = k$                              ▷ save impression minutia index
16:      Update  $i = i + 1$ 
17:    end if
18:  end for
19:   $\text{ClusterCount}[tid] = i$                                ▷ save number of minutiae pairs in cluster
20:  Update  $tid = tid + C_t \cdot C_b$ 
21: end while

```

3.3.4.5. Kernel-4 (device): finding clusters of similar minutiae pairs

This kernel finds clusters of similar minutiae pairs after checking for alignments using the initial set of found minutiae pairs (Algorithm 3.5 and Figure 3.12). First, each thread takes a minutia $p_t \in T, t = tid$ and reads the most similar minutia from the latent fingerprint previously found through the execution of Kernel-2. Therefore, two scenarios are possible: (i) every minutia in T has a match in L , or (ii) not every minutia in T has a similar one in L . In the first case, the workload is balanced between all the threads, so there is no per-

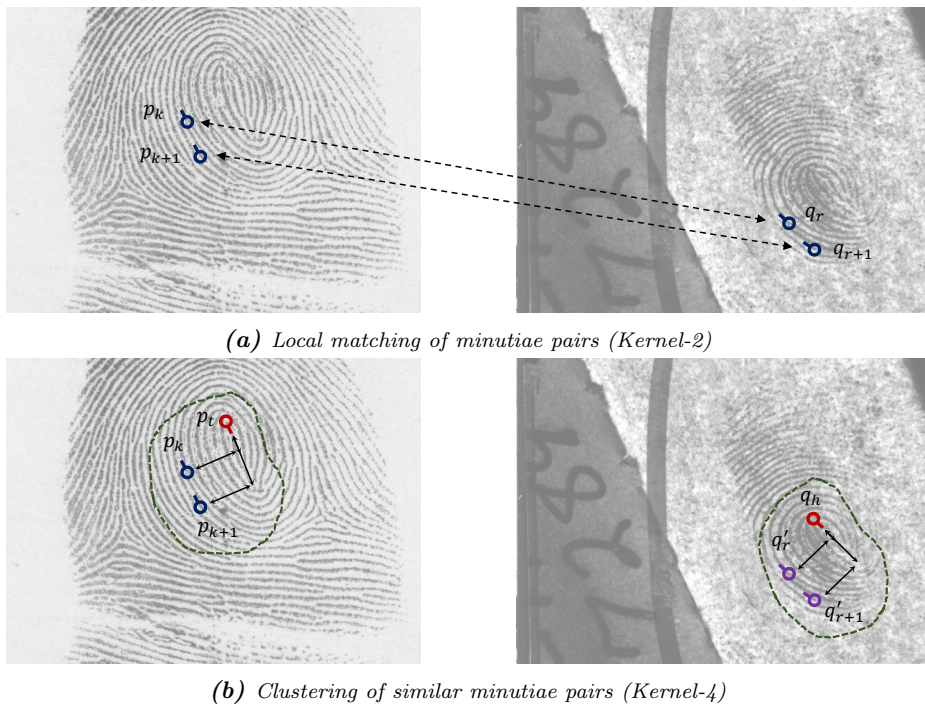


Figure 3.12: Difference between the local matching and clustering stages using the rolled (left) and latent (right) fingerprints. In (a), similar minutiae pairs are found according to their minutiae descriptors. In (b), minutiae pairs—previously found in the first stage—are processed again to check whether they still match based on the distance and angular relationship with another minutia pair as the cluster’s centre. Minutiae in blue have no relevant role, minutiae in red are the centre of the particular cluster represented by the green wrap-around line, and minutiae in purple are projected ones taking the centre of the cluster as reference.

formance degradation. Conversely, the second suffers from the thread divergence problem as a few threads carry on with the processing whereas others are idle. To address this problem, the first minutia from the latent fingerprint is selected as a dummy structure for those minutiae in T that do not have a similar one in L . Thus, the thread divergence problem is minimised since broad if-else statements are avoided.

Once we have solved the thread divergence problem, clusters of minutiae pairs are obtained by performing several alignments to test the consistency of the minutiae pairs found so far following the expressions from Equations A.6-A.9 in Appendix A. Finally, a set of minutiae indexes from the impression fingerprints is obtained and stored in the *ClusterMtiaK* vector, along with the number of minutiae in *ClusterCount*. The host unit then reads the *ClusterMtiaK* vector directly to speed up the process of building the clusters of minutiae pairs in the *FES* function, which is addressed in the next section. Regarding the execution of Kernel-4, it is launched using the same configuration as Kernel-3.

3.3.4.6. FES function (host): final evaluation stage

As above explained, the partial results obtained from the processing of a particular batch of impressions on the device are used as input to the final evaluation stage (*FES*) function executed on the host (Algorithm 3.6). This function carries out the final part of the fingerprint matching process—executed in parallel at the fingerprint matching level. In every fingerprint comparison, the minutiae pairs found through the execution of Kernel-2 and formed by each impression minutia and the most similar one from the latent fingerprint (stored in the *MaxMtiaL* vector) will be placed as the centroid of the corresponding cluster.

The clusters of minutiae pairs are built by reading the impression minutiae indexes previously stored in the *ClusterMtiaK* vector and considering the number of minutiae included in the cluster and stored in *ClusterCount* for each minutia inside the impression fingerprint. Afterwards, a weight value is obtained for each cluster. Then, those containing similar minutiae pairs will be merged to obtain a consolidated set of matching minutiae pairs, contributing to the final similarity score between fingerprints.

Finally, a TPS model is built from the consolidated set of minutiae pairs to correct any deformation affecting the latent fingerprint image, making it possible to find new minutiae pairs. These new minutiae pairs will also contribute to the final similarity value between fingerprints. It should be pointed out that the computational load in the host is reduced, as finding the minutiae pairs belonging

to each cluster is one of the most time-consuming tasks of the algorithm and is carried out beforehand on the device unit. In fact, while the *FES* function is being executed on the host over a particular batch of impressions, the device will have finished processing the next one and delivered the results to the host so that idle times are removed.

Algorithm 3.6 *Host function in charge of the final evaluation stage (FES).*

```

1: for each  $fp \in T$  in parallel do                                ▷ iterate over fingerprint impressions
2:    $s = LUT_S[fp]$                                               ▷ starting position of the  $fp$ -th impression
3:    $e = LUT_S[fp + 1] - 1$                                        ▷ ending position of the  $fp$ -th impression
4:   for each  $p_k \in T, k = s \dots e$  do                            ▷ iterate over impression minutiae
5:      $h = maxMtiaL[k]$                                            ▷ read similar minutia from latent
6:     if  $h \neq \xi$  then
7:        $M = M \cup (q_h, p_k)$                                    ▷ include minutiae pair in vector
8:     end if
9:   end for
10:  for each  $p_k \in T, k = s \dots e$  do                            ▷ iterate over impression minutiae
11:    for  $i = 0, ClusterCount[k]$  do                                ▷ iterate over  $k$ -th row of ClusterMtiaK
12:       $r = ClusterMtiaK[k][i]$                                    ▷ read impression minutia in cluster
13:       $h = maxMtiaL[r]$                                            ▷ read similar minutia from latent
14:       $\forall (q_h, p_r) \in M : B_h = B_h \cup (q_h, p_r)$            ▷ build cluster vector
15:    end for
16:    Perform Step 3b from Section 3.2.5.1                        ▷ sort minutiae in clusters
17:  end for
18:  Perform Steps 3c-5                                           ▷ weigh and merge clusters, build TPS and evaluate
19:  Update Similarity[ $fp$ ]                                       ▷ final similarity value
20: end for

```

3.4. Experiments and results

This section analyses and compares ALFI with the most similar approaches reported in the literature in terms of accuracy and computational performance on widely used databases.

Firstly, the experimental setup and the different databases used in the experiments are presented at the beginning of this section. Then, the accuracy of ALFI is assessed considering the latent fingerprint identification task and, additionally, the verification task. Afterwards, the computational performance of ALFI is measured in terms of execution time and speed-up on Linux and Windows operating systems. Finally, we present the timeline of the application and discuss the results accomplished.

3.4.1. Experimental setup

This research focuses on the design of a new methodology for latent fingerprint identification specifically designed for heterogeneous CPU-GPU systems. With this in mind and after considering the published works in this area so far, we can conclude that:

- DMC [33] is the latent fingerprint identification algorithm that has demonstrated the best results when working with every combination of available fingerprint databases, even in the case of considering a background database of more than 1.1 million impressions [33].
- Apart from its excellent performance in identifying fingerprints, DMC is the algorithm with the second-best performance in the field of fingerprint verification in the FMISO-HARD-1.0 competition of the FVC-onGoing platform [80, 81]—among those developed by academic groups. The algorithm with the best performance in this competition is the MntModel [77]; however, we cannot replicate it since the article omits several steps of the development of the algorithm. Furthermore, this algorithm was not tested on latent fingerprint identification, and its performance in carrying out this particular task is unknown.
- Only the works described in [70] and [33] provide the source code or program that allows researchers to replicate the results with different databases, and therefore the proposal of this work can only be compared with the numbers reported in their articles.
- The DMC-CC version uses the MCC descriptor, which is shown in a recent study to be the best minutiae descriptor for identifying latent fingerprints [32].

To carry out a thorough analysis, we used three heterogeneous CPU-GPU systems (S_{1-3}), whose specifications are given in Tables 3.5 and 3.6.

Table 3.5: Specifications of the host units used in the experiments.

Parameter	S ₁ (Linux)	S ₂ (Both)	S ₃ (Windows)
Processor Type	Intel Xeon	Intel Core	AMD Ryzen
Processor Model	E5-2698 v3	i5-8600K	7-1700x
Number of cores	16	6	8
Number of threads	32	6	16
Frequency (GHz)	2.3	3.6	3.4
Memory RAM (GB)	256	8	16
Cache L1 (kB)	8x64	6x64	8x96
Cache L2 (kB)	8x256	6x256	8x512
Cache L3 (MB)	1x40	1x9	2x8

Table 3.6: Specifications of the device units used in the experiments.

Parameter	S ₁ (Linux)	S ₂ (Both)	S ₃ (Windows)
Model	GTX 980	GT 1030	GTX 1050-Ti
Architecture	Maxwell	Pascal	Pascal
Number of CUDA cores	2048	384	768
Number of SMs	16	3	6
Base clock (MHz)	1.12	1.22	1.12
Global memory (GB)	4	2	4
Memory per block (kB)	48	48	48
Max. threads per block	1024	1024	1024
Threads per warp	32	32	32
Memory bandwidth (GB/s)	224	48	112
Performance (TFLOPs)	4.6	1.13	2.14

Regarding the implementation details, we developed ALFI using C++ and CUDA C++. These are compiled programming languages, i.e., they are translated into machine language before being executed. This fact makes it possible to test the performance of ALFI on both Linux and Windows operating systems using the same implementation. In addition, according to a previous study presented in [82], using C++ significantly improves the computational performance of the latent identification task.

Host codes are compiled with -O2 optimisation flag using the g++ 5.4 and MSVC 14.16.27023 compilers for Linux (Ubuntu 16.04.5 LTS) and Windows 10, respectively. Device codes make use of the NVIDIA NVCC compiler from the CUDA compilation tools V10.0.130. The OpenMP C/C++ version 2.0 is used inside the FES function to enable the multi-threaded execution at the fingerprint matching level. The Armadillo C++ library version 7.800.2 [83, 84] with OpenBLAS 0.2.14.1 is also used to carry out the necessary linear algebra operations.

3.4.2. Databases

We used the popular NIST SD27 [85] database, which includes fingerprints and minutiae, to test the identification performance of ALFI. This widely used database holds 258 latent fingerprints collected from real cases with the images available at 500 dpi. Every case includes the image of the latent fingerprint and its rolled fingerprint mate, where experts have validated all the minutiae. Moreover, six background databases have been designed according to different combinations of fingerprints, as shown in Table 3.7. The NIST SD27 database is further extended with rolled fingerprints from the NIST SD4 [86] and NIST SD14 [87] databases to obtain small (B_{1-3}) and medium (B_{4-5}) sized databases.

In order to obtain a more extensive background database, synthetic plain fingerprints generated using the SFinGe Version 4.1 (build 1746) Demo software were included in B_6 . The fingerprints generated with this last software have been used in several fingerprint verification competitions, proving that the results achieved with these features are similar to the ones achieved on real

Table 3.7: Background databases used in the experiments with their number of fingerprints included. The two right-most columns show the total number of fingerprints and the average number of minutiae extracted per fingerprint, respectively.

Database	NIST SD27	NIST SD4	NIST SD14	Synthetic	N° fps.	N° mtiae./fp.
B_1	258	-	-	-	258	21
B_2	258	-	2,000	-	2,258	149
B_3	258	2,000	-	-	2,258	101
B_4	258	-	27,000	-	27,258	163
B_5	258	2,000	27,000	-	29,258	159
B_6	258	2,000	27,000	357,985	387,243	38

databases [88]. However, plain fingerprints contain less information than rolled ones, which can affect the accuracy and computational performance of the experiments. Regarding minutiae per fingerprint, they are extracted using the VeriFinger SDK [89] for the impression fingerprints.

Even though ALFI is designed for fingerprint identification, we also tested its accuracy on verification databases as well to check whether it is suitable for this particular task. The FVC 2002 [90], FVC 2004 [91] and FVC 2006 [92] databases are used in the verification experiment, where the DB1_A dataset from FVC 2006 was discarded due to the low resolution of the images.

We assessed the computational performance of ALFI under conditions that are as close as possible to a real case. In particular, several of the six background databases (from Table 3.7) have a similar number of fingerprints; hence, some of them can be dismissed to eliminate redundancy. Therefore, medium and large-sized background databases related to B_3 , B_5 and B_6 are considered since they have a representative number of fingerprints.

3.4.3. Accuracy analysis

This experiment presents the accuracy results of ALFI using identification and, additionally, verification databases. ALFI is compared with the state-of-the-art DMC algorithm using the following descriptors: MCC (DMC-CC), M-Triplets (DMC-MT), and Neighboring Minutiae-based Descriptor (DMC-NMD).

3.4.3.1. Identification databases

Cumulative Match Characteristic (CMC) curves, described in [53], are the standard methods used to assess the accuracy of identification algorithms that produce an ordered list of possible matches. This type of result plots the probability that a correct identification takes place (rank- k identification rate) within a group of k returned candidates, where $k = 1...20$. In practice, latent fingerprint examiners may request (i) all returned candidates with a match score above a certain threshold or (ii) a specific number of highest-ranked candidates instead. In any case, examiners normally begin the analysis with the candidate that has the highest rank (rank-1) and continue through the remaining ones if they do not succeed [93]. The search ends when the fingerprint mate is found or after all candidates are analysed. Therefore, not only the rank-1 accuracy is crucial for the identification evaluation, but also the rank-20 and the complete CMC curve to make the experiment as close as possible to a real case.

In this experiment, each latent fingerprint from the NIST SD27 is compared with all impression fingerprints in the background database generating the CMC curves shown in Figure 3.13. These results are complemented by the corresponding rank-1 and rank-20 values presented in Tables 3.8 and 3.9. From these results, we can make the following observations:

- In most cases, the DMC-CC algorithm is the best ranked; however, the difference in accuracy between this version and ALFI is negligible.
- Compared to the DMC-MT algorithm, ALFI outperforms it by approximately 1.6%, 1.6%, and 1.2% on databases B_1 , B_2 , and B_3 , respectively, considering rank-1 accuracy. For rank-20 accuracy, ALFI outperforms the same algorithm by approximately 2.7%, 0.8%, 3.9%, and 3.9% on databases B_2 , B_3 , B_4 , and B_5 , respectively.
- Compared to the DMC-NMD algorithm, ALFI outperforms it by approximately 0.8%, 1.2%, 1.2%, and 0.8% on databases B_1 , B_2 , B_4 , and B_5 , respectively, considering rank-1 accuracy. For rank-20 accuracy, ALFI outperforms the same algorithm by approximately 0.4%, 2.3%, 0.4%, 1.9%, and 2.3% on databases B_1 , B_2 , B_3 , B_4 , and B_5 , respectively.

Table 3.8: Rank-1 accuracy of ALFI and DMC given in percentages (CMC curves shown in Figure 3.13).

Algorithm	B ₁	B ₂	B ₃	B ₄	B ₅	B ₆
DMC-CC	87.21	81.01	80.62	70.16	69.77	62.79
DMC-MT	82.95	76.74	76.36	69.38	69.38	66.67
DMC-NMD	83.72	77.13	79.46	66.67	66.67	62.79
ALFI	84.50	78.29	77.52	67.83	67.44	61.24

Table 3.9: Rank-20 accuracy of ALFI and DMC given in percentages (CMC curves shown in Figure 3.13).

Algorithm	B ₁	B ₂	B ₃	B ₄	B ₅	B ₆
DMC-CC	94.57	91.47	91.09	85.27	84.50	78.68
DMC-MT	93.41	87.60	89.15	79.84	79.84	77.13
DMC-NMD	92.25	87.98	89.53	81.78	81.40	78.29
ALFI	92.64	90.31	89.92	83.72	83.72	75.58

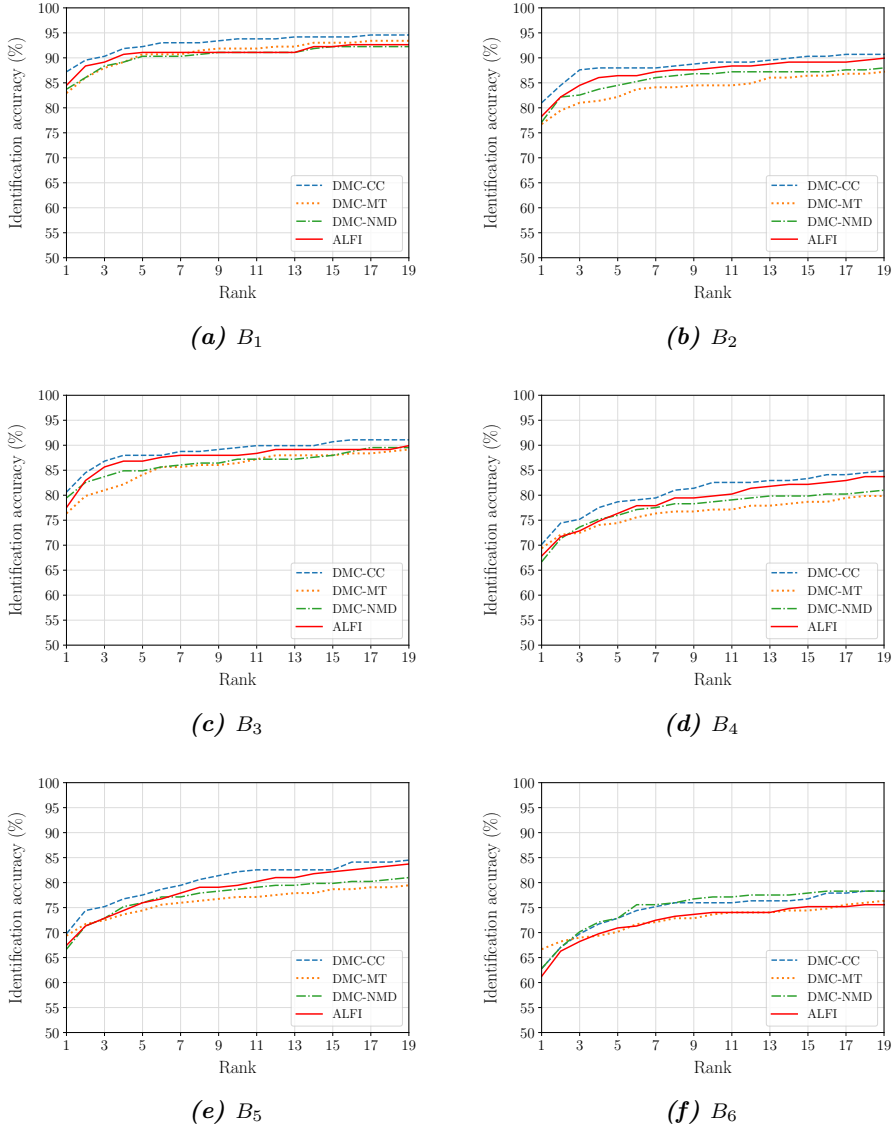


Figure 3.13: Cumulative Match Characteristic (CMC) curves of ALFI and DMC using the NIST SD27 database as reference and six different background databases B_1 – B_6 described in Table 3.7.

3.4.3.2. Verification databases

Although ALFI is a methodology developed specifically for latent identification, its accuracy on fingerprint verification databases is also evaluated. The FVC 2002, FVC 2004, and FVC 2006 databases are used for this purpose, along with the performance evaluation proposed by Cappelli *et al.* [88] based on EER, FMR100, FMR1000, and ZeroFMR indicators, where lower values are related to better performance. The results of this experiment are given in Tables 3.10-3.12. From them, the following observations may be made:

- Compared to the DMC-CC algorithm, ALFI performs equal to or better than it for 5 accuracy measurements.
- Compared to the DMC-MT algorithm, ALFI performs equal to or better than it for 15 accuracy measurements.
- Compared to the DMC-NMD algorithm, ALFI performs equal to or better than it for 21 accuracy measurements.

Table 3.10: Accuracy results of ALFI and DMC on the FVC 2002 databases [90].

Database	Algorithm	EER (%)	FMR100 (%)	FMR1000 (%)	ZeroFMR (%)
DB1_A	DMC-CC	0.55	0.64	0.79	1.18
	DMC-MT	0.65	0.79	1.11	1.25
	DMC-NMD	0.50	0.61	0.79	1.14
	ALFI	0.55	0.79	1.00	1.54
DB2_A	DMC-CC	0.50	0.50	0.71	1.00
	DMC-MT	0.43	0.61	0.75	0.79
	DMC-NMD	0.60	0.61	0.86	1.04
	ALFI	0.59	0.68	1.00	1.36
DB3_A	DMC-CC	2.27	2.43	3.71	4.82
	DMC-MT	2.54	3.18	4.07	5.18
	DMC-NMD	2.39	3.11	4.32	4.64
	ALFI	2.67	3.21	4.14	6.00
DB4_A	DMC-CC	1.08	1.18	1.89	2.18
	DMC-MT	1.51	1.86	2.68	3.79
	DMC-NMD	1.58	1.79	2.50	2.75
	ALFI	1.28	1.43	2.36	3.21

Table 3.11: Accuracy results of ALFI and DMC on the FVC 2004 databases [91].

Database	Algorithm	EER (%)	FMR100 (%)	FMR1000 (%)	ZeroFMR (%)
DB1_A	DMC-CC	3.24	5.39	9.79	17.39
	DMC-MT	3.76	6.36	10.25	15.46
	DMC-NMD	3.62	6.04	12.14	17.75
	ALFI	3.41	5.18	11.39	16.21
DB2_A	DMC-CC	4.18	5.96	9.00	10.68
	DMC-MT	4.23	5.68	8.21	10.04
	DMC-NMD	4.52	6.11	8.96	13.21
	ALFI	4.52	6.86	9.89	11.46
DB3_A	DMC-CC	2.74	4.07	5.96	9.54
	DMC-MT	3.38	4.79	8.32	12.46
	DMC-NMD	2.78	4.79	10.14	15.89
	ALFI	2.77	3.93	6.36	7.82
DB4_A	DMC-CC	2.15	2.82	3.89	4.46
	DMC-MT	2.91	3.25	3.96	4.89
	DMC-NMD	2.80	3.32	4.36	4.86
	ALFI	2.91	3.50	4.46	5.75

Table 3.12: Accuracy results of ALFI and DMC on the FVC 2006 databases [92].

Database	Algorithm	EER (%)	FMR100 (%)	FMR1000 (%)	ZeroFMR (%)
DB2_A	DMC-CC	0.42	0.35	0.50	1.18
	DMC-MT	0.36	0.37	0.50	1.31
	DMC-NMD	0.51	0.42	0.78	1.88
	ALFI	0.48	0.42	0.60	1.39
DB3_A	DMC-CC	3.36	4.46	6.76	10.69
	DMC-MT	3.51	4.95	7.80	12.44
	DMC-NMD	3.39	4.64	8.12	14.13
	ALFI	3.70	5.25	7.96	11.76
DB4_A	DMC-CC	2.52	3.13	5.91	8.17
	DMC-MT	2.75	3.51	5.07	11.13
	DMC-NMD	2.57	3.32	6.30	8.83
	ALFI	3.10	3.79	7.66	10.43

Considering the high variability in the above accuracy results, it could be pointed out that the accuracy of ALFI in verification is within the same range as that obtained by the DMC algorithm, even though ALFI is intended for latent identification.

3.4.4. Computational performance analysis

This section aims to compare the computational performance of ALFI with the results obtained by the state-of-the-art algorithm. The DMC-CC algorithm is chosen for this comparison due to its better performance compared to DMC-MT and DMC-NMD, as stated in [33].

In this experiment, a random latent fingerprint from the NIST SD27 database is matched against the B_3 , B_5 , and B_6 background databases (described in Section 3.4.2) using the S_{1-3} systems (Section 3.4.1). We measured the time required to complete this task and the average throughput in processing. The latter parameter is measured in KMPS, which stands for Kilo Matches Per Second. The results of this experiment are presented according to the operating system, either Linux or Windows, on which the computational performance is measured.

3.4.4.1. Linux

The results of this experiment are shown in Table 3.13 and Figure 3.14 in terms of execution time and throughput, respectively. The baseline DMC-CC algorithm has been ported from C# to C++ to ensure a fair comparison on Linux. The reason for this choice is that the original C# code could not run efficiently on Linux, making it impossible to fairly compare ALFI with the one presented by the authors in [33]. The best results in each system are as follows:

- **On S_1 :** ALFI is on average 28.9x faster than the DMC-CC algorithm on database B_5 . The maximum throughput value is 31.13 KMPS, which is achieved by ALFI on database B_6 .
- **On S_2 :** ALFI is on average 20.6x faster than the DMC-CC algorithm on database B_6 . The maximum throughput value is 43.66 KMPS, which is achieved by ALFI on the same database.

Table 3.13: Average run-time and speed-up results of ALFI and DMC-CC (baseline algorithm) on Linux. A random latent fingerprint from the NIST SD27 [85] is matched against three background databases using two systems with Linux OS.

		S ₁		S ₂	
Database	Algorithm	Time (s)	Speed-up	Time (s)	Speed-up
B ₃	DMC-CC	4.36	-	2.63	-
	ALFI	0.66	6.6	0.36	7.3
B ₅	DMC-CC	91.69	-	57.37	-
	ALFI	3.17	28.9	3.57	16.1
B ₆	DMC-CC	292.85	-	183.08	-
	ALFI	12.44	23.5	8.87	20.6

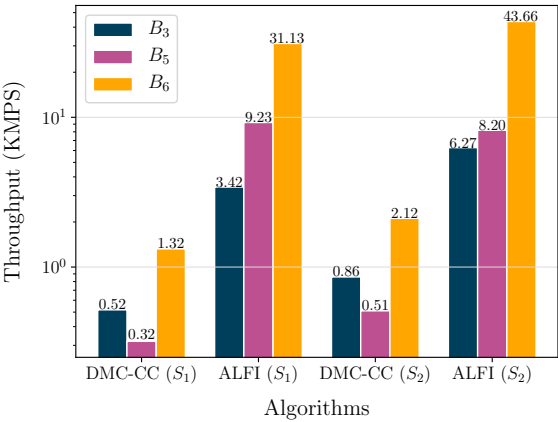


Figure 3.14: Throughput results of ALFI and DMC-CC on Linux. A random latent fingerprint from the NIST SD27 [85] is matched against three background databases using two systems with Linux OS.

3.4.4.2. Windows

The results of this experiment are shown in Table 3.14 and Figure 3.15. The reference DMC-CC algorithm for Windows is the C# implementation presented by their authors. The best results in each system follow:

- **On S_2 :** ALFI is on average 29.2x faster than the DMC-CC algorithm on database B_6 . The maximum throughput value is 23.89 KMPS, which is achieved by ALFI on the same database.
- **On S_3 :** ALFI is on average 44.7x faster than the DMC-CC algorithm on database B_6 . The maximum throughput value is 24.29 KMPS, which is achieved by ALFI on the same database.

Table 3.14: Average run-time and speed-up results of ALFI and DMC-CC (baseline algorithm) on Windows. A random latent fingerprint from the NIST SD27 [85] is matched against three background databases using two systems with Windows OS.

		S_2		S_3	
Database	Algorithm	Time (s)	Speed-up	Time (s)	Speed-up
B_3	DMC-CC	6.61	-	9.48	-
	ALFI	0.61	10.8	0.66	14.4
B_5	DMC-CC	146.98	-	209.22	-
	ALFI	6.71	21.9	5.46	38.3
B_6	DMC-CC	472.66	-	712.78	-
	ALFI	16.21	29.2	15.94	44.7

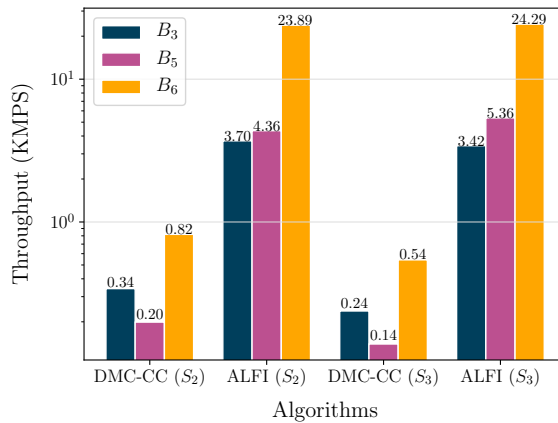


Figure 3.15: Throughput results of ALFI and DMC-CC on Windows. A random latent fingerprint from the NIST SD27 [85] is matched against three background databases using two systems with Windows OS.

3.4.5. Application timeline

Here, we used the NVIDIA Visual Profiler (NVVP) to generate the ALFI timeline regarding the identification process. As an example, this experiment considers the B_5 background database, S_2 with Linux OS, and 1,200 fingerprints processed in each CUDA stream on the device.

Figure 3.16 displays the first second of execution of the ALFI timeline. It visually demonstrates how the host successfully executes the FES function of ALFI on a particular batch of fingerprints while the device is busy processing and transferring the next two batches. On the device, the operations are executed in two non-default streams.

Figure 3.17 best illustrates this asynchronous process by showing two enlarged images of the ALFI timeline. Three kernels (K_{2-4})¹ are executed in one stream while the other stream transfers to the host the partial results obtained in the previous iteration (Figure 3.17a). In the next iteration, the streams exchange tasks with each other (Figure 3.17b) in a process that repeats until the execution is complete.

The proposed method reduces the CPU load by sharing it among the resources available on almost any current computer.

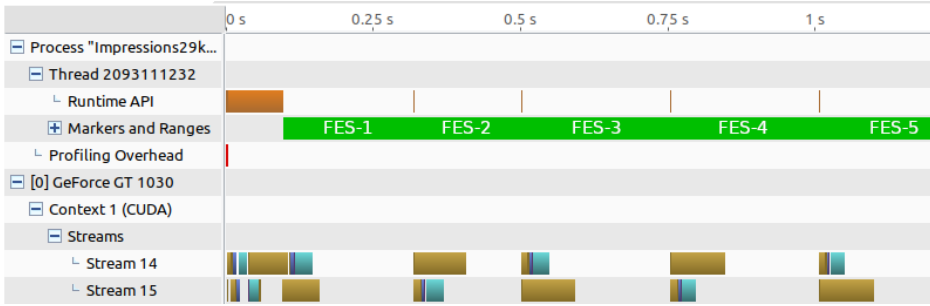


Figure 3.16: First second of execution of the ALFI timeline. The host execution overlaps with the execution in two CUDA streams on the device. FES- n stands for the final evaluation stage function over the n -th batch of fingerprints, K for kernel, and D2H and H2D for device-to-host and host-to-device data transfers. This image is zoomed in on Figure 3.17.

¹ K_1 is only launched once at the beginning of execution.

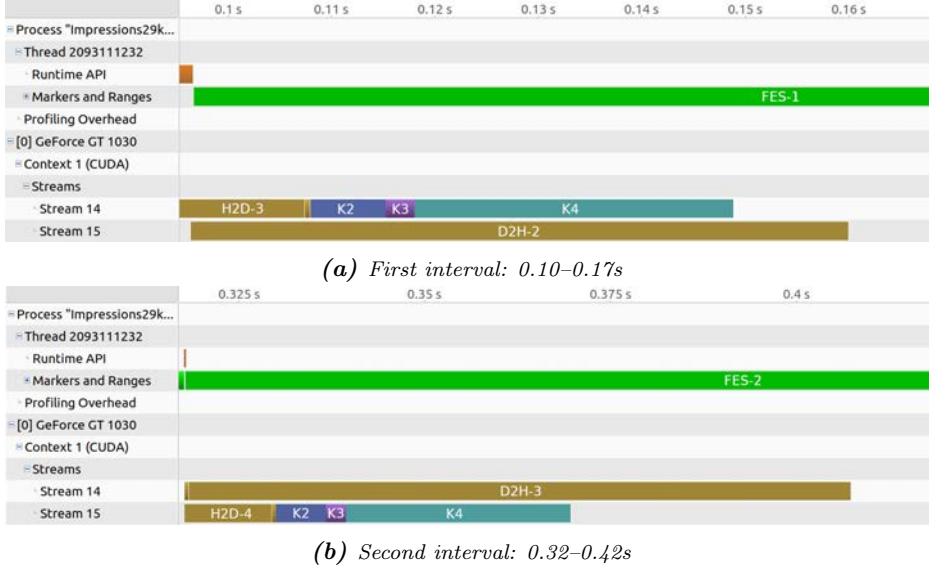


Figure 3.17: Enlarged images of the ALFI timeline given by intervals (Figure 3.16). FES- n stands for the final evaluation stage function over the n -th batch of fingerprints, K for kernel, and D2H and H2D for device-to-host and host-to-device data transfers. The time intervals given are approximate.

3.4.6. Analysis of the results

The primary goal of the development of ALFI was to obtain the best possible computational performance in latent identification without compromising accuracy. Accordingly, ALFI has accomplished significant results in both terms.

3.4.6.1. Accuracy

Regarding the latent fingerprint identification problem, the accuracy values of ALFI are within the same range as the ones obtained by the state-of-the-art algorithms, as given in Tables 3.15 and 3.16. However, the inclusion of the GPU in processing results in a slight accuracy reduction for some background databases and k -ranks compared to the reference algorithm in latent identification. The reason for this lies in the impossibility of developing a dynamic algorithm for GPU processing, which profoundly affects the early stage of the processing where the first matching minutiae pairs must be found.

In the host code, it is possible to accept very different numbers of minutiae pairs—using dynamic memory allocation—from one fingerprint comparison to another without compromising performance. However, in the device code, a maximum number of minutiae pairs must be imposed to improve performance—static memory allocation—resulting in losing some possible matching minutiae pairs. This fact is more noticeable when the number of minutiae per fingerprint is reduced, as is the case with the B_6 background database in which something unusual happens as it contains more plain fingerprints than rolled. Nevertheless, this drawback is balanced with the significant improvement achieved in computational performance.

Table 3.15: Accuracy differences in the identification experiment obtained from analysing the rank-1 accuracy values given in Table 3.8. The lowest accuracy value of the four algorithms is taken as a reference for each background database. These algorithms are those with the (-) symbol.

Algorithm	B_1	B_2	B_3	B_4	B_5	B_6
DMC-CC	4.3%	4.3%	4.3%	3.5%	3.1%	1.6%
DMC-MT	-	-	-	2.7%	2.7%	5.4%
DMC-NMD	0.8%	0.4%	3.1%	-	-	1.6%
ALFI	1.6%	1.6%	1.2%	1.2%	0.8%	-

Table 3.16: Accuracy differences in the identification experiment obtained from analysing the rank-20 accuracy values given in Table 3.9. The lowest accuracy value of the four algorithms is taken as a reference for each background database. These algorithms are those with the (-) symbol.

Algorithm	B_1	B_2	B_3	B_4	B_5	B_6
DMC-CC	2.3%	3.9%	1.9%	5.4%	4.7%	3.1%
DMC-MT	1.2%	-	-	-	-	1.6%
DMC-NMD	-	0.4%	0.4%	1.9%	1.6%	2.7%
ALFI	0.4%	2.7%	0.8%	3.9%	3.9%	-

3.4.6.2. Computational performance

Considering execution time, ALFI has proved to outperform the state-of-the-art algorithm for every studied database and operating system in latent identification. This achievement lies in the fact that the workload is balanced between the CPU and GPU using asynchronous processing and fine-grained parallelism so that idle times are drastically reduced. By contrast, the state-of-the-art algorithm is designed for single-threaded execution and neglects the use of GPUs to accelerate the processing.

The throughput experiment revealed that this parameter increases with the size of the database for the ALFI methodology. Conversely, the throughput of the DMC-CC algorithm decreases between databases B_3 and B_5 , but increases with B_6 . The explanation lies in the difference in the nature of the fingerprints included in the databases and the non-linearity of their processing. In particular, B_6 only contains 7.6% of rolled fingerprints, and the rest are plain fingerprints, each of which includes less information and is therefore processed faster than the rolled fingerprints. Database B_6 has in average 38 minutiae per fingerprint, whereas databases B_3 and B_5 have 101 and 159, respectively, as given previously in Table 3.7.

4 Total Viewshed Computation

Viewshed analysis is the process of finding the area of terrain visible from a specific observer's location. This information is of great interest in environmental planning [94] and fire fighting [95] to obtain the best locations for settlements and watchtowers, as is the case in both examples. These types of problems are addressed using arrays of regularly spaced elevation values called *Digital Elevation Models* (DEMs). Algorithms working with this type of data require large numbers of memory accesses to 2D arrays, which, despite being regular, result in poor data locality in memory. This issue also affects any algorithm intended to solve one of the most computational demanding viewshed problems, called *total viewshed*. It comprises the computation of the viewshed for each point in the DEM, i.e., each point in the DEM will eventually work as an observer during processing—where a DEM typically contains more than several millions of points.

In this chapter, a new methodology called *skewed Digital Elevation Model* (sDEM) is presented, considering the total viewshed problem as a case study. sDEM applies a complete restructuring of the DEM data in memory before the total viewshed computation. In practice, we transform the DEM into a new structure called skwDEM in which data is aligned in memory to improve data locality in accessing the memory, hence increasing the speed of processing. This approach makes it unnecessary to use standard techniques that reduce computational cost in total viewshed problems, e.g., considering a maximum visibility distance within a circular area around each target location. In addition, sDEM could also enhance the performance of other algorithms that analyse relevant topographic features such as slope and elevation.

The contributions of this chapter are:

- We propose a new methodology called sDEM (skewed Digital Elevation Model) to achieve faster processing of terrain surface, which substantially improves data locality in memory. In particular, this approach fully exploits the intrinsic parallelism of the total viewshed computation, achieving maximum performance through efficient memory access.
- We present different implementations for single-core, multi-core, single-GPU, and multi-GPU platforms that are compared with the state-of-the-art tools and algorithms.

Section 4.1 presents an introduction on viewshed analysis. Section 4.2 addresses the state-of-the-art regarding the three main visibility problems. Section 4.3 presents the sDEM methodology specifically designed for solving the total viewshed problem on multi-GPU platforms. Finally, Section 4.4 assesses the sDEM proposal in terms of running time, speed-up, and throughput compared with the most commonly used geographic information systems and the state-of-the-art algorithm.

4.1. Introduction

This section will describe the necessary background knowledge related to viewshed analysis, with a particular focus on the total viewshed problem.

First of all, the current techniques used to represent the Earth's surface are introduced in this section. Secondly, the beginnings of geospatial analysis as a subject of research are described. Afterwards, we explain the history and definition of the Geographic Information System (GIS) software used to analyse geographic data, followed by the different types of digital terrain modelling used by the GIS software. Then, the visibility concept and its formulation are introduced so that the concept of viewshed can be explained later on, as well as the types described in the literature and key computational aspects. Finally, an overview of the most important concepts is presented.

4.1.1. Earth representation

The cartographic representation of the terrestrial globe is a challenging problem due to the impossibility of representing the entire Earth's surface without

deforming it. Throughout history, several different map projection methods have been developed to transform the Earth from its spherical shape (3D) to a planar one (2D). Furthermore, the relationship between a point on this two-dimensional projected map and the real location on the Earth's surface has to be found using a proper Coordinate Reference System (CRS) from those available. The decision to select a proper map projection method and CRS depends on the extent of the target area, the type of research to be carried out, and also on the availability of data, which can be sparse for specific areas.

4.1.1.1. Map projections

A straightforward solution would be representing the Earth's spherical surface as it is by using globes. Although this approach preserves the Earth's shape and the original size of every continent, using globes have several drawbacks, such as (i) serious difficulty in carrying them with you anywhere and (ii) small scales (e.g., 1:100 million) affects both measurement and accuracy.

Current datasets usually work with scales of around 1:250000 or more depending on the level of detail required by the target analysis. This scale value is impossible to achieve without deforming the Earth's shape. Thus, cartographers have designed several techniques called map projections, each with its advantages and disadvantages, to represent the Earth's spherical surface in two dimensions. This decision depends on the map's purpose, which affects the required scale. For instance, a map projection suitable for studying a country may not be for an entire continent due to the distortions. This fact shows how important it is to choose a proper map projection according to the needs of the study.

We can classify map projections according to (i) the shape onto which the Earth is projected and (ii) the characteristics preserved in the resulting map.

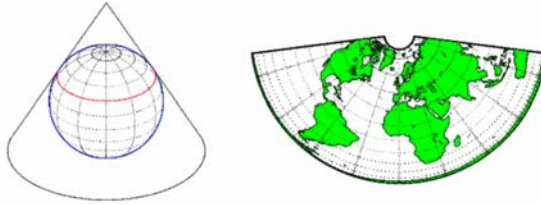
- Classification based on shapes

Each map projection is developed using different shapes over the Earth's surface, where the most common ones are cylinders, cones, and planes. The result from each of these methods is called a map projection family (Figure 4.1).

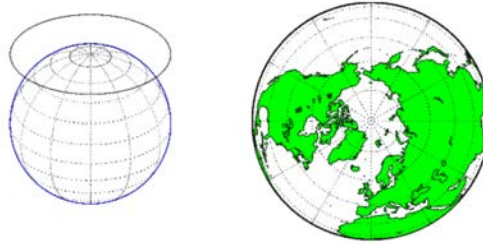
When developing a map projection, it is impossible to simultaneously preserve all characteristics such as distance, area, and angular conformity. In practice, every resulting map has to deal with distortions that may affect all or only some characteristics. For the case where all characteristics are affected, this always happens within some acceptable limits. Examples of this type are the Robinson



(a) Cylindrical projection



(b) Conical projection



(c) Planar projection

Figure 4.1: Map projection families according to the solid used for the projection [96].

and Winkel Tripel projections typically used to visualise world maps. This aspect causes that the selection of a map projection should be related to the variable of the study: if the purpose of our analysis is calculating the distance between two locations, we will use the map projection with higher accuracy for distances. The following are examples of map projections from each family:

- **Cylindrical:** Mercator, Transverse Mercator, Cylindrical equal-area, and Lambert Cylindrical Equal-Area.
- **Conical:** Lambert Conformal Conic and Equidistant Conic.
- **Planar:** Azimuthal Equidistant and Lambert Azimuthal Equal-Area.

- Classification based on the preservation of characteristics

As mentioned above, each map projection technique preserves some or none of the characteristics in the resulting map. Thus, map projections can also be classified based on the characteristics they preserve, where we can distinguish several types:

- **Angular preservation:** this type of map projection is called conformal, orthomorphic, or orthogonal projection and retains the angular conformity. It is typically used when the angular relationships are the target of the analysis, such as in navigation and meteorology. Moreover, orthomorphic map projections are recommended for small areas since preserving angles in large areas is difficult. On the other hand, these map projections have the downside of deforming areas. The difference in the accuracy of the results becomes more significant as the target area increases, so they are not suitable for this type of analysis. The Mercator, Transverse Mercator, and Lambert Conformal Conic are some examples of map projections aiming to retain angular properties.
- **Distance preservation:** equidistant projections are suitable for those analyses where the purpose is to measure the distance between locations, such as in seismic mapping and navigation. Distances are maintained accurately from any place on the map to the centre of the projection. Examples of this type of map projection are the Azimuthal Equidistant, Equirectangular, and Plate Carree Equidistant Cylindrical projections.
- **Area preservation:** equal-area maps maintain the proportional relationships between the mapped areas. It is used when the area computation is required, e.g., urban planning in large areas. On the other hand, this map projection suffers from angular distortions that increase with the size of the area. Examples of this type are the Lambert, Albers, and Mollweide Equal Area projections.

To summarise, there is a wide range of possibilities regarding map projections, and all of them try to represent the Earth's surface as accurate as possible. Selecting one or the other is based on the fact that the map projection should be suitable for the type of analysis required. Unfortunately, this is not an option in many cases since data is scarce and typically only available using one technique.

4.1.1.2. Coordinate reference systems

Every point on the Earth's surface can be located by using a set of values called coordinate reference system (CRS). As a rule, CRSs are divided into two groups: geographic coordinate reference systems and projected coordinate reference systems (also known as Cartesian or Rectangular) that fall out of the scope of this work.

- Geographic coordinate systems

This type of reference system is widespread these days, where WGS84 is the most popular (Figure 4.2). They typically have two coordinate values called latitude and longitude (an additional third coordinate related to height is included

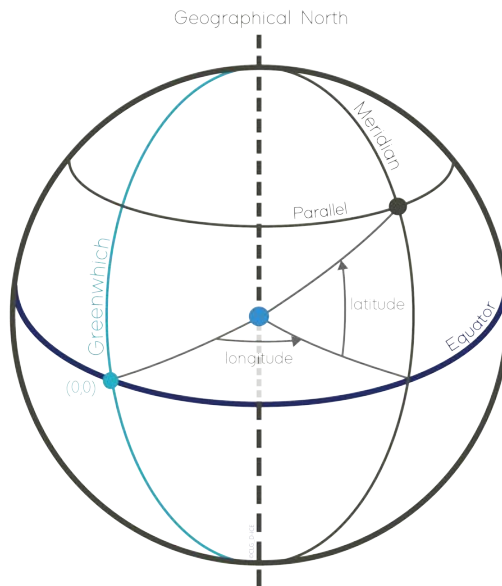


Figure 4.2: Geographic coordinate system representation using WGS84 CRS [97]. The latitude value of the reference point is measured using the corresponding parallel from the Equator. In contrast, the longitude value is obtained using the corresponding meridian from the Prime Meridian at Greenwich (England).

in exceptional cases), measured in decimal degrees ($^{\circ}$), minutes ($'$), and seconds ($''$) to increase the accuracy of measurements. Their definition follows:

- **Latitude:** the angle formed by the intersection of the plane of the Equator and the line perpendicular to the Earth's surface that passes through the target point. Lines of latitude, also known as parallels, are parallel to the Ecuador reference line, dividing the Earth into two hemispheres of 90 degrees each (one to north and another to south) with equally spaced sections of one degree of latitude. Points north of the Equator have positive latitude values, whereas points south have negative values.
- **Longitude:** the angle between the corresponding meridian that passes through the target point and the reference line called Prime Meridian through Greenwich (England). In particular, a line of longitude (also known as meridian) runs perpendicular to the Equator between the North and South Poles. It takes values from 0° to 180° east or west of the Prime Meridian, where west meridians are represented using negative values.

Nowadays, one of the most used CRS is the Universal Transverse Mercator (UTM). It is a horizontal position representation, meaning that it ignores altitude values and considers the Earth as a perfect ellipsoid divided into 60 zones of 6 degrees wide in longitude each from east to west (Figure 4.3). The UTM zone 1 is located at 180 degrees west longitude, whereas the UTM zone 60 is located at 180 degrees east longitude. In practice, the UTM zone number is also followed by the easting and northing planar coordinate pair measured in metres. Their definitions follow:

- The easting coordinate represents the distance from the central meridian (longitude) corresponding to the used UTM zone.
- The northing coordinate represents the distance from the Equator.

It has to be taken into account that the hemisphere has to be specified in some way as the UTM CRS does not consider the use of negative values. There are two known conventions, and it is best to indicate which one is being used: the zone number followed by the *N* or *S* hemisphere designator or the zone number followed by the grid zone—the latitude band designator appended to the zone number. This issue makes a designation such as 32S unclear and error-prone because it could mean *South* or the grid zone *S*, leading to unintended misunderstandings if the convention was not previously specified.

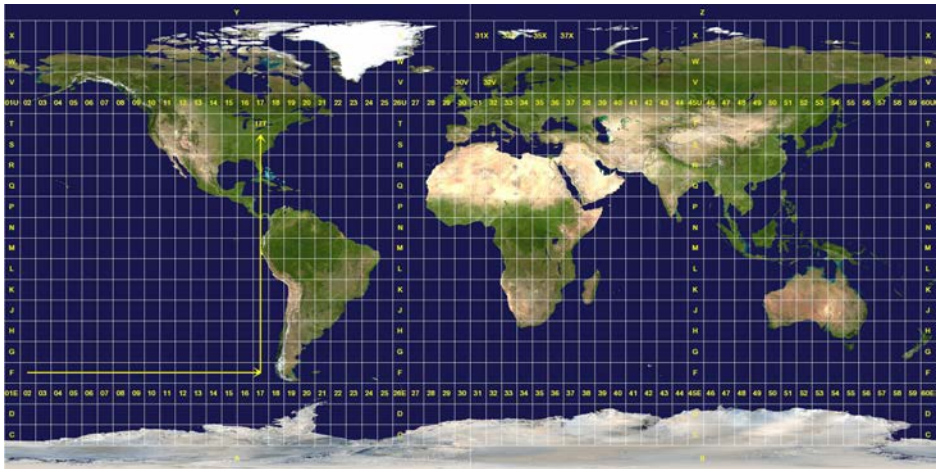


Figure 4.3: Universal Transverse Mercator CRS [98]. As an example, the UTM zone 17T, which covers the area of the city of Toronto (Ontario, Canada), is shown on the map marked with yellow arrows. The complete UTM coordinates of the city of Toronto are 17T 630380.21 mE 4834628.61 mN.

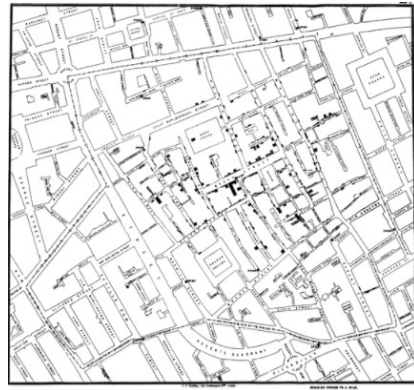
4.1.2. Geospatial analysis

The terrain surface has been widely studied for many years, being a specific case of significant importance nowadays. The first appearance of this concept dated from 1832 when Charles Picquet designed a map representing cholera outbreaks across different districts of Paris (Figure 4.4a) [99]. This map was an early version of a heat map, which would later revolutionise several industries. Inspired by Picquet, John Snow followed the same principle to represent cholera deaths in London in 1854 (Figure 4.4b) using bars located at the specified households. This map is one of the earliest analyses of geographic inquiry in epidemiology.

In the mid-19th century, a new printing technique called photozincography appeared to differentiate the many layers that can form a map [100]. Vegetation, buildings, water areas, and land could be visualised as separate layers. It was revolutionary for its time, although there is no opportunity to analyse the mapped data using this technique—as it is possible with current geographic information frameworks. Nevertheless, the photozincography technique gave rise to modern geographic information systems development.



(a) Charles Picquet's map of cholera outbreaks across Paris in 1832



(b) John Snow's map of cholera deaths in London in 1854

Figure 4.4: First geospatial analyses from the 19th century [99].

4.1.3. Geographic Information Systems

A Geographic Information System (GIS) is a framework for holding, processing, and analysing data mainly related to the science of geography. These tools allow customers to analyse spatial information, edit data in maps, and obtain the results of all requested operations, having access to a vast amount of information of a specific territory. Many types of data can be queried for a given terrain: elevation, boundaries, soils, hydrography, and population density maps, among others. In addition, maps and 3D scenes can be obtained by analysing spatial locations and organising layers of information into visualisations.

4.1.3.1. History

The concept of GIS was first introduced in the early 1960s when hardware and mapping applications were on the rise for the nuclear weapons program. The first operational GIS was launched in Ottawa (Canada) by Roger Tomlinson to store, collate, and analyse data regarding land usage in Canada. Throughout the 1970s and 1980s, researchers enhanced the system until the mid-nineties, when the software contained datasets of the entire country of Canada.

At the same time, research in spatial analysis and spatial data management started to grow at academic centres of renowned prestige, such as Harvard and Esri. In particular, Esri was one of the largest GIS software development com-

panies of the 1990s, and, at that time, it was working on a desktop solution for developing mapping systems via a Windows-based interface.

In 1999, Esri released ArcGIS (Figure 4.5), which despite starting as a simple extension for data visualisation from a previous program, it ended up as an independent and very popular GIS software worldwide offered for both Unix and Windows systems. Since this application was the first GIS software to use a graphical user interface (GUI), many governments, businesses, defence, and non-governmental organisations started to adopt the ArcGIS standard—considered to be more user-friendly. Moreover, at that time, two of the world’s most famous programs appeared: QGIS (2002) and gvSIG (2004), which have grown enormously in capacity and users until today.

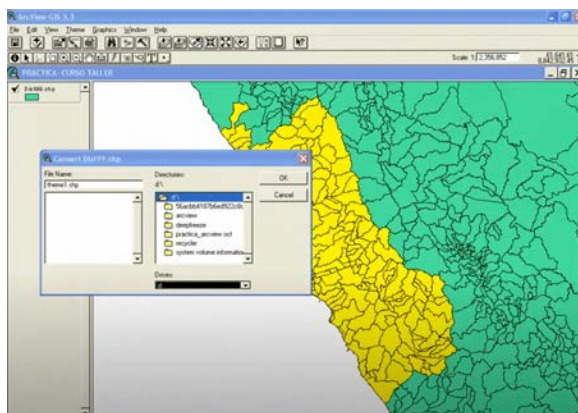


Figure 4.5: Screenshot of the ArcView extension from ArcGIS, the program considered one of the most relevant GIS desktop software in history.

During the next decade, the Internet eased the adoption of GIS software to provide services to cities, municipalities, and private organisations based on the reduction in costs. It also allowed organisations to add datasets to the maps already online, expanding the information available to users.

The first revolutionary event of the 21st century was the appearance of Google Earth, initially developed by Keyhole, Inc (founded in 2001 and acquired by Google in 2004). At first, this software only had a paid version called EarthViewer 3D (Figure 4.6a) until the year 2005, after its purchase by Google, when it became the well-known open-source software called Google Earth (Figure 4.6b). This program allowed the visualisation of Earth satellite images obtained from different sources. The map resolution has increased over the years, along with the data displayed: it can currently display barometric, astronomical, and historical data.

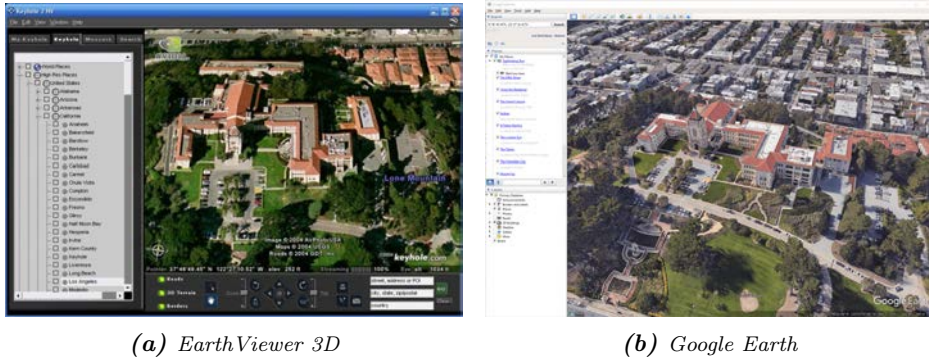


Figure 4.6: Screenshots of EarthViewer 3D and its successor Google Earth GIS software pointing to the same location in San Francisco (California, United States).

4.1.3.2. GIS applications

GIS software is increasingly providing more analytical tools used in many fields of research: slope computation; data analysis; topological, hydrological, and cartographic modelling; map overlay; and visibility computation. This last is a widespread tool provided by these programs, which is of great interest in telecommunications, environmental planning, ecology, tourism, and archaeology [101, 13, 102].

In these diverse fields, knowing the visibility in terrain is almost a requirement to achieve optimal results. However, we must first introduce every possible way of modelling the terrain (e.g., DEM and TIN) before explaining the key aspects of the visibility computation.

4.1.4. Digital Elevation Models

A Digital Elevation Model (DEM) is a digital representation of a terrain's surface based on elevation data, usually referencing the surface of the earth. Before the inclusion of the DEM structures, country-wide elevation data was stored using contour lines in paper maps. This last is still a valid method in topography, but from the perspective of data storage, it has the following deficiencies:

- As contour maps are non-continuous representations of the terrain, the surface forms between the selected contour intervals are unknown.

- The generation of contours depends on visualisation, which enforces cartographic generalisation rules. It removes some details of the topography while overemphasising other forms.

Fortunately, new data structures have appeared to represent the terrain, solving the aforementioned drawbacks.

4.1.4.1. Types

Nowadays, DEM data structures fall into two main groups (Figure 4.7) [103]: raster-based and vector-based. This classification distinguishes whether the elevation data is stored using a regular grid (also known as heightmap) or a triangulated irregular network (TIN), respectively. The raster-based approach uses a matrix of cells organised into rows and columns (grid), where each cell contains the corresponding elevation value. The TIN representation is a vector-based approach for the land surface that uses irregularly distributed nodes and lines arranged in a network of non-overlapping triangles.

Both structures are similar as they include three-dimensional data (x , y , and z) of the shape of the surface without considering vegetation and other structures. However, they differ in how each grid of nodes is defined and in the nature of the data involved. Raster-based DEMs suffer from data redundancy in areas of

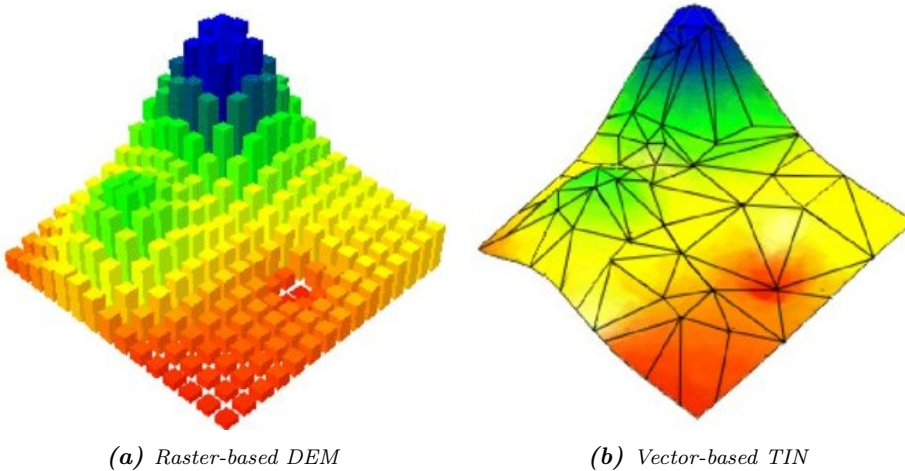


Figure 4.7: Main types of Digital Elevation Models (DEMs) according to the data structure used [104].

uniform terrain, inability to adapt the grid to areas of differing relief complexity, and excessive emphasis along the axis of the grid. In contrast, vector-based TINs have little data redundancy, which means that extra data is included in complex areas whereas less data in non-complex areas. The disadvantages include large datasets, minor changes in the data size leading to changes in data resolution, and complex coordinate transformations.

In both cases, the accuracy of the data depends on the resolution, i.e., the distance between sample points. Other factors affecting this variable are data type resolution (integer or floating point), and the actual sampling of the surface.

4.1.4.2. DEM generation and applications

The generation of DEMs plays a major role in topography research [105]. They can be derived through different techniques such as digitising contours from existing topographic maps, topographic levelling, electronic distance measurement, differential GPS measurements, digital photogrammetry, and Radar remote sensing and Light Detection and Ranging (LiDAR). Nowadays, we can use a wide range of data sources to generate DEMs, whose selection depends on the data availability for the target area, the cost, and the requirements.

DEMs are very significant geospatial datasets because of the wide variety of fields in which they are used: ortho-rectification of aerial photographs; cartographic representations; geophysical, biogeographical, hydrological, and hydraulic analyses and models; water management; landscape dynamics and climate impact studies; geological, agriculture, and forestry applications; visibility analyses; road and telecommunication networks planning; flood risk analysis; among many other uses and applications.

4.1.5. Visibility

The term visibility has several meanings, where the one that most relates to the field of geospatial analysis reads: *degree to which objects that are far away can be seen outside, as influenced by weather condition*. There are many factors to consider when visibility is the variable of the study. However, the main factor that affects the visibility is the planet's relief, which in the particular case of the Earth usually includes a wide range of landforms such as mountains, valleys, ravines, rivers, and plateaus.

Other factors can affect the capacity for visibility, e.g., weather conditions in the territory—scenarios with pollution, fog, rain, or snow can make the object less

visible from the observer's position. Additionally, visibility situations in daylight show better results than at night when specific equipment is required.

Between the observer's location, which is usually called *Point of View* (POV), and the target location, other elements can be found between these two points blocking the line of sight. Whether natural or attributable to human activities, these elements include trees, buildings, bridges, and towers, among others. The points behind these structures are nearly impossible to be visible from the POV. All the factors above increase the complexity of the visibility problem.

In the field of computational research, the definition of the term *visibility* given above can be slightly changed to ease its implementation. For example, we can specify the visibility between an observer and a target object separated by a certain distance using a Boolean variable, which takes 1 when the object is visible to the observer and 0 otherwise. This value set to 1 means that no object stands between the target object and the observer. Likewise, this definition can be extended to the case of having many target objects, as is the case in a common DEM of millions of points. However, first, the simplest example must be expressed mathematically, which is related to the calculation of the visibility between two points:

Let A and B be two locations with $(x_a, h(x_a))$ and $(x_b, h(x_b))$ coordinates in a two-dimensional terrain $T(x, h(x))$, where $h(x)$ is the function that returns the height of every point in T . For the sake of simplicity, the heights of both target points are $h(x_a) = h_a$ and $h(x_b) = h_b$. In this case, B (the target point) is visible from A (POV) and vice versa if the height of every intermediate point is below the straight line that joins them, i.e., there is no obstacle blocking the line of sight. This condition can be expressed as:

$$h(x) < \frac{h_b - h_a}{x_b} \cdot x + h_a \quad \forall x \in (A, B) \quad (4.1)$$

which can be simplified as follows:

$$h(x) < y(x) \quad \forall x \in (A, B) \quad (4.2)$$

where $y(x)$ is the function that returns the height value of the imaginary straight line that joins A and B according to the x variable. This mathematical expression is represented in Figure 4.8 where the height functions $h_1(x)$ and $h_2(x)$ represent two models of terrain with different height values between the locations A and B . The first height function $h_1(x)$ blocks the line of sight $y(x)$ on the section whose height exceeds that of the line of sight, preventing the A and B points from being visible to each other. Conversely, this fact does not apply when considering the second height function $h_2(x)$.

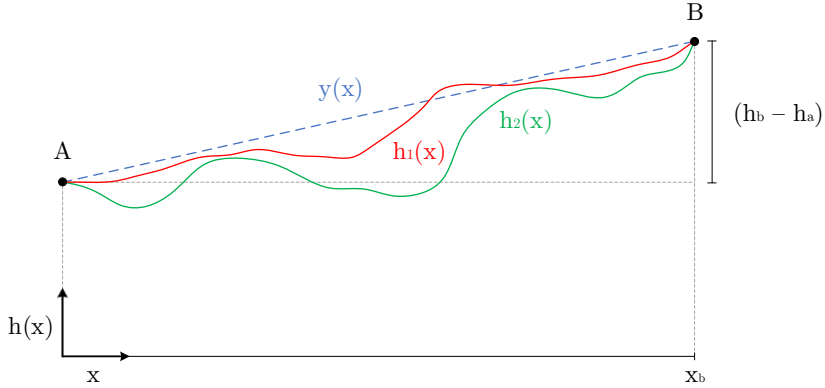


Figure 4.8: First example of the visibility problem between two locations: A (POV at ground level) and B (target point). Two height functions $h_1(x)$ and $h_2(x)$ are presented for the terrain in red and green, respectively. The line of sight $y(x)$ (represented in blue) is blocked when considering the first height function $h_1(x)$ for the terrain, whereas this is not the case with the second height function $h_2(x)$.

In the above case, the observers are considered lying on the ground at A and B points, which is not the typical approach. Figure 4.9 shows a similar example but considering the observer at A with a height above the ground of h (measured from the eyes to the plane of the ground). The mathematical expression is as follows:

$$h(x) < \frac{h_b - h_a}{x_b} \cdot x + h_a + h \quad \forall x \in (A, B) \quad (4.3)$$

This modification highly affects the visibility calculation as the line of sight is not blocked for any of the height functions, with A and B visible. This particular example can be extrapolated to other cases where watchtowers or drones are used for monitoring purposes. We can overcome possible obstacles by increasing the observer's height, which increases the maximum visual distance.

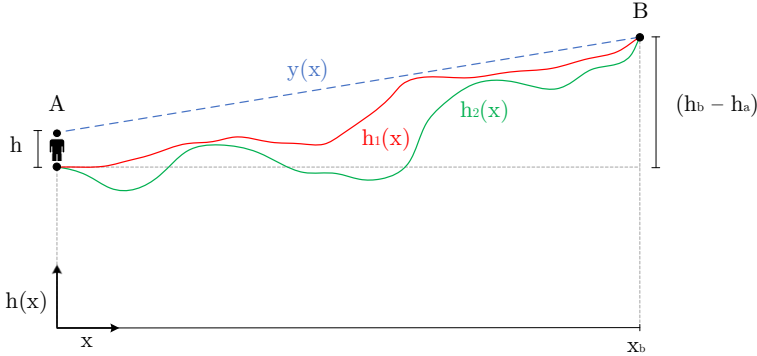


Figure 4.9: Second example of the visibility problem between two locations: A (POV at height h) and B (target point). In this case, it is considered that the observer located at A has a height different from ground level. The same height functions $h_1(x)$ and $h_2(x)$ from Figure 4.8 are shown for the terrain. The line of sight $y(x)$ (represented in blue) is free of obstacles; hence point B is visible from A and vice versa.

In addition to the observer's height, another vital consideration is the observer's maximum range of vision. On a flat surface, the maximum observable distance will always be limited by the horizon, taking this as the imaginary line where the sky seems to join either the land or sea. In that case, the distance is approximately 5 km due to the Earth's curvature assuming favourable weather conditions. However, this statement does not apply when observing non-uniform terrain from an elevated position, where the maximum range of vision is reduced.

Moreover, every observer does not have to present the same visual range as it depends on the individual ability. This condition is relevant since one observer could see another, but the opposite may not be the case. A representative example could be when two observers are far apart, and one of them has a pair of binoculars, giving them an advantage. This fact leaves space for using instruments specifically designed for this purpose, rather than human observers with limited capacities.

Once we have addressed the visibility concept for two points, we can move on to the case of having a single observer in the terrain to analyse the surrounding points. In this case, we seek to obtain (i) the visible area from the observer's location or (ii) the area from which the observer is seen. The results obtained in both cases will be similar if we consider homogeneous visibility conditions throughout the terrain.

4.1.6. Viewshed

Many problems of terrain surface analysis require the evaluation of the data around a reference point. It is the case of the calculation of the viewshed (VS), which represents the geographical area that is visible from a POV in a given DEM. This computation is included in some visibility modules from commonly used GIS software such as ArcGIS [106], QGIS [107], and Google Earth. The importance of this calculation stems from the fact that knowing visibility is almost a requirement in many fields of study such as surveillance, telecommunications, and environmental planning.

The viewshed computation has been thoroughly studied in the recent literature [102, 108], being usually addressed using rotational plane sweep-based algorithms, or only rotational sweep [109]. In this method, a line is traced from the POV, which works as a vertex in the plane. This line rotates by 2π radians, and all the points crossed by that line are analysed with respect to the vertex.

Another related approach involves the discretisation of the plane in azimuthal sectors radiating from the reference point [110] (Figure 4.10). This division is carried out by splitting the area that surrounds the POV into ns azimuthal sectors (commonly 360 sectors of 1° degree each). An axis represents each azimuthal sector at its centre, and points close to the axis are compared with the reference location. Here, the statistical representativeness of every axis progressively decreases as we move away from the vertex since the width of the sector increases

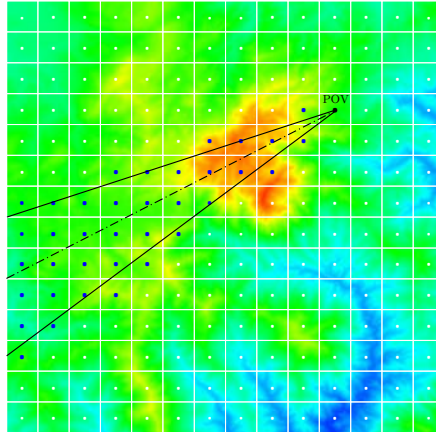


Figure 4.10: Example of the azimuthal discretization of a given DEM. For the sake of clarity, only one azimuth is shown.

linearly with the radius. However, the required accuracy is also reduced to the same extent in most cases. This issue is exploited in some situations where the reference location works as the transmitter or receiver of specific signals whose strength decreases with the square of the distance, such as radio signals, sound waves, and line of sight.

The literature distinguishes three types of viewshed problems according to the number of observers considered in the computation: (i) singular, (ii) multiple, and (iii) total viewsheds.

4.1.6.1. Singular and multiple viewshed

The singular viewshed is the simplest visibility problem: it involves computing the viewshed from a single observer at a specific elevation. As a starting point, most viewshed computation algorithms perform the azimuthal partition of the area. Then, different lines of sight (LOS), which correspond to the axis of each sector, start from the POV and are radially distributed towards the most distant areas (Figure 4.11a). Every target point crossed over by this line is sequentially

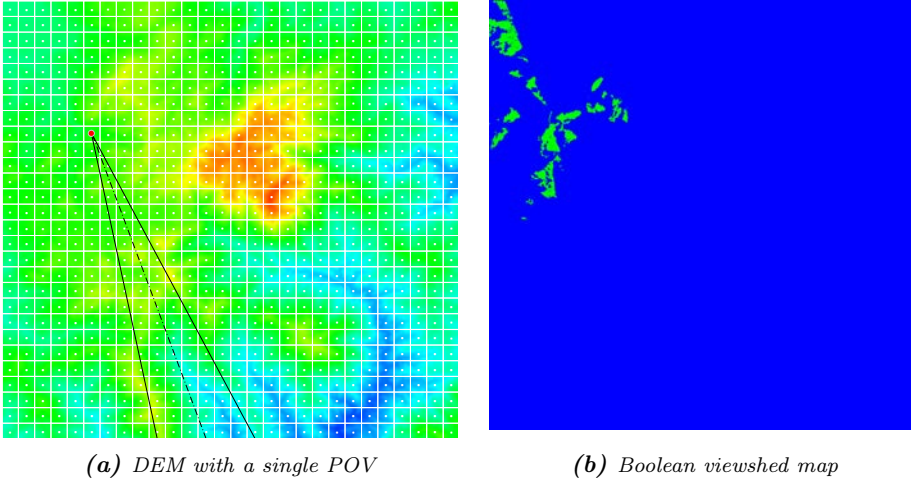


Figure 4.11: Illustration of the singular viewshed problem. (a) presents a discretized DEM with a single POV where only one sector is shown for the sake of clarity, although all sectors are considered in practice. Low elevation values are shown in blue, whereas red cells represent maximum values. (b) shows the Boolean viewshed map result where the areas visible from the POV are shown in green.

compared to the POV based on the elevation values to compute its visibility. Finally, the result is a Boolean raster map containing visible and non-visible points from the POV in the terrain (Figure 4.11b).

Algorithm 4.1 presents an example of the general approach for the calculation of the singular viewshed using a regular Cartesian grid, taking a single POV and DEM as inputs. The initial coordinates of the observer's location are (i_0, j_0, h_0) , where the last coordinate is the height of the observer measured from the eyes to the plane of the ground. The coordinates of the *POV* structure used for the calculations are (i, j, h) , where the last initially corresponds to the elevation of the point in the terrain. *VS* is the variable that will accumulate the viewshed value, *axis* is the set of points included in a particular sector (*s*), and *selectAxisPointSet* adds candidate points within the sector, which will be used to compute the viewshed.

If we consider that all points are aligned, then the viewshed for each sector can be analysed using the *linear_viewshed* function (Algorithm 4.2). It computes the visible area in the input sector with respect to the POV located on the axis of that sector. The visibility of each point within the linearised set of points *axis* is checked following the direction from the nearest point to the furthest.¹

Algorithm 4.1 General approach for solving the singular viewshed problem.

```

1: function SINGULAR_VIEWSHED(DEM, i0, j0, h0)
2:   point POVi,j,h = DEM[i0][j0]                                ▷ observer's location
3:   POVh += h0                                                    ▷ add observer's height
4:   float VS = 0                                                    ▷ accumulated viewshed
5:   for s = 0, ns do                                                ▷ sector loop
6:     pointSet axis = selectAxisPointSet(DEM, POV, s)
7:     VS += LINEAR_VIEWSHED(POV, axis, true)                      ▷ forward direction
8:     VS += LINEAR_VIEWSHED(POV, axis, false)                    ▷ backward direction
9:   end for
10:  VS *= ( $\pi/ns$ )                                                    ▷ Papus theorem scaling
11: end function

```

¹Note that opposing sectors are shown interlinked in the sector loop because many viewshed algorithms exploit the alignment of the corresponding data.

Algorithm 4.2 Function that computes the viewshed given the set of points within the structure axis.

```

1: function LINEAR_VIEWSHED(POV, axis, forward)
2:   global bool visible = true                                ▷ visibility status
3:   global float maxθ =  $-\infty$                                 ▷ max. angle
4:   global pointSet visibleSet = {}                            ▷ set of visible points
5:   do                                                            ▷ point loop
6:     point T = axis.next()                                    ▷ current point
7:     POINT_VIEWSHED(POV, T)
8:   while T != axis.last()
9:   return visibleSet.measure()                                ▷ summation of visible areas
10: end function

```

A target point T is visible from the POV if its angular altitude θ is higher than all the previous ones considered in the *axis* ($max\theta$), as shown in Algorithm 4.3. The visible points from the *axis* are included in a set of points called *visibleSet*. In order to improve efficiency, only the starting and ending points of a segment are measured (*visibleSet.add*) and considered in processing. *startRS* and *endRS* indicate whether a sequence of visible points has been found. First, the distance between the POV and the first point found belonging to a visible section is stored in $dist_0$. Afterwards, when the final visible point of this visible segment is found, its distance with respect to the POV is measured and stored in $dist$. This process is repeated until all points on the axis are analysed, as shown in the side view in Figure 4.12. The projection of all visible segments throughout the sector results in the generation of visible sections, commonly known as ring-sectors. The area of each visible section (A_{vs}), considering sectors of one degree of opening, is computed as follows:

$$A_{vs} = (\pi/360) \cdot (R^2 - r^2)$$

where R and r are the radius of the visible ring-sector related to the *endRS* and *startRS* values, respectively, with respect to a particular POV .

Considering all of the above, the viewshed for a location is the summation of the areas of every visible section (*visibleSet.measure*). This approach can be seen as a vectorisation of the viewshed raster model, which reduces memory accesses and mathematical calculations as proved in [110] and can be combined with other methods.

Algorithm 4.3 Function that computes the visibility of an input point T . Figure 4.12 shows the procedure followed.

```

1: function POINT_VIEWSHED( $POV, T$ )
2:   float  $dist = \sqrt{(T_j - POV_j)^2 + (T_i - POV_i)^2}$            ▷ current distance
3:   float  $\theta = (POV_h - T_h) / dist$                            ▷ angular difference
4:   bool  $prevVisible = visible$ 
5:   if ( $\theta > max\theta$ ) then
6:      $visible = true$                                            ▷ current point visible
7:   else
8:      $visible = false$ 
9:   end if
10:  bool  $startRS = !prevVisible \& visible$ 
11:  if ( $startRS$ ) then                                           ▷ ring-sector begins
12:     $dist_0 = dist$                                              ▷ initial distance
13:  end if
14:  bool  $endRS = prevVisible \& !visible$ 
15:  if ( $endRS$ ) then                                           ▷ ring-sector ends
16:     $visibleSet.add(dist_0, dist)$                              ▷ difference of distances
17:  end if
18: end function

```

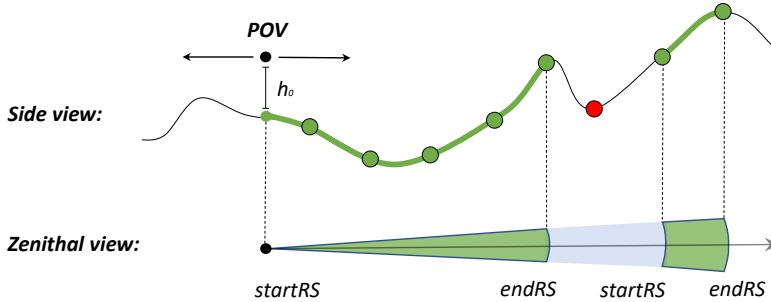


Figure 4.12: Side and zenithal views for a single POV , with a specific height h_0 , from which two segments are visible (represented both by thick green segments). The corresponding visible ring-sectors are obtained for each one considering their starting points ($startRS$) and ending points ($endRS$).

A lot of research effort has been invested in reducing the number of operations required to compute the singular viewshed problem, hence reducing the complexity to the same extent. This computational complexity is initially $O(N^2)$ using a non-optimised approach and considering an input DEM containing N points. For example, the complexity to obtain the viewshed using point-to-point algorithms such as R3 is $O(N^{3/2})$, whereas it is reduced up to $O(s \cdot N^{1/2})$ using rotational sweep.

Furthermore, the number of target points to analyse from a POV in the singular viewshed problem is significantly reduced from N to $s \cdot N^{1/2}$ by implementing rotational sweep based algorithms [109]. Considering that a typical DEM exceeds several millions of points and the discretisation of the sector is rarely above the required accuracy, the accomplished reduction using this type of algorithm is between one and three orders of magnitude [111].

Likewise, the viewshed problem for several POVs can be raised (Figure 4.13a) and will be addressed in depth in Chapter 5. This problem is usually tackled by repeating the procedure applied to the singular viewshed problem for each POV and combining the viewshed results (Figure 4.13b).

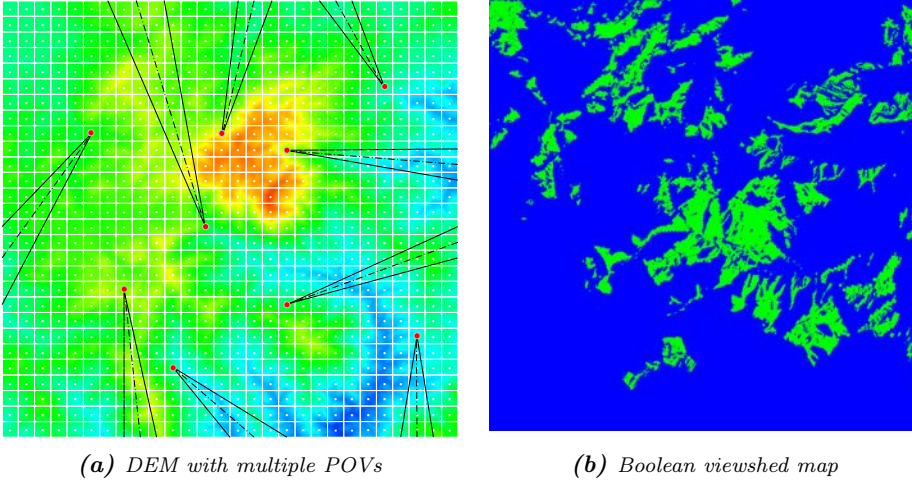


Figure 4.13: Illustration of the multiple viewshed problem. (a) presents a discretized DEM with multiple POVs where only one sector per POV is shown for the sake of clarity, although all sectors are considered in practice. Low elevation values are shown in blue, whereas red cells represent maximum values. (b) shows the Boolean viewshed map result where the areas visible from the POV are shown in green.

However, taking into account that the viewshed computation for a single POV was already very time-consuming, the inclusion of new POVs has an adverse effect on performance as the number of required operations increases. This issue makes parallelism and supercomputing highly recommended for these sorts of approaches and even more so when the aim is to compute the viewshed for each point in the terrain, in a process commonly known as *total viewshed computation*.

4.1.6.2. Total viewshed

A complete visibility study in a particular area involves knowing the viewshed from each point and every direction. This problem, known as total viewshed, is one of the most challenging visibility calculations due to its high complexity and computational cost. It involves obtaining the viewshed from each point in the DEM as a POV (Figure 4.14a), and then accumulating their visibility results in a new ‘heat’ map where each cell contains the viewshed value of the corresponding location measured, e.g., in km^2 (Figure 4.14b).

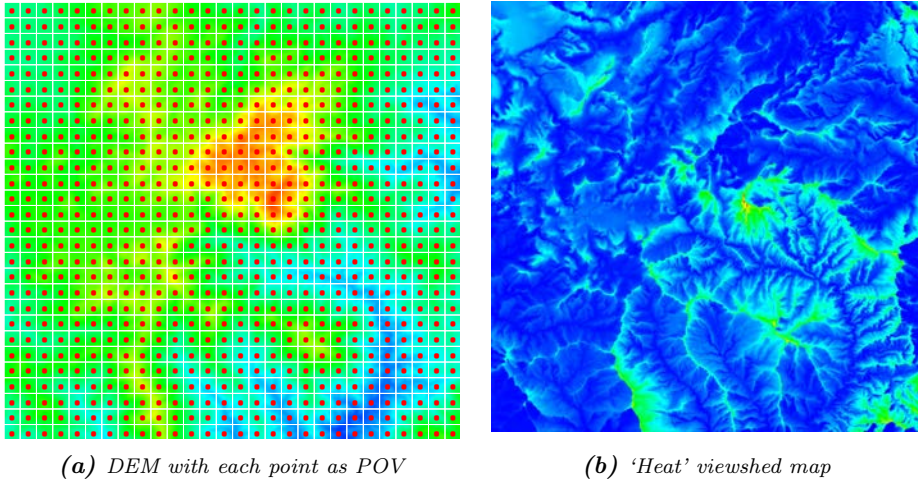


Figure 4.14: Illustration of the total viewshed problem. (a) presents a discretized DEM with each point as POV where no sector per POV is drawn for the sake of clarity, although all sectors are considered in practice for each POV. Minimum elevation values are shown in blue, whereas red cells represent maximum values. (b) shows the ‘heat’ viewshed map result, which uses a colour scale showing the points with greater visibility in red and those with lower visibility in blue.

The total viewshed problem was almost impossible to tackle not long ago because (i) a typical DEM greatly exceeds several millions of points and (ii) the singular viewshed computation is computationally demanding; hence repeating this procedure N times would have been incredibly time-consuming on CPU. The inherent complexity of the total viewshed problem is up to $O(N^3)$ if a non-optimised approach is applied N times over a problem of $O(N^2)$ complexity. However, using rotational sweep, the complexity can be reduced up to $O(s \cdot N^{3/2})$.

Algorithm 4.4 shows the steps required to address the total viewshed problem, considering a DEM represented by a Cartesian grid with $dimy \times dimx$ points. The viewshed value, i.e., the visible terrain area for each point in the DEM is stored in the total viewshed matrix (T_{VS}). This matrix has the same dimensions as the DEM, and each cell contains a specific viewshed value obtained after performing the corresponding singular viewshed computation (Algorithm 4.1). Some authors have observed that swapping the loops of Algorithm 4.4 and Algorithm 4.1 can significantly improve data locality in memory [111, 110]. This operation is one of the pillars of the algorithm presented in this chapter.

Algorithm 4.4 *General approach for solving the total viewshed problem.*

```

1: function TOTAL_VIEWSHED( $DEM, h_0$ )
2:   for  $i = 0, dimy$  do
3:     for  $j = 0, dimx$  do
4:        $T_{VS}[i][j] = \text{SINGULAR\_VIEWSHED}(DEM, i, j, h_0)$ 
5:     end for
6:   end for
7: end function

```

Nowadays, most GIS software packages include specific modules for singular viewshed computation; few provide multiple viewshed calculations and, when they do, they use task queues showing poor computational performance—the singular viewshed computation is repeated for each of the considered POVs. None of these can solve the total viewshed problem, even though this information could be of value to address well-known problems such as siting multiple observers [112] and path planning with surveillance aims [113]. The first problem is related to finding the fewest possible number of POVs providing maximum viewshed for a specific area. The second involves designing a near-optimal path aiming to achieve maximum terrain coverage. Both problems would be substantially simplified if the viewshed from each point in the area is known beforehand [114].

4.1.7. Summary of the main concepts

Since the next section presents our proposal to accelerate the visibility computation considering the total viewshed problem, all previous concepts must be clear before going into detail. To summarise:

- **POV**: stands for Point of View. It is the reference point where the observer is located and from which the viewshed is calculated.
- **Viewshed**: area of the terrain visible from a single POV (the simplest case) and involves a set of points visible from it.
- **Visibility**: status of a target point that indicates whether that point is seen from another point.

On the other hand, the different viewshed problems described so far can be summarised as follows (Figure 4.15):

- **Singular viewshed**: takes a single point in the DEM as POV. The result is a Boolean viewshed map of visible and non-visible points.
- **Multiple viewshed**: takes a few points in the DEM as POVs. The result is an accumulated Boolean viewshed map of visible and non-visible points.
- **Total viewshed**: takes each point in the DEM as POV. The result is a 'heat' viewshed map where each cell contains the measured viewshed value of the corresponding location in the terrain.

As previously mentioned, the total viewshed computation is the most computational demanding problem among visibility-related ones. This is the main reason why it has been considered for this chapter.

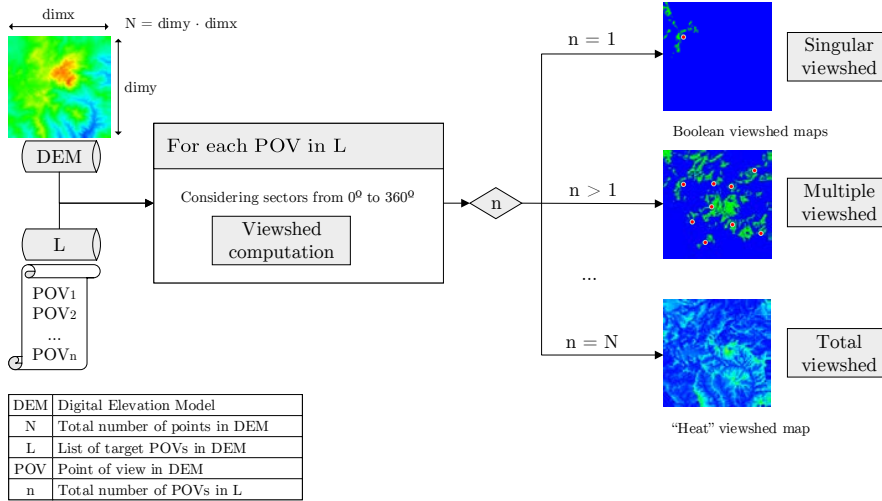


Figure 4.15: Diagram showing the different viewshed problems: singular, multiple, and total viewsheds. This classification is based on the number of target points of view (POVs) chosen for the DEM, each of which produces different outputs.

4.2. Related work

The most relevant problems of visibility analysis, with a particular focus on the viewshed computation, fall into two large groups related to (i) singular and multiple viewsheds and (ii) total viewshed.

4.2.1. Singular and multiple viewsheds

The viewshed analysis problem has been widely studied for many years given the mass of interpolation computations required to produce precise results [115, 116, 117, 114, 118]. Authors usually use line-of-sight-based algorithms such as R3, R2, or DDA [119, 120]. These methods project rays starting from the observer towards the boundary of the DEM to obtain the points included in processing. Another related strategy is XDraw [119], which computes the LOS function in stages arranged as concentric squares centred on the position of the observer.

Many algorithms calculate the viewshed from a single POV, or a small number of POVs at best. In [121] a singular viewshed implementation was developed for built-in GPU systems based on the LOS method and texture memory with

bilinear interpolation. They achieve a speed-up up to 70x with respect to the sequential CPU implementation. The GPU implementation proposed in [122] achieves remarkable results in obtaining a Boolean raster map instead of a map containing viewshed values. A novel reconfiguration of the XDraw algorithm for GPU context is described in [101], which outperforms CPU and GPU implementations of well-known viewshed analysis algorithms such as R3, R2 and XDraw. Furthermore, an efficient implementation of the R2 viewshed algorithm is carried out in [123] with a particular focus on input/output efficiency and obtaining significant results in contrast to the R3 and R2 sequential CPU implementations. The algorithm described in [124] focuses on a two-level spatial domain decomposition method to speed-up data transfers, performing better than other well-known sequential algorithms. Other extended approaches are focused on obtaining the viewshed for multiple points [125, 126]. More recent research is presented in [102] where fast candidate viewpoints are obtained for multiple viewshed planning. These authors have also conducted a parallel XDraw analysis [127, 108] to improve the results obtained by previous XDraw algorithms.

4.2.2. Total viewshed

Few studies addressed the total viewshed problem, and most of these focused on a simplified version, e.g., the total viewshed in [128] is obtained by drastically reducing the number of points to be processed. Likewise, the approach used in [129] computes the visibility of small areas and not for specific points.

So far, the only algorithm that addressed the total viewshed problem using high resolution DEMs is the TVS algorithm proposed in [130, 110]. It considers the closest points to the line of sight as a sample set of points stored in a structure called *Band of Sight* (BoS). In this approach, the distance to the axis determines the number of points in the BoS [118]. Maximum memory utilisation was achieved by reusing the points contained in the list and obtaining the viewshed for every aligned point in the particular sector. However, TVS has important limitations:

- For a given POV, the analysis of the points inside the BoS is performed sequentially because it is impossible to know whether a target point is visible without knowing the state of the previous one.
- The implementation of the data reuse of the BoS produces a significant overhead caused by the selection of the points included for each direction.
- It is not appropriate for high-throughput systems, such as GPUs and many-core architectures, because parallelism is limited to the sector level.

4.3. sDEM: skewed Digital Elevation Model

This section describes the proposed methodology called *skewed Digital Elevation Model* (sDEM) designed to improve data locality in memory for terrain surface analysis using the total viewshed computation problem as a case study. This approach takes into account the technical features of the CPU (host) and GPU (device) processing units to take full advantage of the intrinsic parallelism of the total viewshed computation. For the sake of simplicity, in the rest of this chapter, the input DEM is called DEM and the proposed modification is skwDEM.

First, the critical considerations of the data reuse method of sDEM are described in this section. It is followed by the data relocation method used in the same methodology. Then, the different steps required by sDEM are described. Lastly, the different parts of the sDEM implementation for multi-GPU systems are presented, which are related to (i) Kernel-1 for the data restructuring, (ii) Kernel-2 for computing partial viewshed results, (iii) Kernel-3 for obtaining the final viewshed, and (iv) the host function managing the scheduling process.

4.3.1. Data reuse

Data reuse is key to optimising the total viewshed computation, so first, we will introduce the structure that will manage this process. This structure is also called *Band of Sight* (BoS) and serves as the basis for the process of restructuring the DEM for each POV and sector. In this case, the BoS is used to find the closest points to the line of sight for any reference POV and given sector. Thus, choosing the right width for this structure is vital to improve data locality in accessing the memory.

Figures 4.16a and 4.16b show two BoS widths of $2.5\sqrt{N}$ and \sqrt{N} —considering the sector $s = 45^\circ$ for the sake of simplicity. The extensive statistical study conducted in [110] proves that the width of this structure is not a determining factor, as long as it is of the order of \sqrt{N} . Therefore, sDEM uses this BoS width to address the data repetition problem.

4.3.2. Data relocation

Once the BoS width has been set, the complete data relocation is performed to transform the DEM (Figure 4.16b) into the *skwDEM* (Figure 4.16c). The latter is a new DEM skewed in shape as a function of the BoS width. The use of this

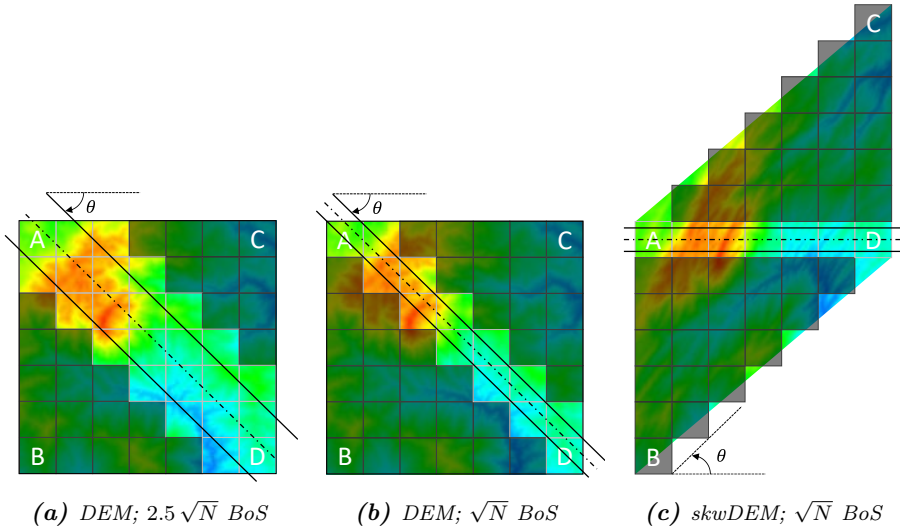


Figure 4.16: Three examples of Band of Sight (BoS) structures on the plane, considering a sector $s = 45^\circ$ for simplicity. The dark grid cells are not considered in the visibility computation. (a) and (b) show two different BoS widths of $2.5\sqrt{N}$ and \sqrt{N} , respectively, with the same layout on the $\dim y \times \dim x$ DEM, whereas (c) presents the restructuring of the $2 \cdot \dim y \times \dim x$ skwDEM considering a BoS width of \sqrt{N} . For the sake of clarity, A-D labels are placed in the corner points of the DEM so that the restructuring approach can be easily visualised. Note that only one BoS is shown.

structure enables the exploitation of the existing parallelism without adversely affecting the accuracy of the results based on the following:

- The Stewart sweep method [111] is applied, which states that an outer loop iterates over the sectors and an inner loop over the points in the DEM. It is the only model that guarantees the reuse of data aligned in each direction.
- Given a sector, all the possible parallel bands of sight that cross the DEM are built simultaneously. We apply the interpolation method based on a simplified version of Bresenham's line algorithm [131], which is commonly used for line rasterisation. This algorithm was chosen for its high speed while maintaining sufficient fidelity to the problem under consideration.
- For each sector, we apply the relocation method to the entire DEM only once. For example, in the particular case of considering 180 sectors, the data relocation takes place 180 times and always before starting the viewshed computation. Thus, the relocation only depends on the chosen sector.

Another advantage is that this method is particularly appropriate for processing on the GPU: conditional structures are reduced to the maximum, avoiding the well-known thread divergence penalty.

Regarding the relocation method, we initially considered three different possible layouts of rows and columns to build the *skwDEM* structure, as shown in Figure 4.17. In practice, the elevation values of the same latitudes are stored contiguously in memory in the DEM (Figure 4.17a); that is, the outer loop iterates from north to south, whereas the inner loop iterates from west to east. Using the interpolation method, all parallel segments from the DEM (Figure 4.17a)

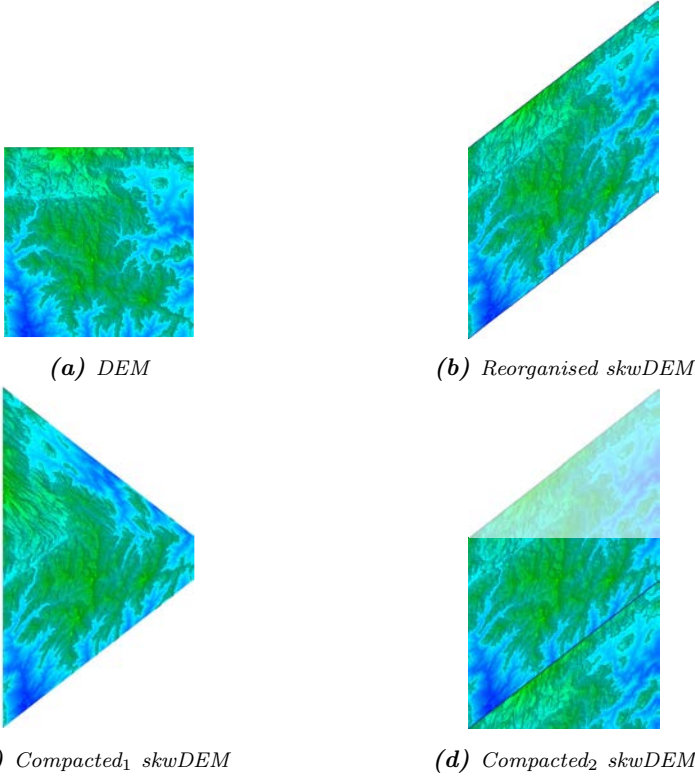


Figure 4.17: The DEM and three possible results from the data relocation process considering sector $s = 45^\circ$, for the sake of simplicity. The DEM corresponds to the area of the Montes de Malaga Natural Park (Malaga, Spain). (a) presents the input DEM, (b) shows the *skwDEM* used in this work, and (c) and (d) introduce two possible ways of compacting the data.

are projected into the *skwDEM* structure (Figure 4.17b) so that the number of non-null elements of both structures matches. Unlike the original, all the points crossed by a given sector are placed in the same row in this reorganised dataset. Therefore, memory accesses are sequentially performed, increasing locality and performance.

The reorganised matrix shown in Figure 4.17b could later be compacted by aligning all data to the left of the structure (Figure 4.17c) or relocating the data within the upper light colour triangle to the lower right area of the structure, thus forming a $dimy \times dimx$ square structure (Figure 4.17d). The second method aims to compact the information further to make memory access as regular as possible at the cost of increasing complexity and, therefore, building time. Although both approaches seem to fit better for GPU processing, they have not revealed significant differences in practice. Therefore, only the simplest and fastest approach shown in Figure 4.17b is used to build the *skwDEM* structure in every implementation of sDEM.

4.3.3. Proposed methodology

Given the specifics of the main structures related to BoS and *skwDEM*, the sDEM methodology can be divided into the following steps (Figure 4.18):

1. For each sector $s \in [0, ns/2]$, do:
 - a) Create the *skwDEM* ($2 \cdot dimy \times dimx$), which is unique for each sector, from the *DEM* ($dimy \times dimx$) and s .
 - b) Calculate the horizontal (A, D) and vertical (B, C) limits of the *skwDEM* structure, which depend on the sector s .
 - c) Let $POV_{i,j}$ be the chosen point of view with i and j coordinates. For each point $POV_{i,j} \in skwDEM$ with $i \in [A, D]$ and $j \in [B, C]$, do:
 - 1) Compute the linear viewshed considering the sectors s and $s + 180^\circ$, i.e., analyse to the right and left the points in the row to which $POV_{i,j}$ belongs in the *skwDEM*.
 - 2) Accumulate the viewshed result in a new structure called *skwVS*, which is similar in shape to the *skwDEM* structure.
 - d) Transform *skwVS* into *VS* (viewshed in the DEM) by undoing the operations performed in Step 1a. This procedure includes Pappus's theorem and also corrects the deformation introduced by the *skwDEM*.
 - e) Accumulate the results in *VS*.

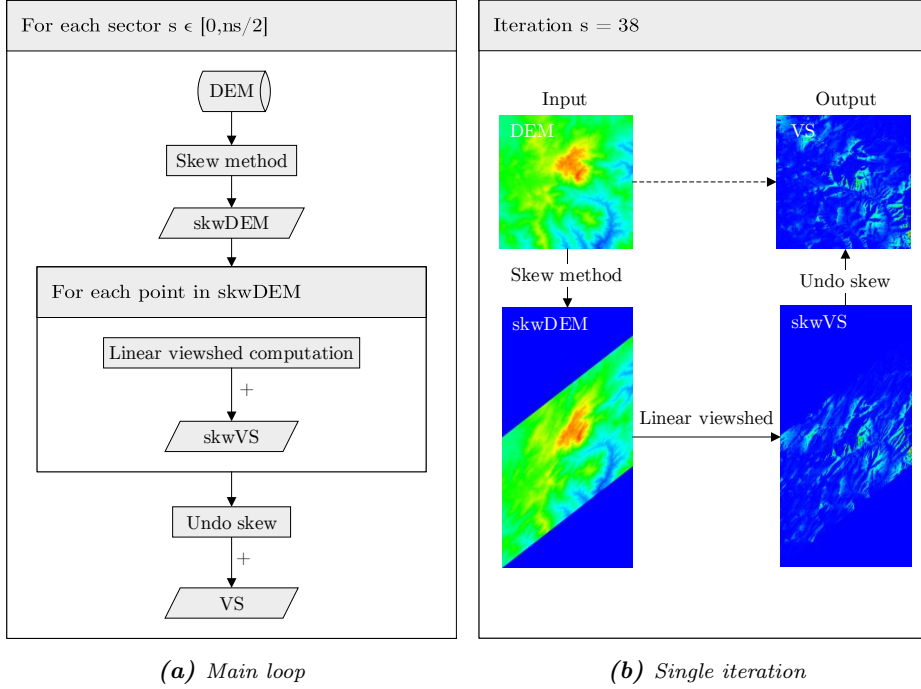


Figure 4.18: Flowchart of *sDEM* for total viewshed computation. (a) shows the steps inside the main loop, which runs through all the sectors, and (b) presents the outputs obtained in a single iteration of the same loop. In the latter, null and low values are shown in blue, whereas red cells represent maximum values. The mathematical symbol (+) represents the accumulation process.

Computing the viewshed for each sector is an embarrassingly parallel task, as each process is independent. This fact reduces the total viewshed problem to the analysis of $ns/2$ sectors considering every point in the *skwDEM*.

By skewing before computing the viewshed for a given sector, *sDEM* ensures that each BoS—needed by each point as a POV—has been previously built and included in the *skwDEM*. Each row of the *skwDEM* corresponds to the static BoS of each point included in it for a given sector; that is, if a point is a POV, the rest of the points in its row will form the BoS for that point and sector. As the *skwDEM* depends on the sector, it has to be reconstructed only $ns/2$ times. In contrast, the BoS structure used in [110] must be reconstructed for each point and sector, which corresponds to $N \cdot ns$ times.

4.3.4. Multi-GPU implementation

The single-core and multi-core implementations achieved excellent results in the different scenarios, as shown below in Section 4.4. Nevertheless, we believed that the solution to the embarrassingly parallel problem at hand could be found in significantly less time if sDEM could run on single or multi-GPU systems. Here, we will explain the proposed methodology with a particular focus on the implementation designed for multi-GPU platforms.

Since the total view problem bears some resemblance to matrix processing, the proposal to accelerate the calculation of the total viewshed focuses on exploiting the intrinsic parallelism. In practice, we distribute the $ns/2$ sectors among all available devices so that each device is in charge of processing a similar number of sectors given by the chosen scheduling.

Each device sequentially launches three kernels, which we will describe below, to process the viewshed from each point in the DEM and corresponding sectors. For the sake of clarity, the kernels shown below are particularised for the sectors from 0 to 45° , although this process is carried out up to 180° —only changing the trigonometric function. After the processing stage, each device will contain partial viewshed results that the host will accumulate in the final stage to obtain the total viewshed. This accumulation method is executed on the host to avoid dependencies between threads while performing the viewshed computation.

In the following sections, the block and thread identification numbers are denoted as b_{id} and t_{id} , respectively, and the thread block dimension as b_{dim} .

4.3.4.1. Kernel-1: obtaining the *skwDEM* structure

This kernel is in charge of transforming the *DEM* into the *skwDEM* structure for a given sector. In this new model and for the chosen sector, points located consecutively in the terrain are also stored sequentially in memory, improving memory access performance. We used the interpolation method based on Bresenham's algorithm to soften the projection of each point.

As shown in Algorithm 4.5, each thread interpolates the corresponding point in the *DEM* according to its 2D thread identification number defined by i and j variables. y and $dest$ are used to obtain the target row of the *skwDEM*, which is stored in p . That value is then used to index the *skwDEM*, which is filled in with the corresponding elevation values as a function of r .

Algorithm 4.5 *Kernel-1 in charge of generating the $skwDEM$ structure from the DEM ($0^\circ < s < 45^\circ$).*

```

1: int  $i = b_{idy} \cdot b_{dim} + t_{idy}$  ▷ thread  $id_y$ 
2: int  $j = b_{idx} \cdot b_{dim} + t_{idx}$  ▷ thread  $id_x$ 
3: float  $y = \tan(s) \cdot j$  ▷ projection based on the sector (whole value)
4: int  $dest = y$  ▷ integer part of  $y$ 
5: float  $r = y - dest$  ▷ fractional part of  $y$ 
6: int  $p = dim_y + i - dest$  ▷ new vertical coordinate
7: if ( $i < dim_y$  &  $j < dim_x$ ) then
8:    $skwDEM[p][j] += (1 - r) \cdot DEM[i][j]$ 
9:    $skwDEM[p - 1][j] += r \cdot DEM[i][j]$ 
10: end if

```

An example of this methodology is presented in Figure 4.19 with three reference points and considering the sector 45° . We calculate their projections as follows:

- Point A with $(0, 0)$ coordinates in the input DEM :

$$p = dim_y + i - dest = dim_y + 0 - \tan(45) \cdot 0 \implies A' : (dim_y, 0)$$

- Point B with $(0.5 dim_y, 0.25 dim_x)$:

$$p = dim_y + 0.5 dim_y - 0.25 dim_x \implies B' : (1.25 dim_y, 0.25 dim_x)$$

- Point C with $(dim_y, 0.5 dim_x)$:

$$p = dim_y + dim_y - 0.5 dim_x \implies C' : (1.5 dim_y, 0.5 dim_x)$$

where $dim_x = dim_y$ since we considered a square DEM . The above procedure is repeated for each point in the DEM , obtaining the complete $skwDEM$ structure. Note that these calculations and the shape of the $skwDEM$ slightly change when considering the sector ranges $45-90^\circ$, $90-135^\circ$, and $135-180^\circ$. However, it is all reduced to simple trigonometric operations.

The best trade-off between the number of thread blocks and threads per block was found experimentally after performing different executions of the algorithm, resulting in $C_{by} = dim_y/8$ and $C_{bx} = dim_x/8$ 2D thread blocks with 8×8 threads per block.

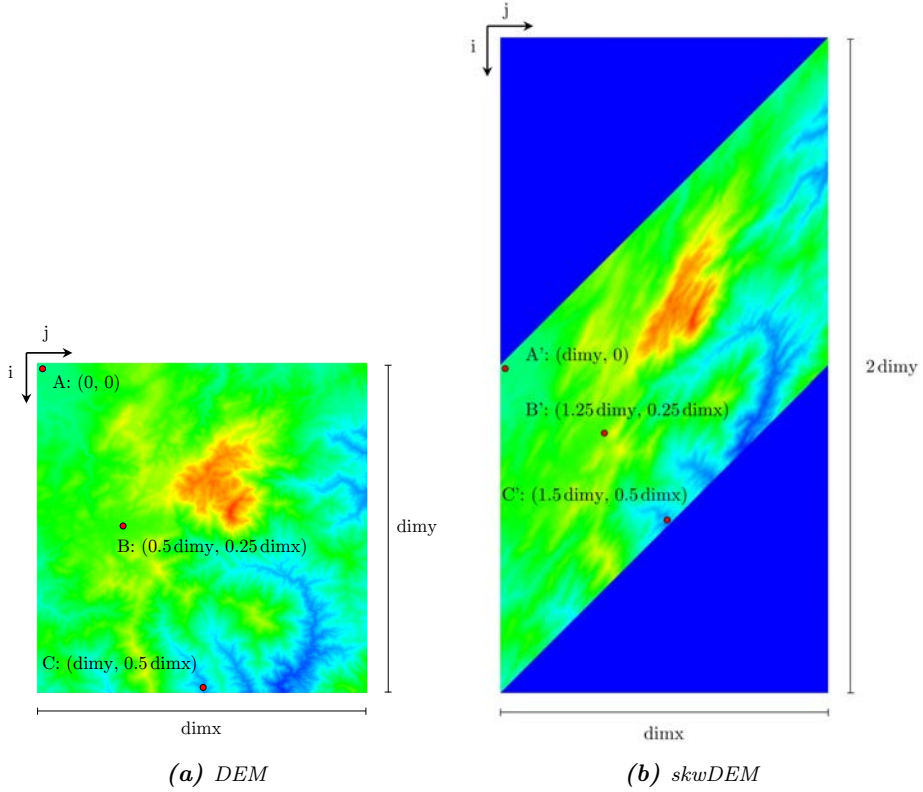


Figure 4.19: The DEM and skwDEM structures showing three reference points (A , B , C) and their projections (A' , B' , C') considering the sector 45° , for the sake of simplicity. (a) presents a DEM with $dimy \times dimx$ points where low elevation values are shown in blue, whereas red cells represent maximum values. (b) shows the corresponding skwDEM structure using the same colour scale.

4.3.4.2. Kernel-2: viewshed computation on the skwDEM

This kernel computes the viewshed for a specific sector and each point of the skwDEM, obtaining the skwVS matrix. Algorithms 4.6 and 4.7 show the pseudo-codes of this kernel.

Each thread manages a particular point $POV_t \in skwDEM, t = \{i, j\}$, where t is the corresponding 2D thread, and the variable h contains both the observer's height and the location elevation in this case. First, each thread obtains its computation range of non-zero values within the corresponding row (contained

Algorithm 4.6 *Kernel-2 in charge of the $skwVS$ computation on the $skwDEM$.*

```

1: int  $i = b_{idy} \cdot b_{dim} + t_{idy}$  ▷ thread  $id_y$ 
2: int  $j = b_{idx} \cdot b_{dim} + t_{idx}$  ▷ thread  $id_x$ 
3: float  $r = (1.0/\cos(s))^2$ 
4: float  $cv = 0$  ▷ viewshed value
5: if ( $i < 2 \cdot dim_y$  &  $j < dim_x$ ) then
6:   float  $h = skwDEM[i][j] + h_0$  ▷ elevation + observer's height
7:    $cv += \text{LINEAR\_VIEWSHED}(i, j, h, skwDEM, true)$  ▷ forward direction
8:    $cv += \text{LINEAR\_VIEWSHED}(i, j, h, skwDEM, false)$  ▷ backward direction
9:    $skwVS[i][j] = cv \cdot r$  ▷ ring-sector projection
10: end if

```

Algorithm 4.7 *Function to compute the viewshed in a row (Algorithm 4.6).*

```

1: function  $\text{LINEAR\_VIEWSHED}(i, j, h, skwDEM, forward)$ 
2:   if ( $forward$ ) then  $dir = 1$  otherwise  $dir = -1$  ▷ current direction
3:   int  $k = j + dir$  ▷ target cell in the row of the  $skwDEM$ 
4:   bool  $visible, above, opening, closing$ 
5:   float  $open\Delta d, \Delta d, \theta, max\theta = -\infty$ 
6:   while  $k \in nzSet$  do ▷ while k with non-zero value
7:      $\Delta d = |k - j|$  ▷ distance difference
8:      $\theta = (skwDEM[i][k] - h)/\Delta d$  ▷ current angle
9:     if ( $\theta > max\theta$ ) then  $above = 1$  ▷ point visible
10:     $opening = above \ \& \ !visible$  ▷ ring-sector begins
11:     $closing = !above \ \& \ visible$  ▷ ring-sector ends
12:     $visible = above$  ▷ save status for the next iteration
13:     $max\theta = \max(\theta, max\theta)$  ▷ save maximum angle
14:    if ( $opening$ ) then  $open\Delta d = \Delta d$ 
15:    if ( $closing$ ) then  $cv += \Delta d \cdot \Delta d - open\Delta d \cdot open\Delta d$  ▷ viewshed
16:     $k += dir$  ▷ next cell in the row
17:  end while
18:  return  $cv$ 
19: end function

```

in the $nzSet$ structure) from the $skwDEM$ structure (Figure 4.20a). Then, each one computes its visibility forwards and backwards across the row to which it belongs in a process called linear viewshed computation. The resulting viewshed value is thereafter stored in its corresponding position of the $skwVS$ structure (Figure 4.20b). It is important to highlight that the use of the $skwDEM$ structure makes it possible to take advantage of the data alignment as we obtain the viewshed results for the sectors s and $s + 180$ using the same $skwDEM$.

This kernel is launched with $2 \cdot C_{by}$ and C_{bx} 2D thread blocks using twice as many thread blocks as in the y-dimension (according to the size of the $skwDEM$).

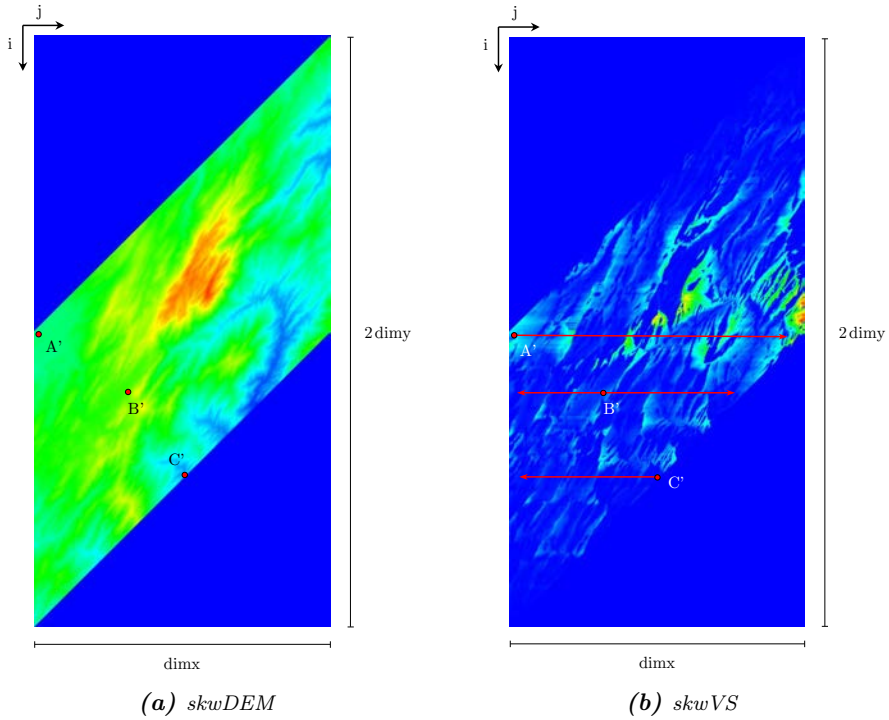


Figure 4.20: The $skwDEM$ and $skwVS$ structures showing the projection of three reference points (A' , B' , C') and considering the sector 45° , for the sake of simplicity. (a) presents a $skwDEM$ with a size of $2 \cdot dimy \times dimx$ where low elevation values are coloured as blue, whereas red cells represent maximum values. (b) shows the $skwVS$ structure resulting from computing the linear viewshed over the $skwDEM$. Red arrows represent the directions in which the linear viewshed is computed (only three points are shown). This map uses the same colour scale as the previous one, but measuring viewshed.

4.3.4.3. Kernel-3: obtaining the viewshed on the DEM

Once the linear viewshed computation on the *skwDEM* is completed for each POV and stored in the *skwVS* structure, this kernel transforms the *skwVS* structure by undoing the rotation operation of Kernel-1 to obtain the final viewshed *VS* structure on the DEM (Figure 4.21). The pseudo-code in charge of performing this procedure is presented in Algorithm 4.8. This kernel has the same execution configuration as Kernel-1.

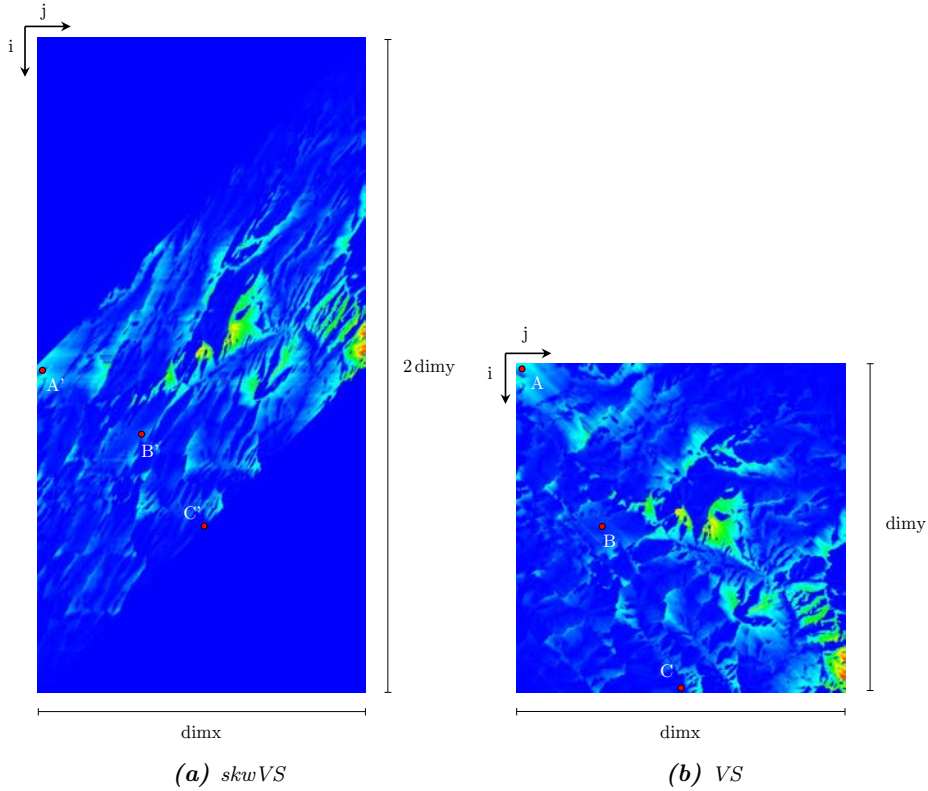


Figure 4.21: The *skwVS* and *VS* structures showing three reference points (A , B , C) and their projections (A' , B' , C') considering the sector 45° , for the sake of simplicity. (a) presents the *skwVS* with a size of $2 \cdot \text{dimy} \times \text{dimx}$ where low viewshed values are coloured as blue, whereas red cells represent maximum values. (b) shows the corresponding *VS* structure over the DEM using the same colour scale as the previous one.

Algorithm 4.8 *Kernel-3 in charge of transforming the $skwVS$ on the $skwDEM$ into the VS structure on the DEM ($0^\circ < s < 45^\circ$).*

```

1: int  $i = b_{idy} \cdot b_{dim} + t_{idy}$  ▷ thread  $id_y$ 
2: int  $j = b_{idx} \cdot b_{dim} + t_{idx}$  ▷ thread  $id_x$ 
3: float  $y = \tan(s) \cdot j$  ▷ undo projection based on the sector (whole value)
4: int  $dest = y$  ▷ integer part of  $y$ 
5: float  $r = y - dest$  ▷ fractional part of  $y$ 
6: int  $p = dim_y + i - dest$  ▷ new vertical coordinate
7: if  $i < dim_y$  &  $j < dim_x$  then
8:   float  $skwVS_a = skwVS[p][j]$ 
9:   float  $skwVS_b = skwVS[p-1][j]$ 
10:   $VS[i][j] += (1 - r) \cdot skwVS_a + r \cdot skwVS_b$  ▷ weighted value of viewshed
11: end if

```

4.3.4.4. Managing multiple GPUs on the host

In order to execute sDEM on a multi-GPU system, several steps must be followed (Algorithm 4.9):

1. The required global memory spaces must be reserved to allocate the different structures in each available device.
2. At this point, the DEM can be transferred from the host to each device (dev) of the total available devices (nd).
3. The target number of sectors $ns/2$ is distributed among the different devices using the modulus operator (%) to balance the workload.
4. Each device executes Kernel-1, Kernel-2, and Kernel-3, accumulating each independent viewshed computation result in their private VS_d structure, considering all the points and every target sector.
5. Finally, these private structures are transferred from each device to the host to carry out the final reduction in parallel, obtaining the total viewshed VS result for every sector.

Algorithm 4.9 *Host code in charge of managing multiple GPUs.*

```

1: for  $d = 0, nd$  do
2:    $dev_d \leftarrow Allocate(|DEM|, |skwDEM|, |skwVS|, |VS_d|)$   $\triangleright$  global device memory
3: end for
4: for  $d = 0, nd$  do
5:    $dev_d \leftarrow MemcpyAsyncH2D(DEM)$   $\triangleright$  asynchr. copy from host to device
6: end for
7: for  $s = 0, ns/2$  do
8:   int  $d = s \% nd$   $\triangleright$  distribution of sectors
9:    $dev_d \leftarrow Kernel - 1, 2, 3(s)$   $\triangleright$  execution of kernels
10:   $dev_d \leftarrow MemcpyAsyncD2H(VS_d)$   $\triangleright$  asynchr. copy from device to host
11: end for
12: for  $d = 0, nd$  do
13:   $dev_d \leftarrow cudaStreamSynchronize()$   $\triangleright$  wait for tasks to complete
14: end for
15: for  $d = 0, nd$  parallel do
16:   $VS += VS_d$   $\triangleright$  parallel reduction
17: end for

```

4.4. Experiments and results

This section assesses the performance of sDEM with respect to well-known GIS software and the state-of-the-art algorithm.

Firstly, the experimental setup is explained at the beginning of this section. Then, we present the comparison between sDEM and the most used GIS software in addressing the multiple viewshed problem. This analysis is followed by the evaluation of the computational performance of sDEM compared with the state-of-the-art algorithm using the total viewshed problem as a case study, where three different scenarios are considered: (i) total viewshed computation for a random direction, (ii) average total viewshed computation considering a set of sectors, and (iii) total viewshed map generation. Finally, we present the CUDA profiling of the multi-GPU implementation of sDEM and discuss the contributions of this methodology in the field of viewshed analysis.

4.4.1. Experimental setup

Based on the requirements of the experiments, we have set up two different systems with the following characteristics:

- **Windows OS:** Windows 10 with an Intel(R) Core(TM) i5-6500 CPU @3.20GHz with 4 cores (4 threads) and 8GB DDR4 RAM.
- **Linux OS:** Ubuntu 16.04.5 LTS with an Intel(R) Xeon(R) E5-2698 v3 @2.30GHz with 16 cores (32 threads), and 256GB DDR4 RAM, along with four GTX 980 Maxwell GPUs with 2048 CUDA cores, 16 SMs, 1.12GHz, and 4GB GDDR5 each one.

The experiment presented in Section 4.4.2 is executed on Windows operating system (OS) because GIS software requires this specific OS. In contrast, the experiments described in Sections 4.4.3 and 4.4.4 are executed on Linux OS to obtain an optimal measurement of the computational performance.

We designed these experiments to represent a real problem where obtaining visibility for a particular direction or region is necessary. That is the reason why we considered three DEMs of the Montes de Malaga Natural Park (Malaga, Spain), with 10 metres resolution each and different extents (Table 4.1). Raster-based DEMs are used because of their availability and simplicity in contrast to vector-based TIN models. Their configuration includes a grid of regularly spaced nodes, simplifying the mathematical operations.

Table 4.1: DEMs used as input data in the experiments.

DEM		UTM		
Dataset	Dimension	Zone ^a	Easting	Northing
DEM10m-2k	2000x2000(10m)	30S	0310000 mE	4070000 mN
DEM10m-4k	4000x4000(10m)	30S	0360000 mE	4100000 mN
DEM10m-8k	8000x8000(10m)	30S	0360000 mE	4140000 mN

^aLatitude band designator.

The input data used in this work is provided by the Regional Government of Andalusia in [132], which is currently available in the UTM format.² Thus, no transformation is required to conduct the geospatial analysis described in the following sections. In addition, the resolution that the UTM format provides is

²GIS software is required to access this link (e.g., QGIS or ArcGIS).

sufficient for the total viewshed problem; therefore, this is the coordinate system used in this work.

In the experiments below, we made the following assumptions:

- The calculation of the total viewshed is not only limited to the area of interest (Montes de Malaga Natural Park), but it includes the surrounding region, which contains both flat and mountainous areas.
- The visibility computation only considers the elevation of the terrain, excluding any other variable (e.g., vegetation).
- Lightning is appropriate and homogeneous throughout the territory.
- The terrain is static, i.e., it does not change over time.
- The observers are located 1.5 metres above the ground.

Regarding the implementation, the OpenMP API enables the multi-threaded execution in parallel of each selected sector with dynamic scheduling since it has proved to obtain the best performance. We compiled host codes using the g++ 5.4 open-source compiler, with `fast-math`, `fopenmp`, and `O2` optimisation flags. CUDA files make use of the NVIDIA NVCC compiler from the CUDA compilation tools V10.0.130. The multi-core implementation of the proposal is launched with the maximum number of threads available in the system, the same way that single-GPU and multi-GPU implementations are configured to operate the devices at full occupancy.

The GIS software used for comparison includes Google Earth Pro 7.3, QGIS-GRASS 3.10.2, and ArcGIS 10.7. In the last one, we used specifically the Spatial Analyst extension, which contains the Viewshed 2 (VS-2), Viewshed (VS), and Visibility (VI) tools.

4.4.2. Comparison with GIS software

Commercial software offers many more packages and options than visibility calculation, although we will limit this analysis to assessing their performance on this particular aspect. A fair comparison would be to compare sDEM and other GIS software/tools in computing the total viewshed using the same DEM. However, as there is no public software/tool capable of solving the total viewshed problem, we will compute the viewshed only from a set of points of the DEM.

This experiment assesses the computational performance of sDEM and the most used GIS software in solving the multiple viewshed problem. In particular, the objective is to compute the accumulated viewshed considering 10 POVs randomly chosen in the DEM10m-2k—a random number generator is used to determine the coordinates. We considered the single-threaded implementations of the sDEM proposal, QGIS-GRASS, and Google Earth, whereas ArcGIS had to be executed using all available cores.

Figure 4.22 and Table 4.2 show the time each software requires to complete the multiple viewshed computation. sDEM outperforms every analysed GIS software: ArcGIS VS-2 with two different configurations (23.7x, 2.4x), ArcGIS VS (1.3x), ArcGIS VI (13.4x), Google Earth³ (5.1x), and QGIS-GRASS (6.7x). Although sDEM achieves significant results in this experiment, the greatest gain is obtained in the total viewshed computation discussed below.

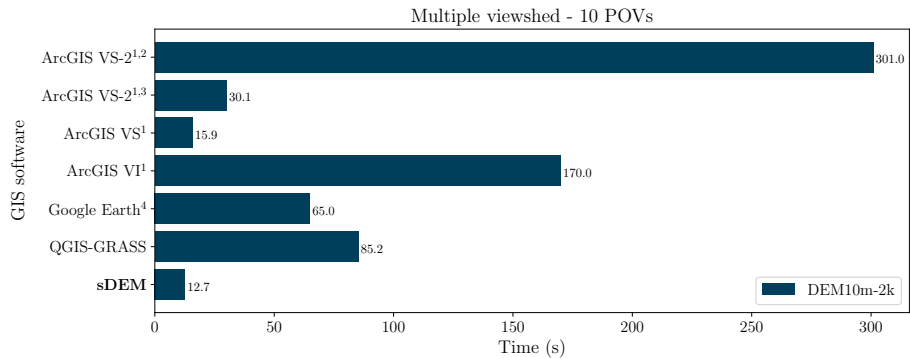


Figure 4.22: Computational performance comparison between sDEM and commonly used GIS software in solving the multiple viewshed problem, considering 10 POVs randomly located in the DEM10m-2k. We considered single-thread execution to obtain the run-time of those programs not otherwise indicated. VS, VS-2, and VI are the ArcGIS Spatial Analyst tools Viewshed, Viewshed 2, and Visibility, respectively. QGIS-GRASS uses the *r.viewshed* module. ¹multi-thread execution was required using the maximum number of cores available. ²using *PERIMETER_SIGHTLINES* parameter. ³using *ALL_SIGHTLINES* parameter. ⁴Google Earth does not have multiple/total viewshed computation capability, so the average time in computing singular viewshed has been multiplied by the number of POVs considered.

³The execution time obtained using Google Earth results from extrapolating the singular viewshed computation to the current case with 10 POVs since this software does not support this operation.

Table 4.2: Analysis of the results of the computational performance comparison between *sDEM* and commonly used GIS software (Figure 4.22). The execution time of *sDEM* has been used as a reference for the calculations.

GIS software	Time (s)	Time difference (s)
ArcMap SA-2 ^{1,2}	301.0	288.3
ArcMap SA-2 ^{1,3}	30.1	17.4
ArcGIS VS ¹	15.9	3.2
ArcGIS VI ¹	170.0	157.3
Google Earth ⁴	65.0	52.3
QGIS-GRASS	85.2	72.5
sDEM	12.7	-

4.4.3. Comparison with the state-of-the-art algorithm

Up to this point, *sDEM* and the TVS algorithm [110] are the only approaches in the literature capable of computing the total viewshed on entire datasets without carrying out prior reductions in workload. To achieve a similar workload for both algorithms, *dimx* has been chosen as the size of the BoS for the TVS algorithm so that it will match the number of points processed per row by *sDEM*.

Single-threaded, multi-threaded, single-GPU, and multi-GPU versions of the *sDEM* proposal were compared with the single-thread implementation of the TVS algorithm based on speed-up and throughput values. We analysed three different scenarios to assess the performance of both algorithms considering the total viewshed problem.

4.4.3.1. Total viewshed considering a random sector

Here, the computational performance of *sDEM* was analysed in computing total viewshed considering a single random sector, where the sector 10° was selected. Figure 4.23 along with Tables 4.3 and 4.4 present the speed-up curves and the throughput results (POVs processed per second) using three DEMs, as well as the run-time results.

sDEM outperformed the TVS algorithm, in the best case achieving a maximum acceleration up to 233.5x using the 1-GPU implementation on DEM10m-4k. Throughput results showed that this variable is approximately $2.11 \cdot 10^7$ POV/s

for the same implementation on DEM10m-2k, compared to the value $1.18 \cdot 10^5$ POV/s achieved by the single-threaded implementation of TVS.

Note that the multi-GPU implementation was not considered in this experiment due to the low workload when distributing one sector among more than one device.

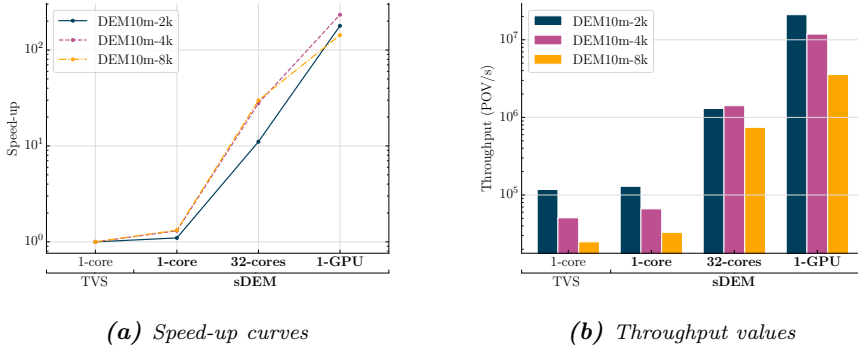


Figure 4.23: Speed-up curves and throughput values for the state-of-the-art total viewshed algorithm, TVS [110] and the different implementations of sDEM using single-core, multi-core, and single-GPU platforms to compute the total viewshed considering a randomly selected sector, $s = 10^\circ$ (BoS size of $\text{dim}x$ points for the TVS algorithm). Each colour corresponds to a particular dataset. The logarithmic scale is used for the y-axis.

Table 4.3: Run-time and speed-up results of sDEM and TVS in total viewshed computation considering a random sector $s = 10^\circ$ (data from Figure 4.23a).

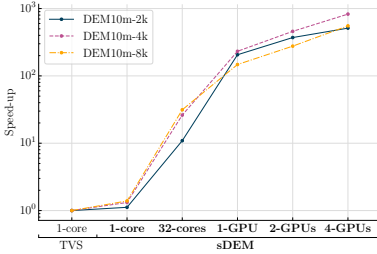
		DEM10m-2k		DEM10m-4k		DEM10m-8k	
Algorithm	Platform	Time (s)	Speed-up	Time (s)	Speed-up	Time (s)	Speed-up
TVS	1-core	33.99	-	315.26	-	2571.66	-
sDEM	1-core	30.92	1.1	241.68	1.3	1939.99	1.3
	32-cores	3.07	11.1	11.31	27.9	85.96	29.9
	1-GPU	0.19	178.9	1.35	233.5	17.95	143.3

Table 4.4: Throughput (Thp) results of *sDEM* and TVS in total viewshed computation considering a random sector $s = 10^\circ$ (data extracted from Figure 4.23b divided by 10^5).

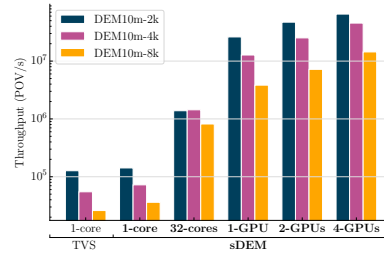
Algorithm	Platform	DEM10m-2k	DEM10m-4k	DEM10m-8k
		Thp (POV/s)	Thp (POV/s)	Thp (POV/s)
TVS	1-core	1.18	0.51	0.25
sDEM	1-core	1.29	0.66	0.33
	32-cores	13.03	14.15	7.45
	1-GPU	210.53	118.52	35.65

4.4.3.2. Total viewshed based on average values

In this experiment, unlike the prior one, the direction range was selected from 0 to 45° to obtain the average values per sector. This choice of design lies in the fact that single-threaded executions of TVS and *sDEM* required several weeks to complete when using a higher range. Furthermore, results within this range are representative and can be extrapolated to any target range. Figure 4.24 along with Tables 4.5 and 4.6 introduce the speed-up curves, the throughput achieved, and the run-time results.



(a) Speed-up curves



(b) Throughput values

Figure 4.24: Speed-up curves and throughput values for the state-of-the-art total viewshed algorithm, TVS [110] and the different implementations of the *sDEM* proposal using single-core, multi-core, single-GPU, and multi-GPU platforms to compute the total viewshed. Directions fulfilling $0^\circ < s < 45^\circ$ are considered to obtain average values per sector (BoS size of $\text{dim} \times \text{points}$ for the TVS algorithm). Each colour corresponds to a particular dataset. The logarithmic scale is used for the y-axis.

Table 4.5: Run-time and speed-up results of *sDEM* and *TVS* in total viewshed computation considering sectors $0^\circ < s < 45^\circ$ (data from Figure 4.24a).

		DEM10m-2k		DEM10m-4k		DEM10m-8k	
Algorithm	Platform	Time (s)	Speed-up	Time (s)	Speed-up	Time (s)	Speed-up
TVS	1-core	1417.61	-	13121.14	-	110671.86	-
sDEM	1-core	1268.50	1.1	9951.31	1.3	79642.97	1.4
	32-cores	130.02	10.9	501.34	26.2	3526.20	31.4
	1-GPU	6.88	206.0	56.55	232.0	750.75	147.4
	2-GPUs	3.82	371.1	28.60	458.8	400.18	276.6
	4-GPUs	2.77	511.8	15.86	827.3	200.00	553.4

Table 4.6: Throughput (*Thp*) results of *sDEM* and *TVS* in total viewshed computation considering sectors $0^\circ < s < 45^\circ$ (data extracted from Figure 4.24b divided by 10^5).

		DEM10m-2k	DEM10m-4k	DEM10m-8k
Algorithm	Platform	Thp (POV/s)	Thp (POV/s)	Thp (POV/s)
TVS	1-core	1.27	0.55	0.26
sDEM	1-core	1.42	0.72	0.36
	32-cores	13.84	14.36	8.17
	1-GPU	261.63	127.32	38.36
	2-GPUs	471.20	251.75	71.97
	4-GPUs	649.82	453.97	144.00

Best-studied cases showed that the maximum speed-up achieved is up to 827.3x with the 4-GPUs implementation with respect to the TVS algorithm considering DEM10m-4k. Throughput results showed that this variable is $6.50 \cdot 10^7$ for the same implementation on DEM10m-2k, compared to the value $1.27 \cdot 10^5$ achieved by the single-threaded implementation of TVS.

4.4.3.3. Total viewshed map generation using sDEM

Figure 4.25 shows the final total viewshed map of the Montes de Malaga Natural Park (Malaga, Spain) obtained using the sDEM algorithm.

No substantial differences have been found after analysing the absolute and relative differences when comparing the viewshed results of TVS and sDEM. Particularly, for the DEM10m-2k used in this study, the absolute difference found was 1.18%, whereas the relative difference was 4.21%. These values are within the limits recommended in this field [130].

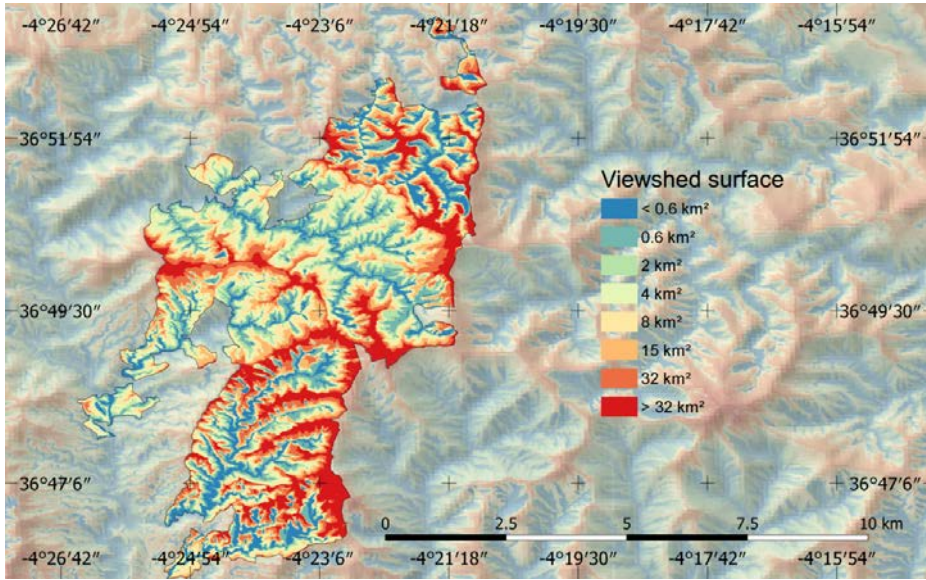


Figure 4.25: Total viewshed map of the Montes de Malaga Natural Park and its surroundings in the province of Malaga (Spain) generated using sDEM.

4.4.4. Multi-GPU implementation profiling

Here, we present the profiling results of the multi-GPU implementation of sDEM obtained on the Linux OS server and considering the DEM10m-2k as input. We used the guided application analysis provided by NVIDIA Visual Profiler (NVVP)—available for the distribution of CUDA already installed in the system. We compiled the device code using the following options to target the underlying GPU architecture (Maxwell for GTX 980): `-lineinfo -gencode arch=compute_52, code=sm_52`.

4.4.4.1. Application timeline

Figure 4.26 displays the timeline of the multi-GPU implementation of sDEM. From left to right, this timeline shows the execution of the three kernels for each range of sectors (differentiated by colour): $0-45^\circ$, $45-90^\circ$, $90-135^\circ$, and $135-180^\circ$, where each device processes a subset within each range. Based on this result, we can conclude that our implementation successfully distributes the workload equally among the GPUs.

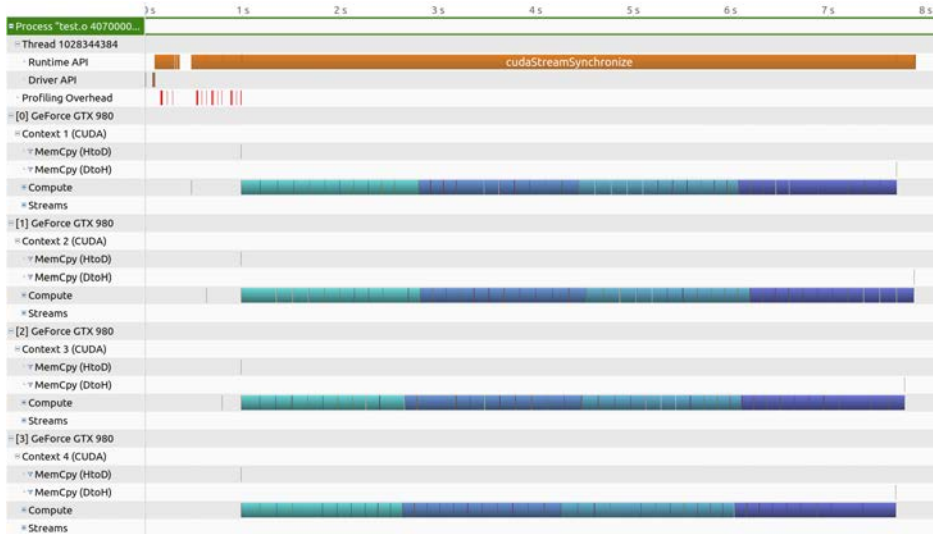


Figure 4.26: Timeline of the multi-GPU implementation of sDEM running on the Linux OS server with four GeForce GTX 980.

For the sake of simplicity, this profiling analysis targets only the main kernels involved in computing the viewshed from 0 to 45° . The difference between the three kernels of this range and those of any other range lies in the line of code containing the trigonometric operation that transforms the DEM into the *skwDEM*.

4.4.4.2. Performance-critical kernels

In practice, a streaming multiprocessor (SM) has a maximum number of warps that can be active simultaneously during processing. Based on this fact, *occupancy* is defined as the ratio between the active warps in a SM and the maximum

number of active warps supported by the SM [133]. Accordingly, we distinguish between theoretical (maximum) and achieved (real) occupancy, where both depend on the release configuration, kernel compilation options, and device capabilities. In sDEM, we chose the combination of blocks per grid and threads per block that accomplished the highest occupancy and the lowest execution time for each kernel execution configuration. Thus, we had two options for each kernel:

- Set the minimum number of threads and blocks that reach 100% of theoretical occupancy in the GPUs. With this approach, there would be more points in the DEM than active threads, so each thread would have to compute the viewshed of more than one point.
- Set as many active threads as there are points in the DEM. This approach would result in an overload of the SMs, but each thread would only calculate the viewshed of one point.

According to the terms mentioned above, the second approach proved to be the best. Under these conditions, we used NVVP to produce a ranked list of kernels based on the optimisation importance: Kernel-2 scored 100 (high importance), whereas Kernel-1 and Kernel-3 scored 1 (low importance). As expected, the most relevant kernel is the one in charge of computing the viewshed on the already skewed DEM, although we will present the results of the three kernels.

- Kernel-1 (low importance)

This kernel was launched with [250,250,1] thread blocks and [8,8,1] threads per block—the DEM size is 2000x2000. The performance limiter analysis of Kernel-1 (Figure 4.27) reveals that this kernel is compute bound since the GPU memory

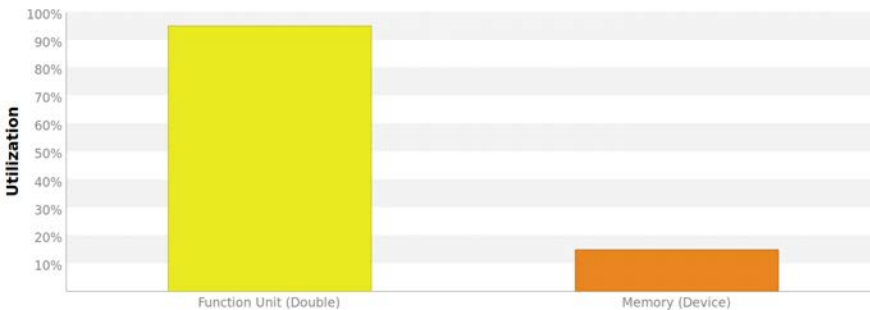


Figure 4.27: Kernel-1 performance limiter analysis.

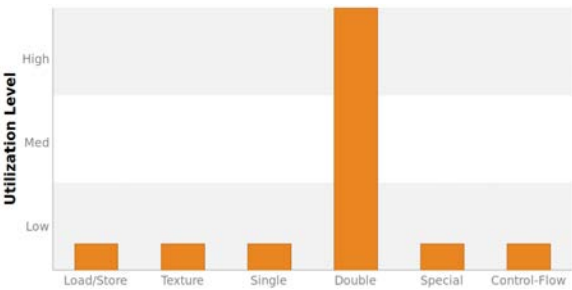


Figure 4.28: Kernel-1 compute analysis.

utilisation is significantly lower than its compute utilisation. The GPU utilisation analysis (Figure 4.28) also confirms this issue: the overuse of the double function unit is limiting the kernel’s performance. It is likely caused by the arithmetic operation that transforms the DEM into the skwDEM structure.

This kernel’s achieved occupancy (Figure 4.29) is 93%, which indicates that occupancy is not an issue. Moreover, the SMs utilisation analysis (Figure 4.30) shows that each SM is being fully utilised during the execution of this kernel.

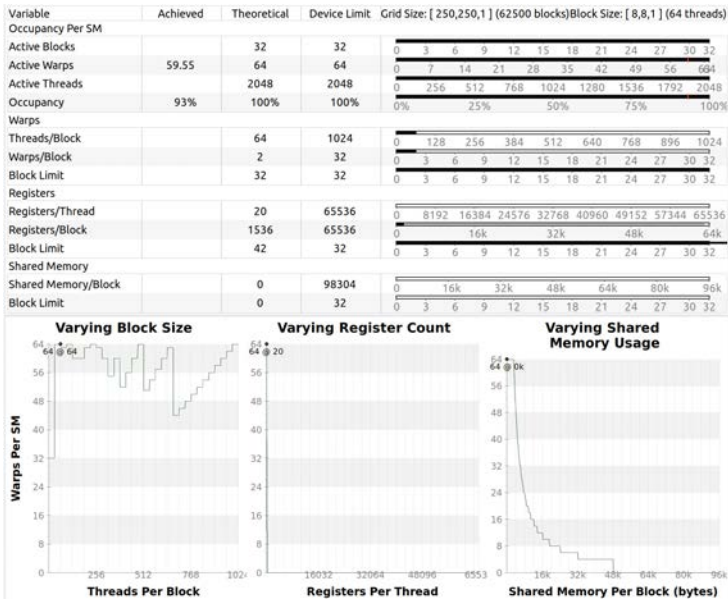


Figure 4.29: Kernel-1 occupancy analysis.

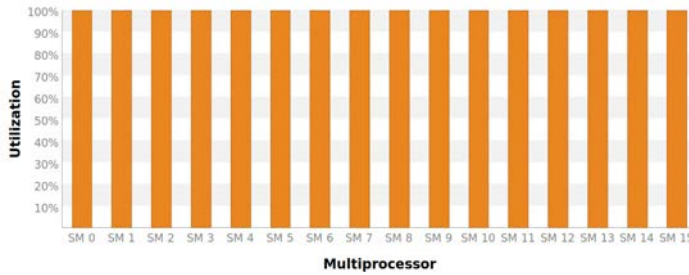


Figure 4.30: Kernel-1 SM utilisation analysis.

Finally, the PC sampling of this kernel (Figure 4.31) demonstrates again that this kernel is execution-dependent. This problem is possibly caused by the tangent calculation necessary for the data relocation process.

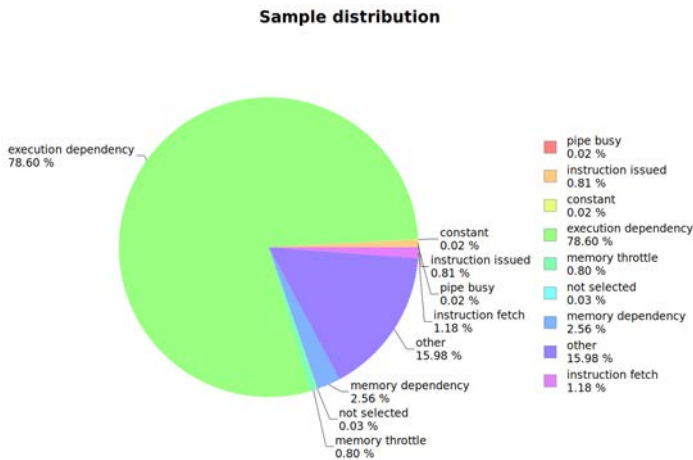


Figure 4.31: Kernel-1 profile based on PC sampling.

- Kernel-2 (high importance)

The execution configuration of this kernel was set to [500,250,1] thread blocks with [8,8,1] threads per block—the skwDEM size is 4000x2000. The performance limiter analysis of Kernel-2 (Figure 4.32) indicates that its performance is most likely limited by the latency of arithmetic or memory operations.



Figure 4.32: Kernel-2 performance limiter analysis.

The compute analysis (Figure 4.33) does not reveal any limitation regarding the overuse of a specific function unit. This kernel behaves as expected due to the large number of memory accesses to the DEM structure necessary to retrieve the elevation of each point to check if they are visible from the origin point for a specific sector.

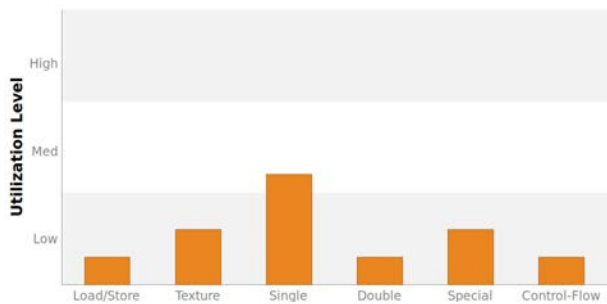


Figure 4.33: Kernel-2 compute analysis.

This kernel’s occupancy analysis (Figure 4.34) indicates that its achieved occupancy is 98.4%, which suggests that there are no issues related to this variable. This result is also corroborated by the 100% utilisation value measured in each of the 16 SMs (Figure 4.35).

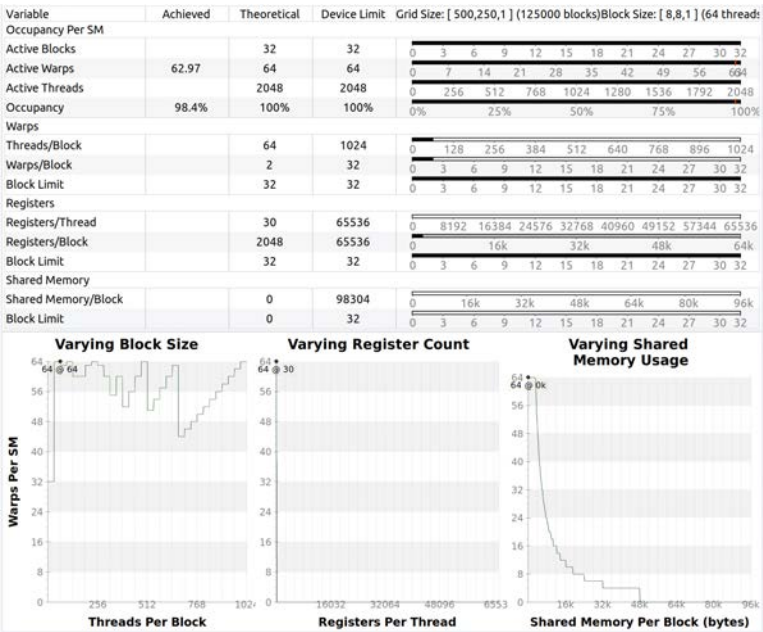


Figure 4.34: Kernel-2 occupancy analysis.

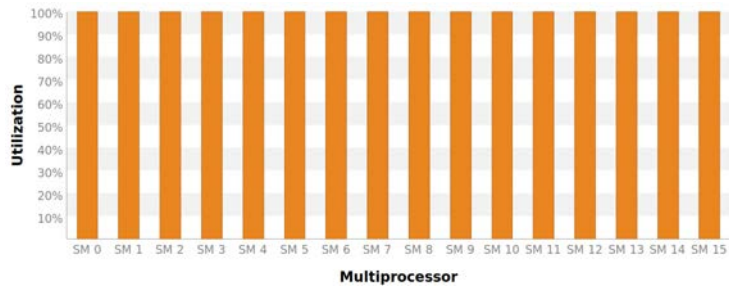


Figure 4.35: Kernel-2 SM utilisation analysis.

Finally, the PC sampling-based profile (Figure 4.36) shows that this kernel is memory bound, the same conclusion drawn from the performance analysis shown above. It is most likely that many simultaneous memory requests of the same type cannot be satisfied during processing.

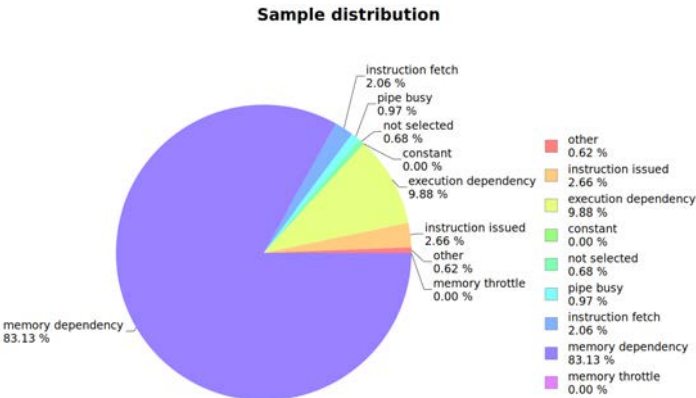


Figure 4.36: Kernel-2 profile based on PC sampling.

- Kernel-3 (low importance)

This kernel was launched with the same execution configuration as Kernel-1. This is because Kernel-3 is in charge of undoing the data transformation performed by the first kernel, so the two behaviours are similar. Therefore, everything discussed in the section associated with Kernel-1 applies here, although we will provide all the graphs below as well (Figure 4.37-4.41). The only difference between these two kernels is related to the achieved occupancy result (Figure 4.39): 91,7% for this kernel and 93% for Kernel-1.

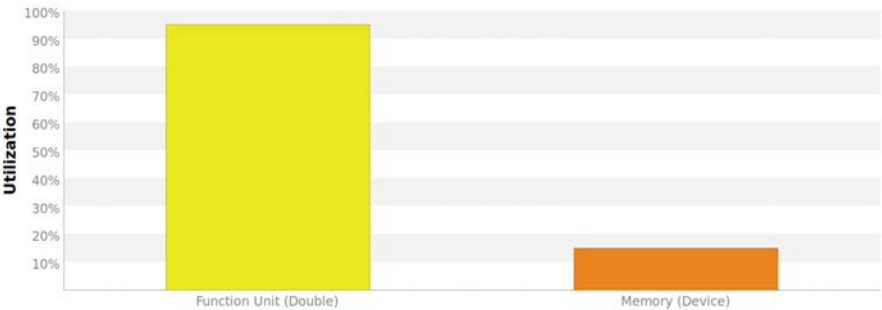


Figure 4.37: Kernel-3 performance limiter analysis.



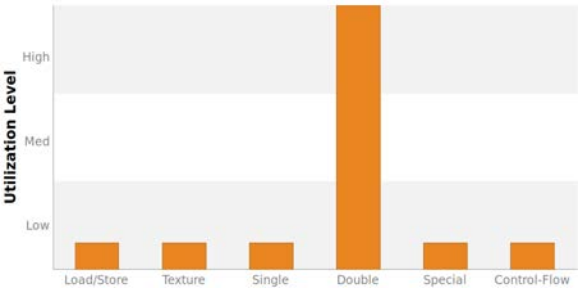


Figure 4.38: Kernel-3 compute analysis.

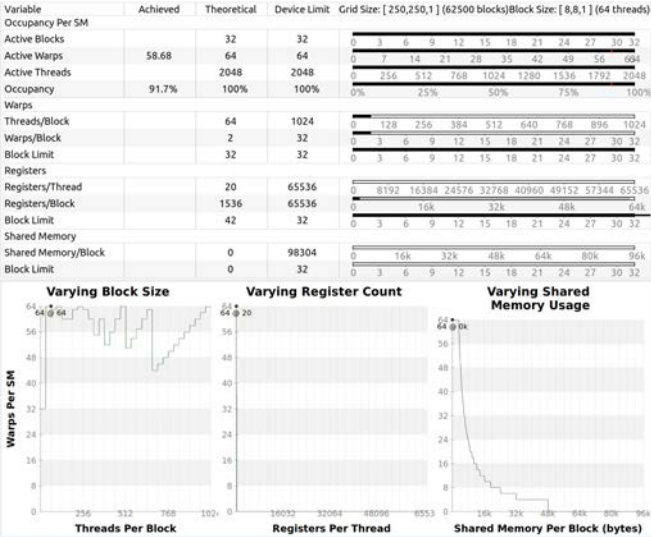


Figure 4.39: Kernel-3 occupancy analysis.

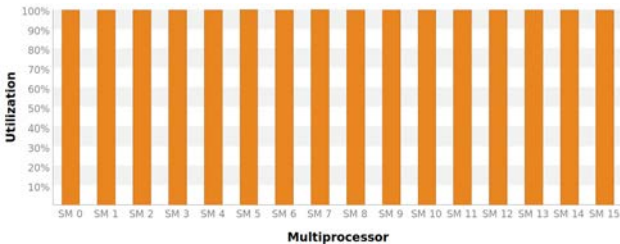


Figure 4.40: Kernel-3 SM utilisation analysis.

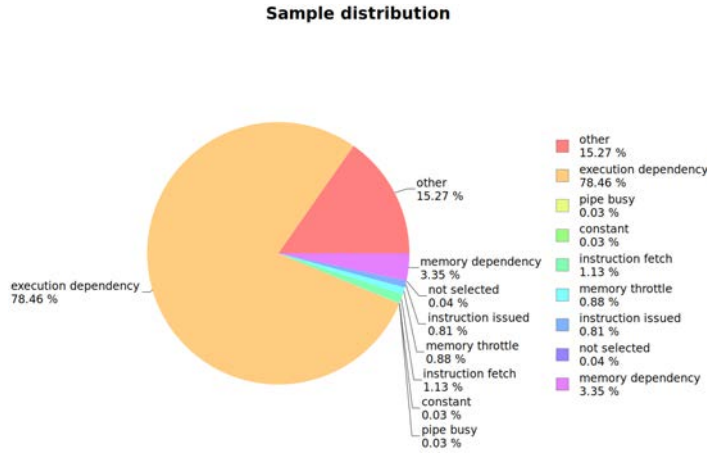


Figure 4.41: Kernel-3 profile based on PC sampling.

4.4.5. Analysis of the results

To solve the total viewshed problem, we designed the sDEM algorithm based on an unusual technique that may seem expensive at first glance: relocating the data of the DEM in memory at the beginning of each iteration of the process and before calculating the viewshed. Nevertheless, the computational cost of reorganising the data—both in terms of memory storage and workload—resulted negligible compared to the high computational demand of the total viewshed problem.

Firstly, the data relocation technique improves the performance of memory accesses, resulting in regularity. Secondly, sDEM exploits the main advantage of the total viewshed problem, which is related to its very high computational intensity of several hundred floating-point operations for each input/output data. These two aspects allowed us to efficiently port the algorithm to GPUs, as shown in the profiling analysis given above.

Compared to the state-of-the-art algorithm, the outcomes of the improvement in both locality and regularity cannot be reduced to just an acceleration of approximately 2 and 3 orders of magnitude; the improvement in the potential parallelism of the algorithm is much more significant. In the case of TVS, the parallelism is only applicable to the sector loop so that the maximum acceleration factor achievable is $ns/2$ at best. Conversely, sDEM uses a finer grain level of parallelism with a potential of $\sqrt{N} \cdot ns/2$ threads in simultaneous execution.

Finally, we should highlight that sDEM was specifically developed to compute the total viewshed for a given terrain, so its performance degrades as the number of points to be analysed is reduced. Generally, the time taken by the data restructuring is negligible compared to the total time spent in the total viewshed computation, but becomes considerable as fewer points are analysed. Despite this, sDEM still outperforms the most used GIS software and the state-of-the-art algorithm in computing singular and multiple viewsheds.

5 Path Planning Based on Visibility Data

The use of Unmanned Aerial Vehicles (UAVs) for remote sensing tasks has grown enormously in recent years. These include environmental and civil monitoring, disaster management, and forest fire fighting. In all these situations, a quick and early response is required to detect the hazard in time. This condition makes it essential to maximise the terrain covered during the flight, especially when the area to be monitored is irregular, large, and includes many blind spots. For this purpose, state-of-the-art total viewshed algorithms can be of great help: we can use them to analyse extensive areas and generate alternative paths providing all-round visibility from visibility information.

This chapter shows how the total viewshed computation is an invaluable tool to generate paths providing maximum visibility during flight. To demonstrate this, we introduce a new heuristic called *Visibility-based Path Planning* (VPP) that offers a different solution to the path planning problem. We can use VPP to generate safe and flyable paths to monitor difficult areas of complex terrain using the onboard camera of a single UAV, with forest fire prevention as a case study.

Moreover, the pointing direction of the onboard camera for each waypoint included in the path is also determined, avoiding overlapping of already monitored areas to increase the amount of terrain covered. Throughout the flight, the images can be analysed in real time to alert local authorities in case of fire.

The contributions of this chapter are the following:

- We presented a new heuristic called Visibility-based Path Planning (VPP). It finds the waypoints that will form a path providing high visibility for a single UAV based on viewshed analysis. This analysis enables the identification of the most hidden areas, achieving the monitoring of the entire region of interest to prevent fires at relatively low cost.
- We described the steps followed to compute the camera direction for each waypoint included in the path. Such directions are determined to maximise the covered area, avoiding overlapping views between adjacent points.

Section 5.1 introduces the path planning problem and every variable involved, with a special focus on fire prevention. Section 5.2 describes the state-of-the-art works using UAVs. Section 5.3 describes the proposed VPP heuristic that generates paths providing high visibility from viewshed data. Section 5.4 describes each step followed to set the camera direction for each waypoint of the path in order to enhance coverage. Finally, Section 5.5 presents the results obtained after applying the VPP heuristic in the case study areas.

5.1. Introduction

For the correct understanding of this chapter, it is highly recommended to have a basic understanding of the principal features involved in the visibility problem and how it is computed for a target area. We have discussed both concepts at the beginning of the previous chapter. Here, we will delve deeper into the visibility problem in the context of finding flyable paths that maximise the area covered for monitoring purposes.

First of all, the main advantages of the use of UAVs for forest fire prevention are explained in this section. Secondly, the path planning concept and its main methods are then described. Finally, we will present one of the most well-known problems in this field: the travelling salesman problem.

5.1.1. Forest fire prevention using UAVs

As mentioned earlier, a complete visibility analysis of a target region involves finding the areas visible from each point in the terrain. It is a tool of paramount

importance in Geographic Information System (GIS) software, which can contribute majorly to nature conservation. In this regard, monitoring and detection systems are essential to fight against forest fires.

In recent years, novel advanced technologies have emerged, e.g., systems using infrared rays and cameras installed in fixed and mobile terrain locations and aerial and satellite environments. These systems are implemented in forest regions with a high fire risk to detect them as soon as possible, but this has not always been the case. The first monitoring systems were typically based on the observation of the forest area carried out by people destined to this end, called forest rangers. Such personnel were located at fixed points within the territory, moving around when required. However, the monitoring systems that require human personnel suffer from significant shortcomings:

- These particular approaches are not suitable for situations requiring real-time response, as a human detects and reports the problem.
- Personnel safety is compromised by being close to the hazard, putting human lives at risk.
- These monitoring approaches do not provide a precise and global understanding of the magnitude of a hazard, e.g., a natural disaster.

The above are only some of the most significant drawbacks of human-based monitoring approaches. Thus, monitoring forest regions of complex terrain is still a challenging task.

We can increase the effectiveness in the fight against forest fires with the arrival of new technologies such as Unmanned Air Vehicles (UAVs), also known as drones. They can fly autonomously or be remotely controlled, and this is why the first known uses of this technology had military purposes. Nevertheless, such systems have become more useful in civil applications as many companies started to research and use them with new objectives. Therefore, UAVs can play a very important role as a preventive measure in firefighting (Figure 5.1). For example, a person can control a UAV using a joystick or mobile application to fly over a specific forest area during daylight hours. Still, the most interesting approach is the one where the UAV flies autonomously following a preset path—this new option is available for the latest devices.

Nowadays, the uses of UAVs are evolving towards much more specific tasks, such as remote-sensing [135], disaster prevention [136], operational support during emergencies [137], damage monitoring in critical infrastructures [138], traffic control [139], and environmental monitoring [140]. All these applications have



Figure 5.1: Photograph taken of a drone flying over a forest region during a fire [134].

in common the need to collect vast amounts of data, making UAVs suitable for this task. They carry multiple sensors and processing units that enable the acquisition of datasets on demand during the flight over a target area. Many other advantages of using UAVs include:

- Low cost of deployment.
- High visibility of the terrain due to low flight height.
- High capacity to reach remote locations and capture high-resolution images.
- Accurate measurements of environmental variables such as temperature, humidity, atmospheric pressure, and vegetation health.
- Adequate ability to compute lightweight algorithms.
- High safety conditions for all personnel.

In real life, a single UAV typically monitors small areas, whereas coordinated groups of UAVs usually cover larger ones. In the latter case, getting a path providing all-round visibility can certainly reduce not only the number of UAVs required to cover the region but also the time required to complete the task. The discretized approach of this problem is similar to the well-known observers siting problem [141, 112]. It focuses on finding the minimal number of observers covering the target area after merging their viewsheds. Therefore, the entire area is covered with the minimum number of monitoring locations, improving response times to unexpected events. This fact is one of the key goals pursued by the research in the field of path planning for UAVs.

5.1.2. Path planning

Motion planning, commonly known as *path planning*, is a classic problem in robotics that has been gaining importance since the middle of the last century. It is a multi-objective optimisation problem that seeks to find the optimal or near-optimal path for a specific vehicle to move from an initial point A to a target point B based on optimising one or more of the variables involved (e.g., travel time) and avoiding obstacles. The result is typically the shortest path between the initial and target locations.

The first step towards finding a solution to the path planning problem is defining the characteristics of an optimal path. For example, it is inevitable to consider a balance between length, safety, and smoothness. Some of the most important variables involved in the problem at hand are the following:

- **Travel time:** the vehicle should complete the journey in the shortest possible time. However, this condition may not be met due to interference with other criteria, such as those related to safety. In this case, the vehicle must reach the target location within a reasonable time.
- **Length:** the path generated should be as short as possible. This term, as well as travel time, directly affects the energy consumption of the vehicle.
- **Energy:** another well-studied criterion in the generation of paths. Finding the shortest path is pointless if the vehicle consumes so much energy to travel through it that it fails to reach the target waypoint. In this case, the cost function comes into play.
- **Safety:** the route must also meet all safety criteria imposed from the outset of the project, ensuring that, at the very least, the robot does not suffer any damage.
- **Smoothness:** the waypoints included in the path should be as distributed as possible across the target area so that connecting them results in a smooth trajectory for the vehicle to follow.

However, these are not the only variables involved in the problem of generating paths. Many others can be considered such as weight, speed, resolution, visibility, and weather. All this makes this problem one of the most complicated in robotics.

The type of vehicle is also relevant, where we can mainly distinguish between ground and aerial vehicles (Figure 5.2). In the case of ground vehicles, the specific variables to be considered during path planning may involve the study of

orography, slope, terrain type, and obstacles (e.g., rock formations and water sources). In contrast, the restrictions are different for aerial vehicles as the main hazards are obstacles, such as trees, buildings, and power lines, depending on the area overflown. Accordingly, terrain orography is of relatively minor importance since, for instance, the slope is not considered in this other problem, although high-altitude rock formations are. In addition, other specific considerations when using aerial vehicles are the maximum flight height and turn restrictions.

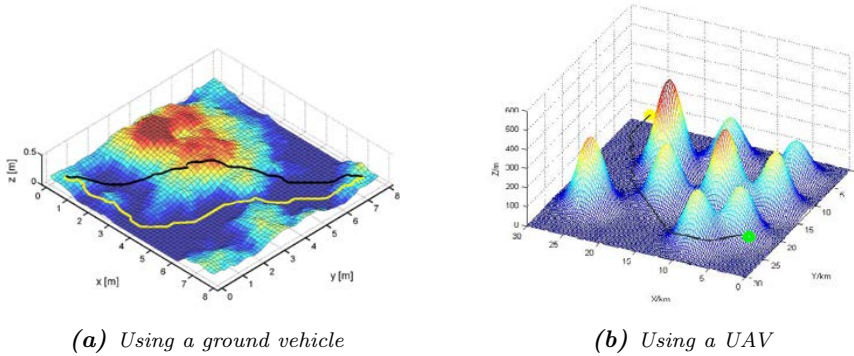


Figure 5.2: Two different path planning strategies depending on the means of transport by which the vehicle is travelling in areas of complex terrain: land and air [142, 143].

5.1.2.1. Classification of path planning methods

Depending on the degree of knowledge of the environment information, the literature identifies two main types of path planning methods [144]: global path planning (GPP) and local path planning (LPP).

GPP, also known as offline path planning, involves generating obstacle-free paths based on the information available from the full knowledge of the environment. The path is usually generated offline, i.e., prior to the start-up of the vehicle. Such methods can find the optimal path as the map and the coordinates and sizes of the obstacles are known from the beginning.

LPP, also known as online path planning, uses little or no prior information from the environment. In this case, the information is obtained through sensors installed in the vehicle to generate different paths avoiding obstacles in real time—while the vehicle moves. Therefore, finding an optimal or near-optimal solution with LPP is extremely difficult and is out of the scope of this work, where we will focus on GPP.

5.1.2.2. Global path planning

GPP is a two-step process that requires first establishing a model of the environment, then choosing an appropriate path planning algorithm [145].

- Map representations

As a first step, we need to build a map suitable for processing either in advance or in real time. Afterwards, we can split this space into two subspaces:

- **Free space** through which the vehicle can move safely (S_{free}). The path is generated in this particular space, in which the points that make it up are commonly known as nodes or waypoints.
- **Occupied space** where the obstacles are located (S_{obs}). It contains a precise and detailed representation of the spatial location of all objects.

The result is a comprehensive spatial model/map suitable for processing using pathfinding algorithms as it includes every obstacle and geographical feature presented in the orography of the terrain. There are currently many widely used methods for modelling the environment when both S_{free} and S_{obs} are known, among which we can highlight the following:

- **Cell decomposition:** it is one of the most straightforward techniques but with the best cost/difficulty ratio for the representation of the environment. Cell decomposition consists in decomposing the environment into a set of either uniform or non-uniform cells. This is why such techniques are also known as grid-based approaches [146]. We can distinguish two main cell decomposition strategies for modelling the environment:
 - **Exact/Accurate cell decomposition:** each of the established cells falls entirely in the obstacle-free space or the space occupied by obstacles (Figure 5.3a). The use of this strategy is lossless because the sum of the obstacle-free cells is equal to S_{free} . This technique can be understood as a vertical sweep from left to right where the line stops each time it meets the edge of an obstacle, forming the cell from the previous leftmost vertical line to where the new one has stopped. The complexity, measured in the resulting number of cells, depends on the geometry of the environment and the number of obstacles and vertices. The path is generated, e.g., by connecting the centres of adjacent cells from the starting point to the target one.

- **Approximate cell decomposition:** in this case, the cells can simultaneously contain both free space and an obstacle partially or completely. As a rule of thumb, cells that contain some part of an obstacle are included in S_{obs} , whereas the rest of the cells belong to S_{free} . There are two types of cells:
 - **Constant cell size:** the simplest option implies that each cell has the same size, but some information may be lost and is, therefore, not a lossless method (Figure 5.3b). The main disadvantage is that this technique requires a lot of memory space, either by considering the environment too large or by using tiny cells.
 - **Variable cell size:** a more optimised option from a memory usage point of view is to define cells of variable size following a tree data structure called quadtree (Figure 5.3c). In this process, we start with a single cell for the whole environment, and if all the space it covers belongs entirely to S_{free} or S_{obs} , the cell will not change; otherwise, it will be divided into four smaller cells. This process is repeated until the above condition is fulfilled or a maximum resolution is reached. This method is simpler than the accurate cell decomposition one. However, some path planning algorithms will not find a solution to the problem due to the loss of information—although the solution may exist.

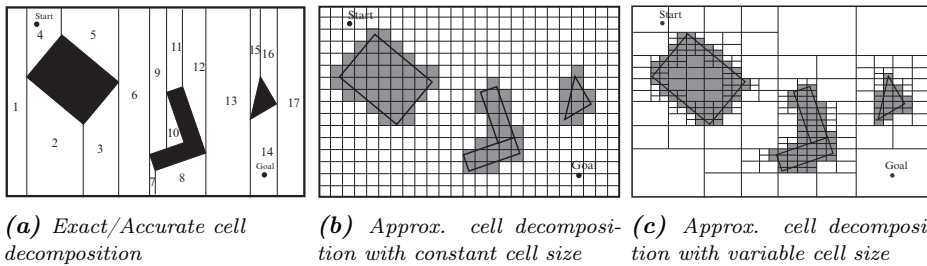


Figure 5.3: Map representations using cell decomposition methods [145].

- **Roadmap:** a series of lines or curves are established which, together with their intersection points called nodes, provide possible connections between points in S_{free} to form a path [145]. This method heavily depends on the environment's geometry as the aim is to find the fewest number of roads that will allow the robot to find its way through the free space. The literature distinguishes two different roadmap building algorithms:

- **Visibility graph:** this type of graph is created by connecting all those vertices that belong completely to S_{free} using lines (Figure 5.4a). Following an iterative process, each vertex is taken as the reference, and all possible lines connecting it to the other visible vertices are drawn. In addition, the starting and target points are also considered as vertices. The result is usually a path close to the obstacles, which is the shortest and the most dangerous, so the distance to obstacles is usually increased to avoid collisions. This strategy is simple but results in greater complexity and lower efficiency as the number of obstacles and vertices increases.
- **Voronoi graph:** this method decomposes S_{free} into a series of regions called Voronoi tessellations (Figure 5.4b). Each line/road dividing one partition from the others is formed from the union of those points equidistant to the two nearest shapes in S_{obs} . Depending on the shape of the object in question, a Voronoi curve can be a straight line (between two vertices or edges) or a quadratic curve (between a vertex and a straight line). Using the Voronoi road map minimises the risk of collision with obstacles when travelling on the generated roads by maximising the distance to obstacles. However, as a result, the generated trajectory is not optimal in terms of length.
- **Triangulation:** this strategy involves the decomposition of the environment into triangular cells (Figure 5.4c), where Delaunay triangulation is one of the most famous algorithms. The three nearest points form each triangle, and each line segment does not intersect with others. An important feature is that the resulting graph will always be the same no matter where you start building it. In addition, the Delaunay triangulation corresponds to the dual graph of the Voronoi diagram. The centre of each triangular cell (centre of the circumscribed circle) coincides with each vertex of the Voronoi graph.

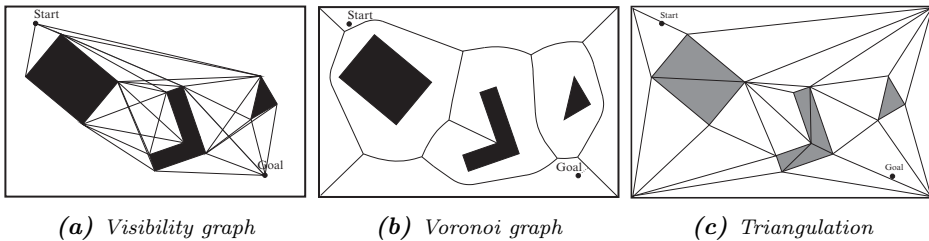


Figure 5.4: Map representations using roadmap methods [145].

- **Artificial potential field (APF):** the target environment is represented as a virtual force field where a gravitational force exists between the starting and target points, whereas repulsive forces are generated between the vehicle and obstacles [145]. It can be seen visually as a terrain with different heights where the target point is located in the lower area while the starting point is located in a higher area, and the obstacles would be high hills (Figure 5.5). The geometric curve resulting from applying the negative gradient of the potential field is the path to be followed by the vehicle. Returning to the example of the terrain, the path would be the one generated by dropping the ball down the hill from the starting point to the target point. It is a simple method but has the following disadvantages: (i) it is likely that the vehicle could become trapped at a local minima point, and (ii) the resulting trajectory can be affected by oscillations when passing between areas equally spaced from two obstacles. In practice, this method is more focused on avoiding obstacles than finding an optimal solution.

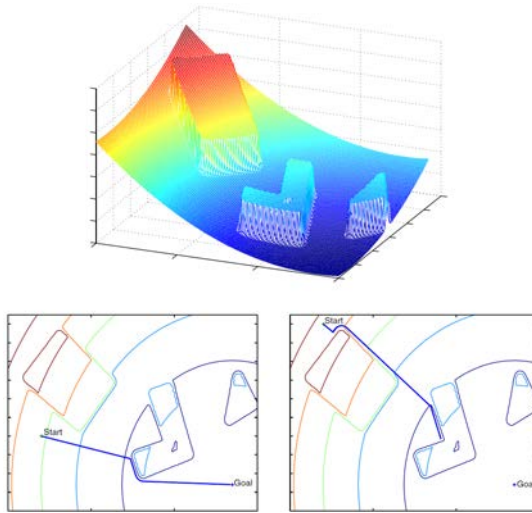


Figure 5.5: Map representation using artificial potential field [145].

- Path planning algorithms

Once the environmental map is generated, we can define the points along the path, also known as waypoints, which are typically chosen after applying any GPP algorithm on the environmental map [147]. Some of the best-known GPP algorithms are the following (Figure 5.6):

- **Graph search algorithms (graph traversal):** the best-known algorithms in this field are Dijkstra's algorithm, A^* , and D^* . They are typically based on weighted graphs, where the aim is to find the path having the smallest cost to reach the target point starting from the specific initial point. The cost is measured according to relevant variables such as shortest time, distance travelled, or less energy consumed.
- **Sampling-based algorithms:** one example of this is the rapidly-exploring random tree (RRT). An RRT algorithm builds a space-filling tree that includes possible movements to connect the initial and target locations. The tree is incrementally built from randomly drawn samples of the free space, and it is likely to grow towards isolated areas.
- **The APF method:** it can be used as both a map representation method or a path planning algorithm if more information of the environment is provided. However, the path obtained is often not optimal.
- **Evolutionary algorithms and particle swarm optimisation:** evolutionary algorithms use some mechanisms inspired by biological evolution to find approximate paths [148]. The most famous evolutionary algorithm is the genetic algorithm (GA). Similarly, swarm intelligence is inspired by artificial life research. The most famous swarm intelligence are particle swarm optimisation and ant colony optimisation (ACO): many possible trajectories are studied based on a series of update rules until the optimal trajectory to the target point is found.

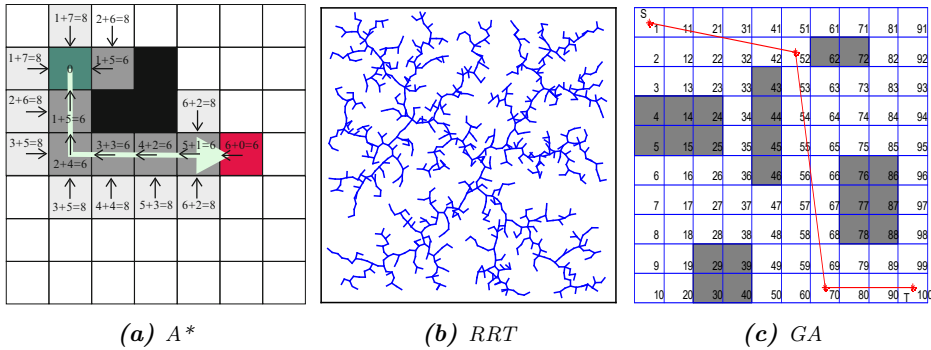


Figure 5.6: One example of each of the most well-known types of path planning algorithms: A^* (graph search), rapidly-exploring random tree (sampling-based), and genetic algorithm (evolutionary) [145, 148].

5.1.2.3. The travelling salesman problem

Every time we use a GPS navigation application (e.g., Google Maps) to go from our current location to our home, work, or favourite restaurant, we ask the underlying software to solve a specific problem for us. This software generates all feasible routes and orders them according to the shortest time to reach the destination.

This problem is commonly known as the travelling salesman problem (TSP) and is of great relevance in modern society (Figure 5.7) [149]. TSP tries to answer the following question:

Given a list of cities and the distances between them: what is the shortest possible route that visits each city exactly once and returns to the city of origin at the end?

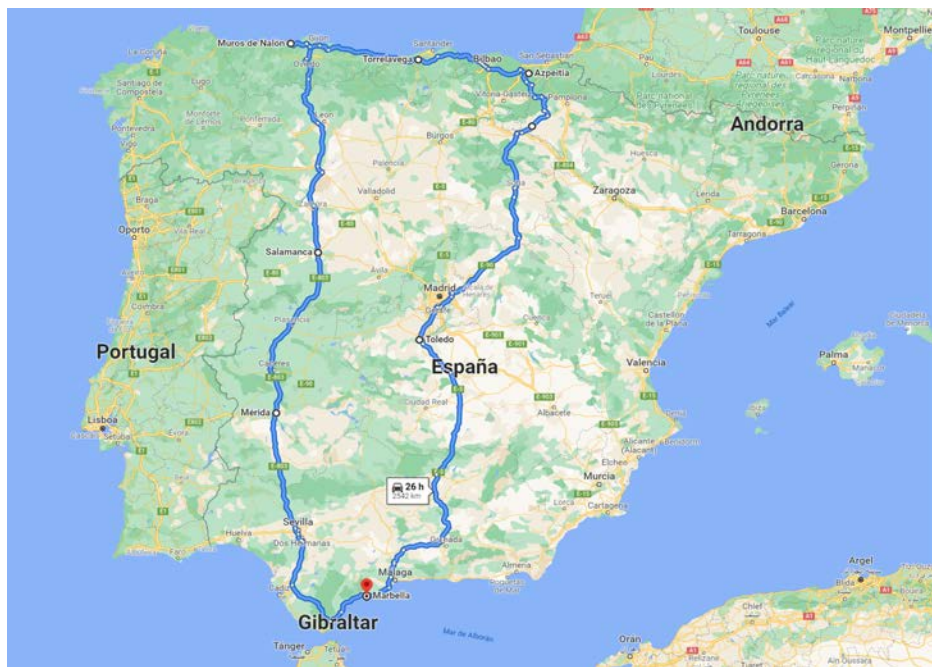


Figure 5.7: Illustration of the travelling salesman problem (TSP) using the following Spanish cities as input in Google Maps: Marbella, Mérida, Salamanca, Muros de Nalón, Torrelavega, Azpeitia, and Toledo.

TSP is a well-studied and famous problem in the area of computer science that belongs to the class of combinatorial problems known as NP-Hard, which means that it can not be solved in polynomial time [150].

The difficulty of solving the TSP problem is easily understood as the number of points in the trajectory increases. For example, the most straightforward solution would be to try all permutations—considering ordered combinations—and look for the best one based on a particular defined parameter such as time, length, energy, or even a combination of them. Therefore, the complexity of this particular problem is $O(N!)$, meaning that the time required to complete this analysis depends on the factorial of the N number of points or ‘cities’ to visit. Consequently, using the direct solution is impractical above even only 20 points. Current suboptimal solutions are found using heuristics or approximation algorithms, deriving good near-optimal results in a reasonable time [151].

As outlined throughout this section, path planning contributes to society in a wide range of fields, but it remains one of the most challenging problems today. In this chapter, the focus is on forest fire fighting using UAVs. Hence, the following section will present the most recent research on path planning using UAVs, which use some of the above-mentioned map representations and path planning algorithms.

5.2. Related work

There is a large body of research on UAV path planning, so we have considered those relevant to this work. The literature distinguishes three main research areas: obstacle avoidance, energy constraints, and area coverage.

5.2.1. Obstacle avoidance

Most research on UAV path planning aims to optimise the path between several points by avoiding obstacles. For example, the method proposed in [152] is capable of generating safe and flyable paths between two points on the sea in the presence of different threat environments (obstacles) using hybrid differential evolution and quantum-behaved particle swarm optimisation. The authors in [153] developed a sampling-based path planning method to generate collision-free paths in real time considering moving obstacles.

The algorithm presented in [154] considers both static and dynamic threats to generate a smooth path with low cost for UAVs based on ACO. The authors

in [155] implemented a UAV path planning strategy based on the multi-ACO algorithm for checking several control points considering obstacle avoidance in a large area. The dynamic path planning method introduced in [156] generates an optimal path in a known environment based on the A* algorithm for the UAV so that it can avoid both static and moving obstacles. The authors in [157] presented a robust and effective path planning strategy for UAVs based on the searching ability of the ACO algorithm.

These algorithms are useful for dynamic environments, e.g., surveillance flights over military areas and reconnaissance flights at low height in the presence of buildings and other obstacles. However, such solutions are not generally suited for the particular requirements of forest fire monitoring. This scenario can be considered as a static environment—no significant changes occur over time—and the only restriction on flight height is imposed by the competent authority.

5.2.2. Energy constraints

Research is also conducted considering energy consumption constraints for the UAV to achieve an efficient flight, either with the path determined beforehand or by generating it in real time. The proposal given in [158] focuses on developing an energy-aware path planning algorithm that minimises energy consumption for the UAV and other conditions such as coverage and resolution. The method proposed in [159] aims to assign energy-efficient trajectories for UAVs. It estimates the energy consumption resulting from basic UAV movements and considers discrete spaces to avoid collisions. The authors in [160] assessed the endurance flight of a solar-powered UAV, which was improved by optimising the path based on gravitational potential and controlling the flight height in real time. The novel approach proposed in [161] performs an energy-efficient inspection to minimise the overall energy consumption of the UAV flight.

These approaches addressed the coverage problem by using discrete spaces (subregions) that the UAV must visit. This solution is inefficient because it neglects that the UAV could have covered some locations from others nearby if the horizontal coordinate of the onboard camera is considered.

5.2.3. Area coverage

Coverage path planning algorithms aim to find optimal paths from which every point in the target area is visually covered. Very few path planning algorithms aim to guarantee significant or total visual coverage of the target area.

The work in [162] introduced a real-time path planning methodology for a single UAV based on limited energy and power consumption, which depends on the vehicle motion. It aims to maximise the spatial coverage of a given area by optimising the turning rates according to the maximum load factor of the UAV. The main advantage of this proposal is that it does not require adjusting the space discretisation beforehand. However, the weaknesses include the following: (i) the UAV coverage is limited to the vertical component below it, and (ii) some areas are not finally covered, one of which is of considerable size.

The algorithm proposed in [163] generates paths specially designed for UAV photogrammetry based on E-Spiral and E-BF algorithms. The method considers the camera characteristics, overlapping rates, and energy consumption. The main downsides are: (i) the discretisation of the target area is required, making the UAV fly following a path that sweeps the whole territory; and (ii) only vertical photography is considered. Moreover, the authors in [164] proposed addressing the coverage path planning problem for a UAV using approximate cellular decomposition and a wavefront expansion algorithm. It succeeds to cover the study area but, similarly to the previously presented approaches, it performs a prior grid-based area decomposition, so the UAV must fly over each centre inefficiently.

The proposal in [113] addressed the path planning problem using a new method based on siting multiple observers and an evolutionary algorithm. First, an iterative process removes redundant points according to how each particular viewshed value affects the area coverage. As a result, they obtained a set of locations that includes the fewest number of places providing maximum coverage. This method generates a near-optimal path covering the terrain, but it suffers from the following drawbacks: (i) it considers the restricted viewshed from each location in the territory that results in a loss of information in processing, and (ii) a grid-based decomposition method is required for the target area.

Unfortunately, few coverage path planning approaches focus on monitoring using UAVs for fire prevention purposes. For example, the authors in [165] introduced a model to monitor regions with zone occlusions due to vegetation through a two-steps approach to determine near-optimal flight paths. First, they arranged the waypoints using the Centroidal Voronoi Tessellation (CVT) method. Secondly, they generated Dubins paths using a clustered spiral-alternating algorithm. This model achieved to cover dense areas by considering more points for the path than typical grid-based methods. Nevertheless, it suffers from several limitations: (i) the decomposition of the target area following a CVT-based approach, which, although it seems to perform better than other grid-based works, is still inefficient as the drone must fly over every independent region; (ii) the consideration of restricted viewshed; (iii) the high cost of development that im-

plies a multi-UAV approach; and (iv) the unavailability of the Digital Surface Model of forest regions for most territories.

The authors in [166] presented a path planning approach for forest fire prevention based on two stages: local and global. At the local stage, they find the optimal path between every two target points by applying a Dubins path—based on the A* algorithm. At the global stage, the authors determine the visiting order of each target point using an improved ACO algorithm. This approach successfully calculates a near-optimal path that passes through every point in the terrain. However, this proposal has several weak points: (i) the waypoints are directly placed and only the path between them is generated, and (ii) each waypoint in the area is placed in a 2D space without considering its elevation or visibility, which are essential variables in real-world applications.

The above approaches, which are specific to the coverage path planning problem, suffer from several shortcomings:

- None of the proposed solutions so far considers the visibility from the initial set of locations for further calculations, e.g., the viewshed results from the starting and ending points. This data should be the input of the heuristic methods used to solve the path planning problem.
- Those alternatives using grid-based discretisation for computing the waypoints in a given area have the drawback of forcing the UAV to fly over each independent subregion. This solution neglects whether a particular subregion has already been visually covered from adjacent locations.
- Many algorithms consider restricted viewshed. It means that the terrain captured by the camera sensor is related to the area that falls within a preset circle of fixed radius just directly underneath the UAV. Therefore, the UAV may visit many redundant locations within the territory, despite the high resolution of the cameras currently available. These approaches are typically motivated by some performance constraints.
- All these approaches may be valid for small territories but hardly applicable in large and complex scenarios since they do not perform a thorough visibility analysis of the area.

In this work, we use the total viewshed analysis to solve the path planning problem for large regions of complex terrain. This type of analysis implies computing the viewshed from each point in the DEM with no visibility restrictions. Therefore, our approach does not require the application of a discretisation technique of the space beyond that provided by the DEM spatial resolution.

5.3. VPP: Visibility-based Path Planning

Unfortunately, forest fires have become more common, putting our ecosystems at risk. Millions of hectares of forest burn every year, bringing substantial costs to the countries affected and damaging the environment [167]. Experts consider early detection the most important way to minimise the consequences of fires; therefore, governments have invested substantial resources to develop better systems in this area.

This section introduces a new heuristic called *Visibility-based Path Planning* (VPP) designed to address the problem of monitoring large regions of complex terrain based on a thorough visibility analysis. To this aim, VPP generates a safe and flyable path from which the UAV onboard camera visually covers most of the target terrain.

As a starting point, we assume that the UAV must fly over some initial locations of the given region for monitoring purposes (e.g., high environmental locations and battery recharging points). From this initial assumption, the goal is to maximise the visual coverage of the entire area by optimising the number of points that will form the path. We achieve this by identifying the hidden locations of reduced visibility, usually called hidden areas, to monitor them during the flight and warn the local authorities in case of detecting fire hotspots.

Another consideration is that the UAV will fly following a straight line between every two target points and maintaining a steady height along the path. We do not consider obstacle avoidance as the flight height of the UAV is typically much higher than the height of trees found in any forest region. If the environment changes with respect to the initial model (e.g., installation of new power lines), the path can be adjusted to avoid these new static obstacles using GIS tools.

For the sake of clarity, throughout the following sections, we will use some figures showing partial results to explain the stages of the VPP heuristic. The selected study area comprises a forest region particularly affected by fires within the last ten years: the Montes de Malaga Natural Park (Figure 5.8) [168, 169, 170]. This area is located in the south of Spain and has significant environmental interest with extremely varied relief—which causes remarkable monitoring difficulties.

First, we will explain the types of waypoints that will form the path. It is followed by the description of the main structures required for the visibility analysis. Then, the workflow of VPP is presented, which consists of four stages. In the first stage, the visibility is analysed from the initial set of locations. Afterwards, in the second stage, the aim is to find new waypoints based on maximising visibility.

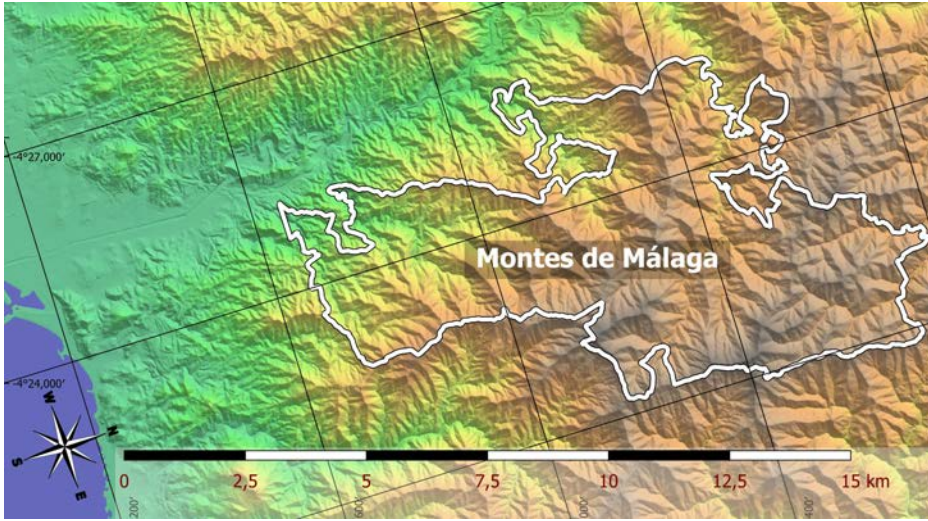


Figure 5.8: Area of the Montes de Malaga Natural Park (Malaga, Spain). The park boundaries are represented using a white line.

The third stage involves a new method to identify the remaining hidden areas to improve coverage. Finally, the fourth stage generates the final flight path by connecting the waypoints obtained from each of the previous stages.

5.3.1. Types of waypoints

A *path* is defined as a route between one place and another formed by connecting several points located in a territory. The selection of these points, called waypoints, is essential for the resulting path to provide maximum visibility during flight. Note that there are many possibilities for defining the waypoints and the shortest way to connect them, but each possibility should consider the viewshed from the initial locations of mandatory monitoring in the target region. We distinguish between two types of waypoints according to their arrangement and importance along the path:

- **Primary waypoints (W_p):** this group includes the initial locations of mandatory monitoring, as well as those found by the VPP heuristic. These points are connected using straight lines to form the path.

- **Secondary waypoints (W_s):** to this type belong those intermediate points placed on the straight line of travel between two consecutive primary waypoints from which the UAV can also capture the terrain (more detail in Section 5.4). The distance between two consecutive waypoints in W_s is adjusted so that the processing of the image captured at the first point completes before reaching the second.

Obtaining a suitable solution to the path planning problem based on maximising visibility is highly challenging due to the many variables influencing the result. Some examples are the UAV flight height and camera characteristics such as field of view and resolution. These variables significantly change the area covered from each of the waypoints in W_p and W_s . For example, it is possible that after getting a first approximation of the path, some waypoints set at the beginning of the process become redundant. It happens when some of the areas monitored from these points have already been covered from other nearby locations—which can lead to a challenging vicious circle. This issue states the importance of visibility analysis to reduce overlapping viewsheds in the path generation process.

5.3.2. Main data structures

The VPP heuristic requires different types of data structures at each of its stages, so these are defined below:

- **Digital Elevation Model (DEM):** 3D representation of the terrain with a size of $dim_y \times dim_x$, where each cell represents the elevation above sea level of the corresponding point of the ground. Cells are represented with i and j subscripts corresponding to x and y coordinates, respectively.
- **Singular Viewshed (SVS):** map storing the viewshed, i.e., the area visible from a particular point of view (POV), which represents any point/location of the DEM where we ‘place’ an observer. SVS is a unique matrix with the same size and grid as the DEM. Each cell, related to a point in the DEM, contains a Boolean value representing whether the corresponding location is visible from the POV. No visibility restrictions are considered for this computation, and, therefore, each line of sight is considered without distance constraints.
- **Point Set Cumulative Viewshed (PS-CVS):** map that accumulates the SVS results. Only the viewsheds from primary waypoints contribute to this structure.

- **Masked Total Viewshed (M-TVS):** map obtained from solving the total viewshed problem where specific points/areas are left out of the analysis. Instead of considering all the points in the DEM, we only find those providing maximum viewshed over no redundant areas. Therefore, we exclude already monitored areas from the viewshed computation [112].

5.3.3. Workflow of the VPP heuristic

Figure 5.9 shows the flowchart of the VPP heuristic with each stage represented. First, the VPP heuristic requires setting two initial design thresholds α and γ . Each of these thresholds is related to a target percentage of visual coverage (Cv) sought for the region of interest: α represents the minimum target percentage of area to be covered as a result of the process; γ indicates the minimum percentage increase in area coverage that must occur between two measurements. In general, low thresholds imply few waypoints and short but imprecise

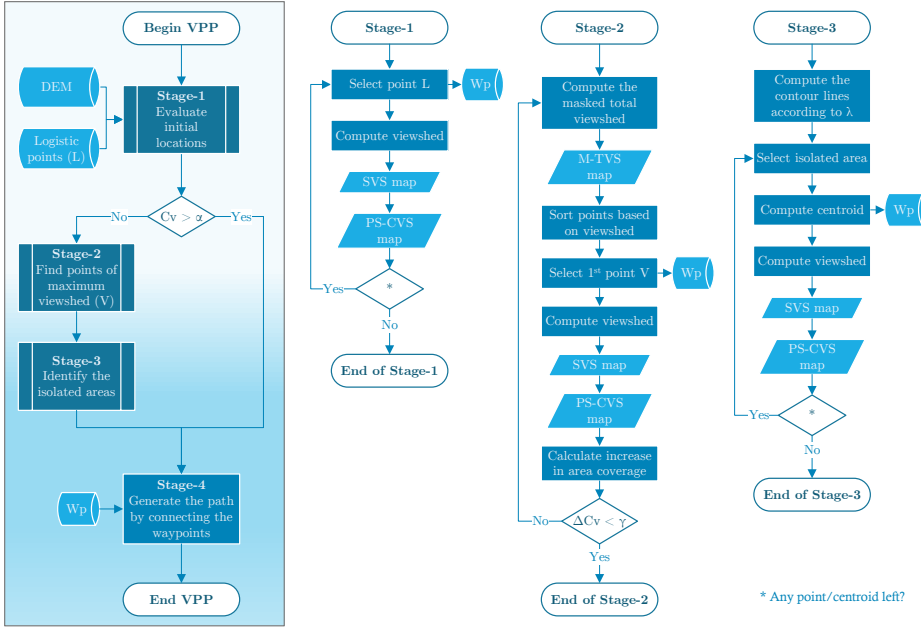


Figure 5.9: Flowchart of the VPP heuristic showing the main workflow on the left and the workflows within the first three stages on the right (the fourth stage is not broken down as it comprises only one step).

flight paths, whereas high thresholds increase the number of waypoints, turning the path into a tremendously expensive zigzag sweep.

In our case study, we set $\alpha = 90\%$ and $\gamma = 5\%$. It means, respectively, that we would accept a result of 90% of the area visually covered and a 5% increase in area coverage between two consecutive measurements, based on the requirements and difficulty of the terrain at hand. These thresholds may need to be adjusted for other environments depending on the goal and relief of the target region.

Each of the four stages of the VPP heuristic is detailed below.

5.3.4. Description of the four stages

5.3.4.1. Stage-1: evaluating the contribution to the coverage of the initial locations

For any path planner, the first step is to define a target point or set of target points on the ground the UAV must monitor. These locations are called *logistic points* (L), and among them are such obvious ones as the starting and ending points of the flight. We can also include some locations for recharging batteries, ensuring communications coverage, or effectively monitoring some areas of high environmental value. Note that these requirements may vary depending on the study's objective and the terrain's complexity, but the logistic points are always the first to be included in W_p .

Based on the initial conditions of our case study, the UAV has to fly over five logistics points, including three locations for monitoring critical natural areas. The viewshed map of each of these must be computed and merged to obtain a first approximation of the terrain coverage (Cv).¹ The SVS map is obtained for each $P_i \in W_p, i = 1...|W_p|$ so that, afterwards, they are all added up to form the PS-CVS map. The generated PS-CVS map represents the amount of terrain covered and, at this early stage, provides a global overview of the coverage results obtained from considering the logistic points at the beginning of the process.

Once the viewshed from each logistic point has been computed, the focus is on the uncovered terrain. To show these areas, we will use the Cumulative Hidden Areas (CHAR) map—the complementary structure of the PS-CVS map. The CHAR map highlights the uncovered areas, allowing us to find and define the most remote ones (Figure 5.10). Although we have analysed five initial logistic locations, this map shows that many hidden areas of difficult access are still

¹Hereafter, the observer's height (flight height) is set to $h = 30m$.

widely scattered and diverse in shape, not forming simply connected topological spaces.

At this early stage, connecting the points to generate the path, which is a problem similar to the TSP, is impractical as we will not obtain good coverage results. Fortunately, adding new locations to W_p will increase the visual coverage over the target region in the following stages, ensuring successful monitoring.



Figure 5.10: Cumulative Hidden Areas (CHAR) map generated at the end of Stage-1. It shows the remaining uncovered area (purple layer) after analysing the viewsheds from the initial five logistic points (L) in W_p . The white line represents the boundaries of the Montes de Malaga Natural Park.

5.3.4.2. Stage-2: finding new points of maximum visibility

Finding the following locations to be included in W_p is vital to increase coverage. For instance, the authors in [112] addressed a similar problem and concluded that it is already possible to guarantee a visual coverage above 90% with only two or three points considering a similar territory of 50 km^2 with complex orography.

In this case, we must ensure an appropriate balance between the number of points to be included and their locations to minimise path length. This consideration will allow implementation in a real environment at relatively low cost. Following this line, we naturally infer that the points providing maximum view-

shed are good choices at this stage. These points are called *POVs of maximum viewshed* (V), and we find them by solving the total viewshed problem using the M-TVS map.

In the first iteration, we compute the viewshed from each point in the DEM and accumulate their results in the M-TVS map (same number of cells as in the DEM). Each cell in this map relates to a specific location on the terrain and contains its viewshed value (measured in, e.g., km^2). The point with the highest viewshed value is included in W_p . In the next iteration, the area visually covered from that point is excluded from the analysis; the new POVs of maximum viewshed will be the ones covering the largest uncovered area. Including these points progressively in W_p gradually increases the area covered so that it may at some point exceed the α threshold. We will stop including POVs of maximum viewshed when this does not significantly increase the coverage result, i.e., $\Delta Cv < \gamma$ calculated from the difference between two consecutive PS-CVS maps.

In our case study, the inclusion of the two best POVs of maximum viewshed to W_p has proved to be the best strategy, guaranteeing a visual coverage above 80% for our region of interest (Figure 5.11), but there are still many hidden areas.



Figure 5.11: Cumulative Hidden Areas (CHAR) map generated at the end of Stage-2. It shows the remaining uncovered area (purple layer) after having analysed the viewshed from the initial five logistic points (L) and two points of maximum viewshed (V) in W_p . The white line represents the boundaries of the Montes de Malaga Natural Park.

The advantage is that the hidden areas at this stage begin to form small regions in objectively isolated valleys and are therefore called *isolated areas*. Due to their visual isolation, only the UAV flight over them guarantees adequate monitoring. The next step, covered in the third stage, is to identify these isolated areas to include them or points close to them in W_p .

Solving the TSP at this stage is discouraged for the same reason as in the first stage, although better results would be obtained.

5.3.4.3. Stage-3: identifying isolated areas

Some path planning strategies, such as in [113] and [165], ensure that the UAV reaches every isolated area of the target region following a greedy approach: they force the UAV to fly over almost any point on the ground. Conversely, VPP provides a better solution: identify visually isolated areas for the UAV to fly over their centroids, avoiding the need to sweep the whole region.

We propose two methods to characterise the isolated areas according to the perspective from which it is analysed:

- From the observer's viewshed, we define an area as isolated if it is not visible from a set of POVs of maximum viewshed.
- From the viewshed of the target area, we consider an area isolated if the area of terrain visible from its centre—similar to placing an observer there—is less than a fixed parameter λ . This result is achieved by using contour lines based on this parameter.

In this work, isolated areas are identified using the latter method, as it is the most objective. We chose the parameter $\lambda = 1 \text{ km}^2$ after having carried out several optimisation tests so that the resulting areas are more delimited in most cases, constituting simply connected spaces that are easily identifiable by their centroids (Figure 5.12). These centroids are included in W_p and, finally, the PS-CVS map is generated to check whether the percentage of terrain coverage is above the α threshold. If this condition is met, we can proceed to the final stage.



Figure 5.12: Isolated areas identified using contour lines with a viewshed area less than $\lambda = 1 \text{ km}^2$ at the end of Stage-3. Their centroids are marked with green pins only within the Montes de Malaga Natural Park. The white line represents the park boundaries.

5.3.4.4. Stage-4: generating the flight path

The last stage of the VPP heuristic consists in generating the path by connecting the primary waypoints included in the final set W_p . After having completed the previous three stages, the set of waypoints W_p contains:

- 5 logistic locations, including the take-off and landing locations and three easily accessible locations of high environmental value that we can use as recharging points.
- 2 POVs of maximum viewshed obtained after computing the M-TVS map.
- 12 centroids of the isolated areas.

Figure 5.13 shows the path generated after connecting all the primary waypoints (similarly to solving the TSP) in the area of the Montes de Malaga Natural Park (Malaga, Spain). This 34 km-long path offers all-round visibility during the flight and, therefore, the UAV can successfully monitor the target area using its onboard camera.



Figure 5.13: Resulting flight path (red line) generated by VPP in the Montes de Málaga Natural Park area. The area covered is represented using a white layer over the terrain model. The white line represents the park boundaries.

Note that the PS-CVS map calculated here may differ from a real flight because a 360-degree onboard camera has been considered. However, it is more common to use cameras with a narrower field of view, which affects the resulting coverage.

Here, we have discussed a new way of addressing the coverage path planning problem based on a thorough visibility analysis from which new points are included in the path. These points correspond to primary waypoints: logistic points, POVs of maximum viewshed, and the centroids of the isolated areas. Nevertheless, there is still a fundamental fact to be addressed, which is related to obtaining the best camera direction for each waypoint to maximise the ground covered. This issue is further discussed below.

5.4. Viewshed-rejection method for setting each direction of the onboard camera

The previous section presented the VPP heuristic for generating flight paths based on a thorough visibility analysis. That analysis considered full visibility range from any point, as if a 360-degree camera were there. However, in real life, the use of this type of camera in UAVs is not a common practice, and therefore the camera's *Angle of View* (AOV) comes into play.

In photography, the camera's AOV describes the angular extent of a given scene captured by a camera, measured horizontally, vertically, or diagonally. This variable can be seen as the maximum number of consecutive sectors (i.e., angles) that the camera can capture. This is why the camera's AOV highly affects the choice of the direction of the camera—the middle sector of the set of consecutive sectors—for each waypoint of the flight, which is called *Center of Interest* (COI).

Here, we propose a new methodology to find the best COI for each waypoint of the path. Each COI is set according to the direction of maximum visibility, obtained after rejecting those directions of the camera pointing at areas already covered from preceding waypoints along the path. As a result, visual overlapping is avoided, increasing the terrain covered during the flight.

Firstly, the data structures required to perform this simulation are described, followed by a detailed description of the workflow of the algorithm. An iteration of this workflow involves (i) processing the viewshed from the target waypoint in the DEM using a memory optimisation technique, (ii) obtaining the best COI for the target waypoint, (iii) simulating the coverage result, and (iv) repeating the process on the next waypoint.

5.4.1. Photographs taken by the onboard camera

The flight path generated by VPP at the end of Stage-4 consisted of 19 primary waypoints. The UAV will take a photograph from each of these primary waypoints but, in addition, more photographs can be taken on the way from one waypoint to another, e.g., every 150 m approximately, with these new points being the secondary waypoints (W_s). We chose this distance to ensure that the processing of each image completes before reaching the following location.

This method leads to a total of 235 waypoints along the flight path (between W_p and W_s), where the camera's AOV is crucial for maximising the coverage of the region.

5.4.2. Additional data structures

The following structures are required for this approach:

- ***RS*** stands for ring-sectors and is a look-up table where each row corresponds to a particular sector (from 0° to $n_s = 360^\circ$), and each column represents the distance to the given reference location. The first column ($d = 0$) of each row stores the number of set values (visible locations) for the corresponding sector. The stored value for a given sector s and distance d represents the state of the location: whether it is visible from the reference location.
- ***AOV*** is a Boolean array with 360 cells (one per degree) that represents the camera's angle of view, i.e., the sectors captured by the UAV onboard camera from a specific waypoint of the path.
- ***CC-SVS*** is the Camera Captured Singular Viewshed and contains the viewshed result from a given waypoint of the path in a similar way as the *SVS* structure but only analysing the flagged sectors in the *AOV* structure—sectors captured by the onboard camera.
- ***PL-CVS*** is the Point Line Cumulative Viewshed and accumulates each viewshed result stored in *CC-SVS* as the UAV flies along the path. This structure also works as a Boolean mask that contains a set value in each cell corresponding to a location already covered by the UAV.

5.4.3. Finding the best COI for each waypoint

Computing the best COI for each waypoint of the path is key to maximising visibility during flight. Once the flight has started, we must avoid pointing the camera in a direction where previous locations have already covered a high proportion of the terrain.

We define a lookup table called *RS* that contains the number of covered points from a given waypoint and for each sector. *RS* has as many rows as sectors (from $s = 0^\circ$ to $n_s = 360^\circ$) and columns as the maximum possible distance between two points (measured in cells of the DEM). Figure 5.14 shows how the *RS* structure is filled for a single sector: given a reference waypoint and a target location, if the target location is visible from the waypoint and it has not been covered yet—checking the Boolean value inside *PL-CVS*—its corresponding cell in *RS* is flagged. The total number of flagged cells in a particular row is stored in the first column of the *RS* structure, which also represents the weight of the sector.

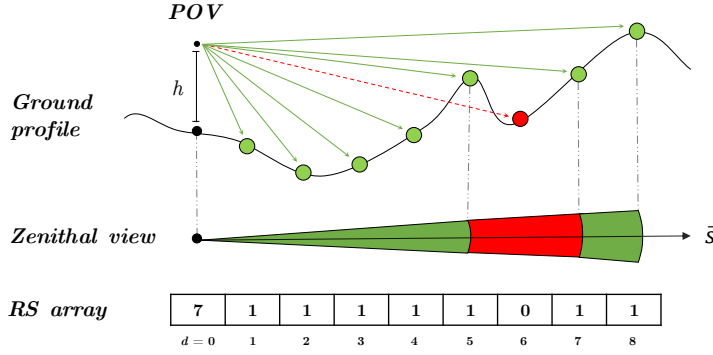


Figure 5.14: Viewshed computation from a reference point of view (POV) of h height and only one sector s , for the sake of clarity, showing the ground profile, zenithal view, and lookup table ring-sectors (RS). The visible points in the ground profile are represented in green, whereas the red points correspond to hidden locations. The zenithal view shows the ring-sector shape obtained from the ground profile. The RS structure is filled following the results shown in the ground profile.

Finally, a fixed number of consecutive sectors (n_{aov}) will be captured by the camera at each waypoint of the path, in a process represented using the AOV structure. AOV works as a Boolean mask: it enables visibility processing for some sectors, whereas disabling processing for the remaining ones (not captured by the camera). Therefore, only those consecutive sectors that are flagged will contribute to the coverage result, where the COI is the middle sector.

The best set of consecutive sectors for each waypoint in AOV is chosen based on their weights and seeking to maximise the terrain coverage by avoiding visual overlapping. The weight of each sector is measured as the total number of points visible in the sector and not yet covered (previously stored in the first column of the RS array corresponding to $d = 0$). In the end, the set of consecutive sectors holding the highest overall weight will be flagged in AOV for each waypoint. The overall weight of each possible grouping of n_{aov} consecutive sectors is obtained by following an iterative process: we add the weights from $s = 0^\circ$ to $s = n_{aov}$, then from $s = 1^\circ$ to $s = n_{aov} + 1^\circ$, etc., until the highest overall weight is found.

The last step involves computing the CC-SVS and PL-CVS maps for each target waypoint sequentially to obtain the individual and accumulated terrain coverage results, respectively.

5.4.4. Workflow for simulating the camera behaviour during the flight

Once all necessary data structures have been defined, we can present the general workflow for any path and DEM to simulate the camera behaviour during the flight:

1. For each waypoint in the flight path, do:
 - a) Fill *RS* based on the current visibility and the *PL-CVS* map.
 - b) Fill *AOV* according to the set of consecutive sectors with the highest overall weight.
 - c) For each point in the DEM, do:
 - 1) Compute the angular difference (A_d) and Euclidean distance (E_d) between the waypoint and the point.
 - 2) Index the *AOV* Boolean array using A_d to check whether the camera captures the current sector from the waypoint.
 - 3) If this condition is met, do:
 - a' Index the *RS* look-up table (using A_d as row index and E_d as column index) to retrieve the state of the target point, i.e., whether it is visible from the waypoint.
 - b' If so, the corresponding cell is flagged in the *CC-SVS* map.
 - d) Accumulate the result in the *PL-CVS* map.

For the sake of clarity, four iterations of the proposed methodology are shown in Figure 5.15 as a basis for explanation.² From the input DEM and the first waypoint P_1 , we can fill the *RS* and *AOV* structures and compute the first *CC-SVS* map (Figure 5.15a). At the beginning of the process, the *CC-SVS* map coincides with the *PL-CVS* map as there are no previous results (Figure 5.15b). Then, from the data stored in *PL-CVS* and considering again the *DEM*, *AOV*, and *RS* structures, we can compute the *CC-SVS* for P_2 (Figure 5.15c). After having processed the first two waypoints, the *PL-CVS* map is as shown in Figure 5.15d. Finally, repeating these steps will produce the *CC-SVS* maps for P_3 and P_4 (Figures 5.15e and 5.15g) along with the data in the *PL-CVS* maps (Figures 5.15f and 5.15h).

²The onboard camera's angle of view and the height of the UAV flight are set to 84° and $h = 30\text{ m}$, respectively.

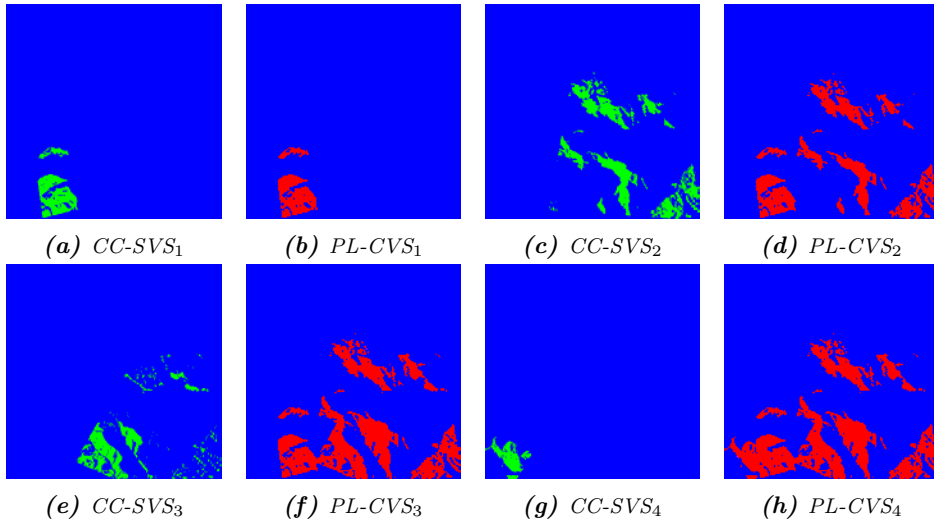


Figure 5.15: Simplified sequence of operations showing the Camera Captured Singular Viewshed (CC-SVS) and Point Line Cumulative Viewshed (PL-CVS) maps obtained after simulating the UAV flight over the first four waypoints of the path in the area of the Montes de Malaga Natural Park. The subscripts refer to the waypoints of the path, which also coincide with the main loop iterations.

With the presented method, we avoid overlapping viewsheds between the waypoints included in the flight path, thus maximising the terrain covered by the onboard camera throughout the flight.

5.5. Experiments and results

This section evaluates the performance of VPP based on the terrain coverage captured by the UAV onboard camera considering three scenarios: two flight simulations over two complex terrain regions of 48 and 202 km^2 and a real flight test over a reduced area of 1 km^2 .

Firstly, the experimental setup is described at the beginning of this section. Then, we present the coverage results obtained from the two UAV flight simulations and the real flight. Finally, we discuss the contributions of the results achieved using the VPP heuristic.

5.5.1. Experimental setup

5.5.1.1. Systems and algorithms

For the simulations, we used a system with Ubuntu 16.04.5 LTS and an Intel(R) Xeon(R) CPU E5-2698 v3 @2.30GHz with 16 cores (32 threads) and 256GB DDR4 RAM.

To connect the waypoints of each flight path, we solved the TSP using the Network Analyst extension from ArcGIS [171]. On the one hand, the design of this path has not considered restricted visibility, as current camera technology allows high-definition image processing with minimal loss of definition. On the other hand, the obstacle avoidance problem has not been considered for generating each path because there will be no obstacles between two locations if we establish a sufficiently high flight height.

5.5.1.2. Key features of the UAV

We took as a reference for the simulation the characteristics of the well-known DJI Phantom 4 Pro drone [172]. Accordingly, the case study onboard camera used is the same as the one installed in that UAV model. We set the camera's AOV parameter to 84° and the flight height to $h = 30\text{ m}$.

5.5.1.3. Memory optimisation

Given a reference location, the viewshed computation involves checking the elevation data in the DEM, resulting in many cache misses in the system. This problem is caused by a lack of locality in memory that degrades performance.

To deal with this performance issue, we used a modified version of the sDEM algorithm described in [173] to obtain the viewshed result from each point.³ Locality increases using this procedure by performing sequential memory accesses in processing.

³The run-time of the viewshed computation can be found in Chapter 4.

5.5.1.4. Main characteristics of the target areas

Tables 5.1 and 5.2 present the UTM coordinates and elevation statistics of the case study areas, respectively. The values in the latter table reflect the abrupt orography of the regions studied according to the high value of the standard deviation of the elevation compared to the mean.

Table 5.1: UTM coordinates of the three regions used in the experiments.

Target area				UTM		
ID	Name	DEM (cell) size	Area	Zone ^a	Easting	Northing
1	Montes de Malaga Natural Park	2000x2000 (10m)	48 km ²	30S	0370000 mE	4090000 mN
2	Sierra de las Nieves National Park	2500x2500 (10m)	202 km ²	30S	0310000 mE	4075000 mN
3	Surroundings of Montes de Malaga		1 km ²	30S	0379430 mE	4069400 mN

^aLatitude band designator.

Table 5.2: Elevation statistics of the three regions used in the experiments.

Target area				Elevation statistics (m)			
ID	Name	DEM (cell) size	Area	Min.	Max.	Range	Mean \pm SD
1	Montes de Malaga Natural Park	2000x2000 (10m)	48 km ²	76	1041	965	561 \pm 505
2	Sierra de las Nieves National Park	2500x2500 (10m)	202 km ²	93	1917	1824	848 \pm 764
3	Surroundings of Montes de Malaga		1 km ²	177	454	277	296 \pm 61

5.5.2. Flight simulation 1: Montes de Malaga

The first experiment involves a flight simulation over the Montes de Malaga Natural Park using a single UAV for monitoring. This park was previously used in Sections 5.3 and 5.4 to explain the stages needed to generate the path and set the onboard camera behaviour, respectively. Declared a protected natural space in 1989, this park near Malaga city (Spain) has 4,800 hectares of complex terrain with areas from 80 to just over 1,000 m above sea level. These areas host different protected flora and wildlife species and many water resources. These facts make the Montes de Malaga Natural Park a suitable scenario for fire prevention analysis and evaluation.

Once we have generated the path using VPP and set the COI for each way-point, we can simulate the UAV flight and obtain the final terrain coverage value. Figure 5.16 shows the area captured by the UAV onboard camera after flying



along the path and travelling it counterclockwise. The colour scale indicates the terrain covered and the distance from the starting point where the photograph was taken. For example, if the colour of an area on the map is blue, the camera captured it within the first kilometres of the flight. Conversely, the red colour indicates that the UAV took the photograph near the end of the path [174]. In this case, the rightmost logistic point is considered the starting point of the flight path.

As a result, VPP generated a 34 km-long path that includes 235 waypoints located every 150 m along the path: 5 logistic locations, 2 POVs of maximum viewshed, 12 centroids of isolated areas, and 216 secondary waypoints. We obtained that the flight of a single UAV, following the path generated by VPP and considering each COI, manages to capture 98.7% of the natural park area—48 km² of complex terrain—during the flight.

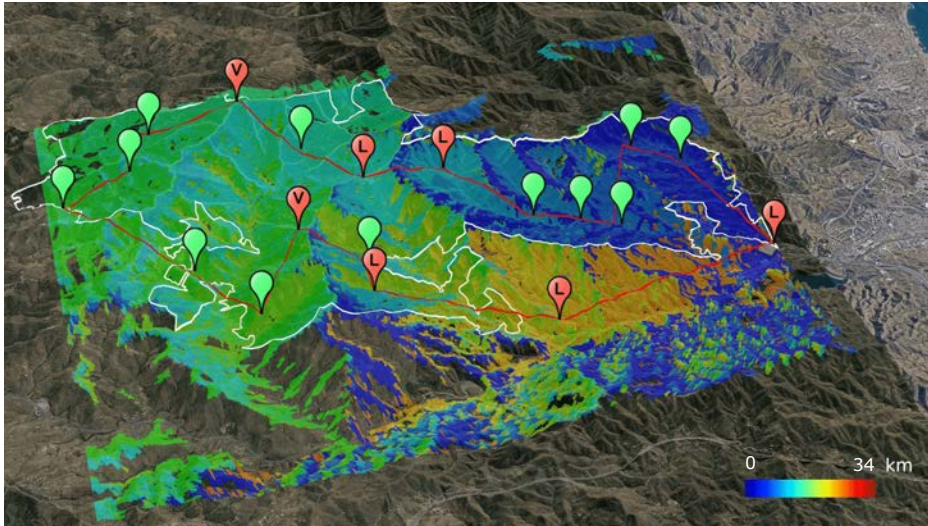


Figure 5.16: Final Point Line Cumulative Viewshed (PL-CVS) map of the Montes de Malaga Natural Park (Malaga, Spain). The red line represents the path generated by VPP, and the white line indicates the park boundaries. The UAV travels the path counterclockwise and takes off from the rightmost logistic point. The colour scale indicates the terrain captured by the UAV onboard camera and the distance from the starting point where the photograph was taken. The UAV covers 98.7% of the 48 km² region from the 34 km-long path.

5.5.3. Flight simulation 2: Sierra de las Nieves

The second experiment seeks to achieve the same objective as the previous one but over an even larger area: the Sierra de las Nieves National Park. It is a protected natural area of 22,979 hectares located in the northwest of Malaga province, with the highest peak reaching up to 2,000 *m* in height. This area hosts approximately 1,500 types of plants, many of them protected because they grow in only a few regions of Spain. These features make this park another good scenario to test VPP for fire prevention.

Figure 5.17 shows the terrain coverage result obtained by simulating the flight over the case study area. In this case, VPP generated a 79 *km*-long path including 566 waypoints (located every 150 *m*): 2 logistic locations, 5 POVs of maximum viewshed, 19 centroids of isolated areas, and 540 secondary waypoints. As a result, the UAV covers approximately 94.5% of the Sierra de las Nieves National Park. Additionally, we also simulated a flight with $h = 50$ *m* and the camera's AOV at 360°, enabling the UAV to cover up to 97.8% of the same area.

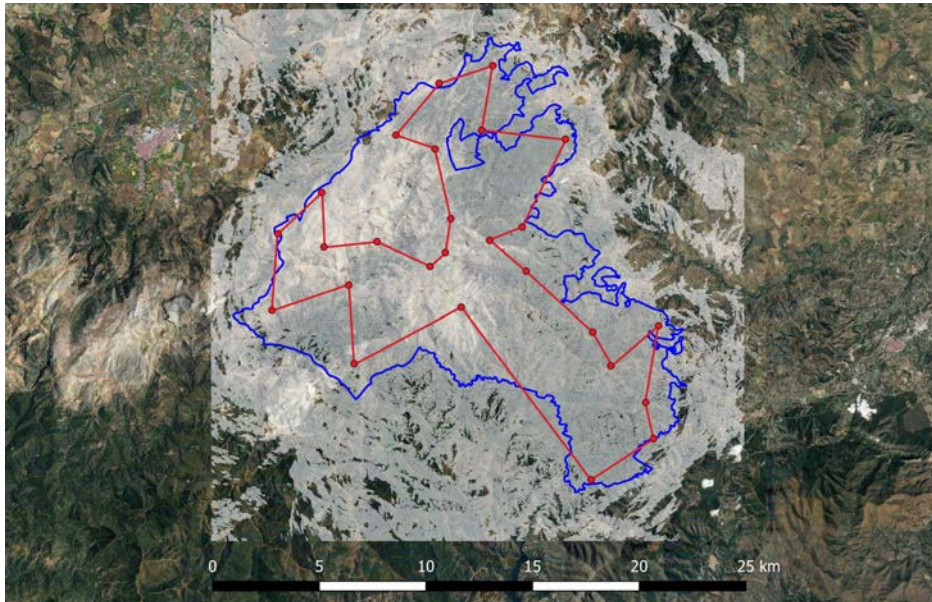


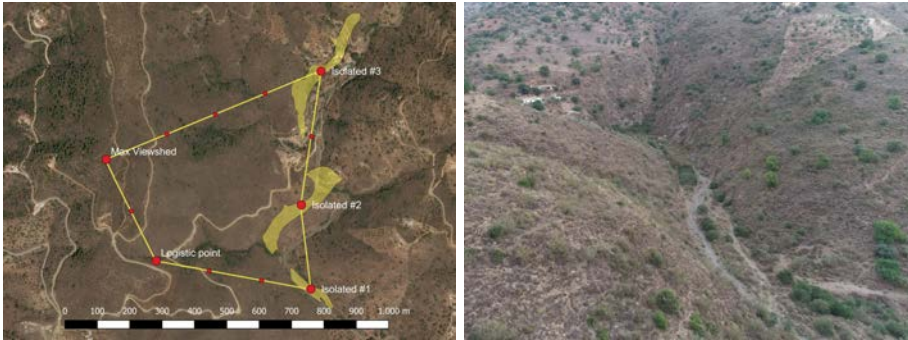
Figure 5.17: Flight path (red line) generated by VPP and terrain captured by the UAV onboard camera (white layer) in the Sierra de las Nieves National Park area. The blue line represents the park boundaries. The UAV covers 94.5% of the 202 km^2 region from the 79 *km*-long path.

5.5.4. Real flight test: surroundings of Montes de Malaga

Finally, our last experiment consists in a real flight of the UAV over a reduced extension of land located in the surroundings of the Montes de Malaga Natural Park (San Anton mountain). This scenario comprises only an area of 1 km^2 due to the impossibility of flying the UAV in larger regions since it is a commercial device whose use falls within the scope of the limited aerial regulations established by the Spanish Government.

Figure 5.18a shows the distribution of the 12 waypoints included in the 4 km-long flight path: 1 logistic location, 1 POV of maximum viewshed, 3 centroids of isolated areas, and 7 secondary waypoints, located every 150 m along the path. Figure 5.18b displays the photograph captured by the UAV from the waypoint located above the first isolated area [175].

The video recorded during the flight illustrates the high visibility obtained from the path generated by VPP, although the flight had to be interrupted due to the presence of new buildings below the path that did not appear on the satellite images. According to current aerial restrictions, it is forbidden to fly over private constructions. This problem has also demonstrated the importance of analysing the terrain in situ to discover potential drawbacks.



(a) Waypoints of the 4 km-long flight path (b) Capture of the first isolated area [175]

Figure 5.18: Real flight test over a small area located in the surroundings of the Montes de Malaga Natural Park.

5.5.5. Analysis of the results

Path planning and visual coverage are still challenging problems as they depend to a large extent on the complexity and diversification of terrains. To deal with this, VPP includes a comprehensive viewshed analysis in the earliest stages of path planning so that many points of maximum viewshed and isolated areas of difficult access are discovered.

As a result, we effectively monitor the target regions from the generated flight paths, as shown in the three experiments presented: two simulations and a real flight test. These three scenarios comprise large and varied regions with many remote/isolated locations where monitoring is an almost impossible task to be performed by observers siting on the ground. However, we could overcome this issue using a single UAV flying over each region following the generated path, achieving an extensive visual coverage of the terrain.

Together with the relevant authorities, we are working on transferring these results to real environments to improve fire prevention in forests. This implementation would significantly improve the monitoring activity of challenging areas, which would reduce firefighter response times in the event of a fire and thus prevent a catastrophe or minimise the damage caused.

6 Conclusions and Future Work

This chapter summarises the findings of this thesis. Section 6.1 presents the conclusions drawn from the work previously explained and the publications derived from it. Section 6.2 focuses on the future lines of research that would be the natural continuation of this thesis.

6.1. Conclusions

Spatial analysis has become increasingly relevant since the invention of the Internet because of the vast amount of spatial information available year after year. Furthermore, the improvement in the computational capacity of the systems used today has also contributed to this fact. Throughout this thesis, we tackled three spatial problems of paramount importance: latent fingerprint identification, total viewshed computation, and path planning.

Chapter 3 presented a novel methodology called *Asynchronous processing for Latent Fingerprint Identification* (ALFI) for heterogeneous CPU-GPU systems. ALFI efficiently overlaps and synchronises two tasks running on different devices. The first task, related to the data processing on the device, involves preprocessing and finding matching minutiae pairs, computing each minutia quality, and obtaining clusters of minutiae pairs. The second task comprises the multi-threaded final evaluation stage performed on the host that evaluates the clusters of minutiae pairs and returns the possible matched fingerprints. The novel strategy applied to the data processing on the device takes advantage of the intrinsic parallelism of the latent identification process. Each CUDA thread processes a specific minutia

from the batch of fingerprint impressions and compares it with each minutia from the latent fingerprint. Thus, ALFI reduces idle times in host and device units, achieving faster similarity results between latent and fingerprint impressions.

ALFI has been tested on Linux and Windows operating systems using three CPU-GPU pair systems. We used the well-known identification databases NIST SD27, NIST SD14, and NIST SD4 to test the accuracy of the proposed algorithm in latent fingerprint identification. Additionally, we used the FVC 2002, FVC 2004, and FVC 2006 verification databases to test the verification performance. The experiments proved that ALFI outperforms the state-of-the-art DMC algorithm, achieving a speed-up of 22x on average and maintaining accuracy results within the same range. In particular, considering the best-studied case, ALFI yields a speed-up of 44.7x using a background database holding 387,243 fingerprints.

To the best of our knowledge, ALFI is the first methodology for latent fingerprint identification specifically designed to exploit all the hardware resources of heterogeneous CPU-GPU systems. The negative point is that this algorithm is not being used in current criminal investigations due to problems related to the patent of the source code. However, we hope that this issue will be solved soon so that law enforcement authorities can use ALFI to identify individuals involved in real crimes.

Chapter 3 has been developed based on the following publications:

Acelerando la comparación de huellas dactilares basadas en agrupaciones deformables de minucias

A.J. Sanchez-Fernandez, L.F. Romero, and S. Tabik

In *XXIX Jornadas de Paralelismo (Jornadas SARTECO)*, Teruel, Spain, September 2018

[82] A First Step to Accelerating Fingerprint Matching based on Deformable Minutiae Clustering

A.J. Sanchez-Fernandez, L.F. Romero, S. Tabik, M.A. Medina-Pérez, and F. Herrera

In *18th Conference of the Spanish Association for Artificial Intelligence (CAEPIA 2018)*, Granada, Spain, October 2018

Published in: *Advances in Artificial Intelligence*. Book Series: *Lecture Notes in Computer Science*, vol. 11160, pp. 361–371, 2018

[176] Asynchronous Processing for Latent Fingerprint Identification on Heterogeneous CPU-GPU Systems

A.J. Sanchez-Fernandez, L.F. Romero, D. Peralta, M.A. Medina-Pérez, Y. Saeys, F. Herrera, and S. Tabik

IEEE Access, vol. 8, pp. 124236–124253, 2020

Chapter 4 introduced a new methodology called *skewed Digital Elevation Model* (sDEM) to accelerate terrain surface analysis. In particular, we chose the total viewshed problem as a case study to assess the performance of sDEM, which is developed from scratch and differs from state-of-the-art methods by adding a preprocessing stage.

sDEM applies a complete data relocation of the DEM before starting the total viewshed computation process, hence increasing the performance of the memory accesses. The proposed data relocation technique makes adjacent points in the terrain stored contiguously in memory but not based on the regular north-south and east-west directions, but considering the angular directions from 0 to 180°. Because of this, computing the visibility map of any terrain is much less computationally costly using sDEM than with the approaches reported in the literature.

We proposed different versions of sDEM for single-core, multi-core, single-GPU, and multi-GPU platforms, along with intensive performance studies considering the current GIS software and the best algorithm reported in the literature. sDEM has been tested on Windows and Linux operating systems using two different systems and three DEMs of up to 64 million points from the Montes de Malaga Natural Park (Malaga, Spain).

Our implementations have performed better than the most used GIS software regarding the multiple viewshed computation. Moreover, sDEM vastly outperforms the state-of-the-art algorithm according to the speed-up and throughput results for the three evaluated DEMs in the total viewshed computation. In fact, our approach accelerates this particular computation up to 827.3x for the best-studied case with respect to the baseline single-threaded implementation on a DEM formed by 16 million points.

Given the above, sDEM opens the door to take full advantage of multi-GPU systems to optimise many algorithms for which they had never been considered, typically due to their irregularity and low efficiency, as was the case with the total viewshed problem.

Chapter 4 has been elaborated on the basis of the following publications:

Reorganización de matrices en algoritmos de barrido radial sobre Modelos Digitales del Terreno

A.J. Sanchez-Fernandez, L.F. Romero, S. Tabik, and G. Bandera

In *XXX Jornadas de Paralelismo (Jornadas SARTECO)*, Cáceres, Spain, September 2019

[173] A data relocation approach for terrain surface analysis on multi-GPU systems: a case study on the total viewshed problem

A.J. Sanchez-Fernandez, L.F. Romero, G. Bandera, and S. Tabik

International Journal of Geographical Information Science, pp. 1–21, 2020

Chapter 5 presented a new heuristic called *Visibility-based Path Planning* (VPP) that offers a different solution to the path planning problem based on the exploitation of visibility data, where we have chosen forest fire prevention as a case study. VPP shows the benefits of the total viewshed computation: we find several safe and flyable paths providing all-round visibility from the results obtained from a thorough visibility analysis that exposes the most hidden areas. These results open the possibility of monitoring challenging regions of complex terrain using the onboard camera of a single UAV, providing more effective territory monitoring in terms of reduced path length and increased area covered.

As the generated path is composed of several waypoints, we also provide a method to determine the direction of the onboard camera for each one, aiming to avoid overlapping visual areas of already monitored locations to increase the amount of terrain covered. These images—taken throughout the flight—can be analysed in real time to alert local authorities in case of fire. Finally, we assessed the performance of VPP by simulating the UAV flight over two complex terrain regions of 48 and 202 km^2 , where we achieved 98.7% and 94.5% terrain coverage, respectively. In addition, a real flight test over a reduced area was also presented to analyse the real limitations of the proposal.

The results proved that our proposal successfully covered each studied area, which would mean high protection against forest fires at relatively low cost. Important companies and organisations, such as the Andalusian Regional Government, Malaga City Council, and AERTEC Solutions [177] have expressed their interest in writing in the findings of this study.

The results of the work presented in Chapter 5 have led to the publication of the following article:¹

[178] VPP: Visibility-based Path Planning Heuristic for Monitoring Large Regions of Complex Terrain using a UAV Onboard Camera

A.J. Sanchez-Fernandez, L.F. Romero, G. Bandera, and S. Tabik

IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 2021

In a nutshell, this thesis has tackled three well-known spatial problems from different points of view:

- a) In Chapter 3, we started by analysing the behaviour of the most advanced latent fingerprint identification algorithm available to date. Specially, we focused on finding weaknesses and processing patterns that we could optimise. We realised that this algorithm did not exploit the heterogeneity of current systems. Therefore, we developed an alternative based on asynchronous data processing between CPU and GPU specially designed for heterogeneous systems.
- b) Afterwards, in Chapter 4, we applied what we had learned so far to another spatial problem of great interest: the total viewshed computation in large areas. We found an alternative that significantly improved the execution time by completely restructuring the input data before processing. We tested several alternatives based on heterogeneous computing, but those approaches only using dedicated GPU hardware outperformed them due to the matrix nature of the problem.
- c) As a final part of this work, in Chapter 5, we addressed the well-known path planning problem considering fire prevention in forest regions as a case study. Specially, we tried to find the path providing the maximum visibility of the terrain during the UAV flight. To develop this heuristic, we used the heterogeneous programming concepts learned during the development of the two preceding algorithms. Finally, the heuristic obtained from this study can generate an efficient trajectory for the complete monitoring of the target area.

¹This article has been accepted for publication in a future issue of IEEE J-STARS.

6.2. Future work

The lines of research that would form the natural continuation of the work included in this thesis could be the following:

Regarding ALFI, the next step would be assessing energy efficiency (e.g., in FLOPS/watt) by comparing this approach and the DMC-CC algorithm. We want to study how the gain in execution speed affects energy consumption, always bearing in mind that energy consumption is a secondary variable in the problem at hand. The most critical variables in the problem of identifying individuals are processing speed and accuracy. Furthermore, we could study whether considering machine learning techniques in matching minutiae increases the absolute identification accuracy provided by ALFI.

For the second line of research concerning sDEM, the natural step would be to extend the tool to other types of topographic analysis and release a cross-platform plug-in for research purposes. It could include slope and elevation analysis and any other feature having similar processing needs. Additionally, we could explore applying sDEM to other problems requiring the analysis of large amounts of data organised in matrix form.

For the path planning problem, the future of the VPP heuristic includes searching for new processing stages that will increase the terrain covered, with fewer waypoints in the path if possible. Moreover, another objective is to test this heuristic on more forest areas of different complexity.

Finally, we can apply many of the proposed solutions to spatial problems completely different from those addressed in this thesis. As an example, we are working on modifying the sDEM algorithm (Chapter 4) to process the data provided to solve one of the problems included in the second Square Kilometre Array Science Data Challenge [179]: the analysis of a vast number of images from a worldwide network of radio telescopes [180].

Appendix A

Formal definitions

This appendix contains the essential functions used in the ALFI algorithm presented in Section 3. Section A.1 describes the definitions of the main structures. Section A.2 presents the different mathematical functions used by ALFI.

A.1. Definitions

- **Definition 1.** Minutiae are points located in the ridge discontinuities of a fingerprint. Every minutia structure is characterised by x and y positions, θ direction, cylinder $c = (\nu, \eta)$ containing its value and norm, respectively, and also type $t \in [0, 1, 2] : t \rightarrow [\text{unknown}, \text{end}, \text{bifurcation}]$.
- **Definition 2.** The set of all possible minutiae is defined as $A = \{(x, y, \theta, \nu, \eta, t) : x, y, \theta, \eta \in \mathbb{R} \mid \nu, t \in \mathbb{N}\}$, where \mathbb{R} and \mathbb{N} represent the sets of real and natural numbers, respectively.

A.2. Mathematical functions

- The function d_θ computes the minimal difference between two quantized angles $\hat{\alpha}$ and $\hat{\beta}$:

$$d_\theta : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$$

$$(\hat{\alpha}, \hat{\beta}) \rightarrow \min(|\hat{\alpha} - \hat{\beta}|, z - |\hat{\alpha} - \hat{\beta}|) \quad (\text{A.1})$$

where z is the total number of quantized angles.

- Likewise, the above function can be used with two minutiae a and b as input:

$$d_\theta : A \times A \rightarrow \mathbb{R}$$

$$(a, b) \rightarrow \min(|\theta_a - \theta_b|, 2\pi - |\theta_a - \theta_b|) \quad (\text{A.2})$$

- The function d_e computes the Euclidean distance given two minutiae a and b as input:

$$d_e : A \times A \rightarrow \mathbb{R}$$

$$(a, b) \rightarrow \sqrt{(x_b - x_a)^2 + (y_b - y_a)^2} \quad (\text{A.3})$$

- The function σ computes the similarity score between two given minutiae a and b by using their minutiae descriptors:

$$\sigma : A \times A \rightarrow \mathbb{R}$$

$$(a, b) \rightarrow 1 - \frac{\sqrt{\text{pop}(\nu_a \oplus \nu_b)}}{\eta_a + \eta_b} \quad (\text{A.4})$$

where ν is the minutia cylinder value, η is the cylinder norm, the XOR operator is denoted as \oplus , and pop is the bit population count operation.

- The function ρ returns the corresponding quality value for an specific minutia depending on H_{q1} and H_{q2} thresholds and a given distance d :

$$\rho : \mathbb{R} \rightarrow \mathbb{R}$$

$$d \rightarrow \begin{cases} 1 & \text{if } d > H_{q2} \\ 0 & \text{if } d < H_{q1} \\ (d - H_{q1}) / (H_{q2} - H_{q1}) & \text{otherwise} \end{cases} \quad (\text{A.5})$$

- The function ψ maps the minutia a into another by using a minutiae pair (b, c) as reference:

$$\psi : A \times A \times A \rightarrow A$$

$$(a, b, c) \rightarrow \left[\begin{pmatrix} c(\Delta\theta) & -s(\Delta\theta) & 0 \\ s(\Delta\theta) & c(\Delta\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_a - x_b \\ y_a - y_b \\ \theta_a - \theta_b \end{pmatrix} + \begin{pmatrix} x_c \\ y_c \\ \theta_c \end{pmatrix} \right]^T \quad (\text{A.6})$$

where c and s denote the sine and cosine trigonometric functions, respectively, and $\Delta\theta = \theta_c - \theta_b$.

- The function σ_e computes the similarity score between two given minutiae a and b according to their Euclidean distance:

$$\sigma_e : A \times A \rightarrow [0, 1]$$

$$(a, b) \rightarrow \begin{cases} 0 & \text{if } u \text{ or } v \text{ or } w \\ 1 - \frac{d_e(a, b)}{H_e} & \text{otherwise} \end{cases} \quad (\text{A.7})$$

where d_e is the Euclidean distance function described in Equation A.3, H_e is an empirical threshold value. Moreover, u denotes $|\Delta x| > H_e$, v is $|\Delta y| > H_e$, and w is $d_e(a, b)^2 > H_e^2$, considering $\Delta x = x_b - x_a$ and $\Delta y = y_b - y_a$.

- The function σ_θ computes the similarity score between two given minutiae a and b according to their direction difference:

$$\sigma_\theta : A \times A \rightarrow [0, 1]$$

$$(a, b) \rightarrow \begin{cases} 0 & \text{if } d_\theta(a, b) > H_\theta \\ 1 - \frac{d_\theta(a, b)}{H_\theta} & \text{otherwise} \end{cases} \quad (\text{A.8})$$

where d_θ is the angular difference function described in Equation A.2 and H_θ is an empirical threshold value.

- The function σ_t computes the similarity score between two given minutiae a and b based on their types:

$$\sigma_t : A \times A \rightarrow [0.5, 0.75, 1]$$

$$(a, b) \rightarrow \begin{cases} 0.75 & \text{if } t_a = 0 \text{ or } t_b = 0 \\ g & \text{otherwise} \end{cases} \quad (\text{A.9})$$

$$g : [0, 1, 2] \times [0, 1, 2] \rightarrow [0.5, 1]$$

$$(t_a, t_b) \rightarrow \begin{cases} 1 & \text{if } t_a = t_b \\ 0.5 & \text{otherwise} \end{cases}$$

Appendix B

Resumen en español

Nuevos enfoques paralelos para resolver eficazmente problemas espaciales en sistemas heterogéneos CPU-GPU

En los últimos años, los enfoques que buscan extraer información valiosa de grandes conjuntos de datos han cobrado especial relevancia en la sociedad actual. En esta categoría destacan los problemas espaciales que comprenden el análisis de datos distribuidos en escenarios bidimensionales. Su objetivo es extraer información relevante que conduzca a una determinada conclusión o decisión. Un ejemplo de ello podría ser la realización de un estudio de visibilidad de una zona determinada para decidir la mejor ubicación de una torre de observación. En la práctica, el análisis espacial suele consistir en procesar una serie de características distribuidas en un plano determinado o una matriz de valores en la que cada una de las celdas se corresponde con un punto del plano. Esto demuestra el carácter abierto y complejo de este tipo de problemas, pero también permite utilizar la imaginación en la búsqueda de soluciones.

Una de las principales complicaciones que encontramos al trabajar con problemas espaciales es que son muy intensivos desde el punto de vista computacional, por lo que suelen requerir mucho tiempo para obtener el resultado deseado. Aquí es donde podemos encontrar en los sistemas heterogéneos una oportunidad para abordar estas complicaciones de una forma más eficiente. Estos sistemas aportan al desarrollador una mayor libertad para acelerar muchos tipos de algoritmos

al aumentar las opciones disponibles de programación paralela, haciendo posible que ciertas partes de un programa se ejecuten en aquellos dispositivos hardware que mejor se adapten a sus requerimientos.

Varios de los problemas bidimensionales que no han sido optimizados utilizando sistemas heterogéneos abarcan áreas muy diversas y que parecen muy diferentes a primera vista. No obstante, estas áreas acaban estando estrechamente relacionadas gracias al procesamiento parecido de las estructuras de datos, haciéndolos adecuados para el uso de hardware dedicado.

La presente tesis doctoral aporta nuevos enfoques paralelos para abordar los siguientes tres problemas espaciales: la identificación de huellas dactilares latentes, el cálculo de la visibilidad total sobre una región de grandes dimensiones y la planificación de trayectorias basada en la maximización de la visibilidad. El trabajo realizado ha servido para desarrollar tres nuevos enfoques paralelos para la resolución de problemas de análisis espacial que no habían sido considerados hasta la fecha, y con la posibilidad de ser aplicados por la comunidad científica a otras áreas con condiciones similares para acelerar el procesamiento de datos.

B.1. Introducción

El desarrollo de Internet junto a su uso generalizado han permitido que tengamos una cantidad inmensa de datos disponibles en casi cualquier área imaginable. En concreto, existe un área de especial interés que ha experimentado un importante crecimiento en los últimos años: el análisis de datos en entornos bidimensionales, comúnmente conocido como análisis espacial [1].

El análisis espacial podría entenderse en términos generales como: *una habilidad general para manipular datos espaciales en diferentes formas y extraer un significado adicional como resultado* [2]. Siendo un poco más específicos, podemos restringir aún más esta definición de forma que el análisis espacial incluya cualquier técnica analítica que busque hallar (i) la distribución espacial de una variable, (ii) la relación entre la distribución espacial de diferentes variables o (iii) las asociaciones entre variables. Cualquiera de estos aspectos se suele conseguir analizando las propiedades topológicas, geométricas o geográficas de cada variable bajo estudio.

En la práctica, los problemas derivados de este tipo de análisis no están claramente definidos, ni tienen una solución directa y sencilla. No obstante, se puede partir de la idea de que los problemas espaciales suelen requerir un tipo específico de estructura de datos donde los valores implicados no se encuentran aislados. De

hecho, suele existir una relación entre valores adyacentes dentro de la base de datos. Por lo tanto, los datos espaciales proporcionan información relacionada con la ubicación, donde destacan especialmente dos tipos de estructuras de datos [3]:

- Los **archivos vectoriales** representan formas geométricas básicas que incluyen puntos (vértices individuales, por ejemplo, edificios), líneas (dos o más vértices que forman una figura geométrica abierta, por ejemplo, ríos) y polígonos (tres o más vértices que forman una figura geométrica cerrada, por ejemplo, distritos).
- Los **archivos ráster** consisten en una matriz de píxeles o celdas dispuestas en filas y columnas, donde cada celda puede contener un valor discreto o continuo como la densidad de población o la elevación, respectivamente. Además, este tipo de datos suele ser cuadrado y estar espaciado regularmente.

Como norma general, los datos vectoriales son ideales para representar características discretas como ubicaciones específicas, territorios y fronteras; mientras que los datos ráster son más adecuados para características continuas, como podrían ser la elevación o la precipitación, entre otras. El ejemplo más representativo de los datos ráster está relacionado con los sistemas de información geográfica y, más concretamente, con los modelos digitales de elevación. En estos, cada celda de la matriz representa el valor de la elevación del punto correspondiente en el terreno. De esta forma, se puede observar, por ejemplo, cómo un punto que podría corresponderse con la cima de una montaña tendrá celdas adyacentes con valores de elevación similares pero ligeramente inferiores. No obstante, hay que tener en cuenta que los datos espaciales no se limitan únicamente al ámbito geográfico, sino que tienen un enfoque mucho más abierto relacionado con la distribución espacial.

Otra característica interesante de los problemas espaciales reside en el carácter repetitivo de las operaciones utilizadas para analizar los datos. Normalmente se repiten las mismas operaciones sobre el conjunto de datos pero variando la variable objetivo, por ejemplo, la dirección de la línea de visión va variando entre 0 y 360° si se considera el problema de la visibilidad desde un punto dado. Es por ello que los datos utilizados en el análisis espacial cumplen el principio de localidad, también conocido como localidad de referencia, que establece que un procesador tiende a acceder a las mismas ubicaciones de memoria a lo largo del tiempo, es decir, los datos utilizados recientemente tienen más probabilidades de ser reutilizados por el programa en un futuro próximo. En la práctica, se pueden distinguir dos tipos de localidad:

- **Localidad temporal:** los datos consultados recientemente tienen una probabilidad alta de ser consultados en un futuro próximo.
- **Localidad espacial:** los datos almacenados en posiciones cercanas en memoria tienen una alta probabilidad de ser accedidos en un futuro cercano. Un caso particular de esto sería la localidad secuencial, que tiene lugar cuando los datos están dispuestos en la memoria de forma secuencial y se accede a ellos de la misma manera, como ocurre con el acceso lineal a una matriz unidimensional.

Los datos espaciales suelen cumplir el primero de los tipos de localidad cuando se analizan, aunque no es el que los representa plenamente. El segundo tipo sí lo hace, ya que los datos en entornos bidimensionales suelen estar relacionados entre sí—como ocurre en el ejemplo mencionado de la cima de la montaña y los puntos circundantes—y, por tanto, son almacenados de forma contigua en la memoria. Este hecho abre la puerta al desarrollo de nuevos enfoques para acelerar el procesamiento de datos espaciales utilizando sistemas modernos.

B.2. Motivación y Línea de Investigación

A primera vista, muchos problemas espaciales parecen completamente diferentes y sin relación alguna entre sí; sin embargo, existen algunas similitudes, normalmente ocultas, que deberían explorarse para poder aplicar nuevos enfoques que se han mostrado válidos en otros ámbitos. Utilizando la amplia definición de análisis espacial, dos escenarios similares podrían ser: los píxeles de una imagen o los valores de elevación de un Modelo Digital de Elevaciones (DEM, del inglés *Digital Elevation Model*). Así, el color de un píxel tiene una fuerte relación con los de su entorno, al igual que la altitud de un punto en un terreno con los colindantes. Este hecho aumenta las posibilidades para encontrar otras soluciones a problemas de gran complejidad, como la identificación de huellas dactilares, el cálculo de la visibilidad total y la planificación de trayectorias basadas en maximizar la visibilidad. A continuación, se describirán algunas de las características más relevantes de estos algoritmos y se introducirán las preguntas a las que se intenta dar respuesta a través de la presente tesis.

La identificación de huellas dactilares se considera uno de los métodos de identificación biométrica más utilizados en la actualidad. Cada huella dactilar tiene un conjunto de características denominadas minucias que se refieren principalmente a los puntos donde se dividen o terminan las crestas de la superficie de los dedos, aunque existen otros. En la práctica, cada minucia suele estar definida

por tres valores: dirección, tipo y coordenadas dentro de la imagen bidimensional de la huella dactilar. Por lo tanto, la identificación de un individuo se logra comparando las características relativas de las minucias de la huella dactilar con las minucias de cada una de las huellas dactilares contenidas en una determinada base de datos.

El proceso de identificación es un reto debido al gran número de relaciones complejas que se definen y comprueban en cada comparación de minucias para su similitud. Además, el problema se complica aún más cuando se busca la **identificación de huellas dactilares latentes** (en inglés, *latent fingerprint identification*). Este tipo de huellas dactilares se dejan de manera involuntaria en una superficie por los depósitos de sudor y/o aceite de la yema del dedo. En términos generales, suelen tener baja calidad y requieren mucho más procesamiento.

Los algoritmos de comparación de huellas dactilares (latentes o no latentes) más avanzados son muy costosos desde el punto de vista computacional, consumen mucho tiempo y son particularmente ineficientes cuando se trabaja con grandes bases de datos. Actualmente, algunos algoritmos de comparación de huellas dactilares emplean aceleradores hardware, pero no existen enfoques similares aplicados a la identificación de huellas dactilares latentes. Por lo tanto, se plantea una cuestión principal:

- *¿Es posible desarrollar un nuevo algoritmo de identificación de huellas dactilares latentes basado en el método de identificación más preciso para que funcione en sistemas heterogéneos?*

Si esto es posible, otras preguntas derivadas son:

- *¿Es el algoritmo resultante suficientemente rápido para la identificación de huellas dactilares latentes?*
- *¿Logra un valor alto de precisión en la identificación?*

La experiencia adquirida al abordar el problema bidimensional de la identificación de huellas dactilares puede servir de base para encontrar nuevas soluciones al segundo problema espacial mencionado: el cálculo de la **visibilidad total** (*total viewshed*). Este problema se relaciona con la obtención del área visible para todos y cada uno de los puntos que conforman un DEM. Ambos problemas espaciales tienen varias similitudes de las que se pueden extraer ventajas como la explotación del paralelismo basado en la partición y proximidad de los datos según su localización bidimensional.

En los sistemas de información geográfica, es habitual el uso de algoritmos similares al trazado de rayos (*ray-tracing*) o de barrido radial (*rotational sweep*) cuando se trabaja con DEMs para estudiar parámetros cuya magnitud disminuye con el cuadrado de la distancia (por ejemplo, señales de radio, ondas sonoras y visión humana). En la práctica, estos algoritmos llevan asociado el acceso a los datos en forma matricial que, en la mayoría de los casos, aunque regulares, conducen a un pobre aprovechamiento de la localidad espacial en memoria. Por ello, se pretende analizar cómo una reorganización completa de las matrices de datos en función de cada dirección produce una mejora considerable en el rendimiento, especialmente en aquellos algoritmos con alta intensidad computacional. Se abordan las siguientes cuestiones:

- *¿Es posible encontrar una nueva estructura de almacenamiento de datos que mejore la localidad en memoria de los datos de elevación para los problemas de análisis del terreno?*

Si es posible, otras preguntas a las que hay que dar respuesta son:

- *¿Aumentará sustancialmente el rendimiento en los algoritmos que buscan analizar la visibilidad en un terreno?*
- *¿Es especialmente eficaz para el problema de la visibilidad total?*

La parte final de esta tesis pretende aportar una nueva solución a un problema muy importante en la actualidad conocido comúnmente como **planificación de trayectorias** (*path planning*), donde se precisa el conocimiento del cálculo de la visibilidad, así como la gestión y procesamiento de datos distribuidos espacialmente—como se da también en el problema de identificación de huellas dactilares.

En particular, se abordará el problema de encontrar el camino más corto que proporcione la mayor cobertura visual para un terreno determinado, basándose en el cálculo de la visibilidad. El dispositivo encargado de seguir el camino y vigilar la zona será un vehículo aéreo autónomo no tripulado (UAV), comúnmente conocido como dron, con una cámara instalada. Las preguntas clave de esta investigación son las siguientes:

- *¿Se puede encontrar una nueva alternativa para abordar el problema de la planificación de trayectorias cuyo objetivo principal sea la maximización de la visibilidad desde la trayectoria generada?*

- *¿Es de utilidad para este supuesto el algoritmo desarrollado para abordar el problema de la visibilidad total?*
- *¿Es la dirección de la cámara del UAV un factor crítico a tener en cuenta para maximizar la cobertura visual durante el vuelo?*

B.3. Computación en la Actualidad

Los algoritmos intrínsecamente secuenciales (por ejemplo, el problema de los n -cuerpos en física) son imposibles de paralelizar de forma eficiente y, por tanto, son más adecuados para ser ejecutados en CPUs de un solo núcleo que funcionan a altas frecuencias. Por el contrario, los algoritmos vergonzosamente paralelos, como el análisis de Monte Carlo, los métodos de diferencias finitas y el renderizado de modelos 2D/3D, se benefician de ser ejecutados en sistemas de CPU multinúcleo, o incluso en aceleradores hardware con muchos núcleos que funcionan a frecuencias más bajas. Por lo general, la mayoría de los algoritmos que tratan de resolver problemas del mundo real pueden ser divididos en subtareas, las cuales requieren ejecución en serie o en paralelo, y que se pueden beneficiar de la ejecución concurrente en diferentes dispositivos.

B.3.1. Principales dispositivos implicados

En el pasado, el procesador era el protagonista de la ciencia de la computación con los primeros sistemas de CPU de un solo núcleo en los que el rendimiento se maximizaba al aumentar la velocidad del reloj. Este parámetro aumentó progresivamente hasta que llegó un punto en el que no se podía incrementar más la frecuencia y fue entonces cuando se introdujeron por primera vez las CPUs multinúcleo. Estas unidades consiguen más rendimiento con un menor consumo unitario de energía, pero con el inconveniente de consumir mucha energía en unidades no computacionales (lógica y caché, por ejemplo) y también tener que escribir código paralelo para aprovechar al máximo el hardware subyacente.

Al mismo tiempo, y siguiendo un enfoque similar, los aceleradores hardware empezaron a ganar también cierta importancia en la computación. Estos dispositivos son elementos hardware especializados que tienen una mayor eficiencia que las CPUs de propósito general para realizar ciertas tareas específicas, principalmente aquellas que son computacionalmente intensivas. Los núcleos están diseñados específicamente para maximizar el rendimiento, consumiendo una cantidad importante de energía en la realización de cálculos. Estos núcleos tienen

menos transistores funcionando a frecuencias más bajas, con la contrapartida de sacrificar funcionalidad: imposibilidad de ejecutar sistemas operativos, realizar operaciones lógicas de forma eficiente, etc. En la actualidad, algunos de los aceleradores más utilizados son los circuitos integrados para aplicaciones específicas (ASICs), las matrices de puertas lógicas programables en campo (FPGAs) y las unidades de procesamiento gráfico (GPUs). Cada uno de estos dispositivos ofrece diferentes balances entre eficiencia, configuración y coste de desarrollo [17].

Por todo ello, se puede decir que la situación ha cambiado de forma significativa desde la aparición de los primeros procesadores y el futuro de la informática de alto rendimiento parece dirigirse hacia el procesamiento en sistemas heterogéneos en los que la tarea de procesamiento es realizada por la CPU en combinación con uno o varios aceleradores hardware para aprovechar al máximo todos los recursos disponibles.

Un sistema heterogéneo contiene diferentes tipos de unidades de cálculo, entre ellas (i) uno o varios procesadores de propósito general que ejecutan un sistema operativo y también el programa, además de distribuir los datos entre (ii) todos los aceleradores hardware disponibles que ayudan al procesador a acelerar el cálculo. En la práctica, cada acelerador se adapta mejor a un tipo de tarea concreta, siendo realizada de una forma más eficiente que si la llevara a cabo únicamente la CPU. Así, el consumo de energía y la velocidad de ejecución mejoran al introducir hardware especializado.

B.3.2. Sistemas heterogéneos CPU-GPU

La estrategia más habitual en la computación heterogénea es combinar el cálculo de propósito general en la CPU, donde es posible un mayor control de la implementación, con el procesamiento rápido especializado en la GPU, lo que permite reducir el tiempo de ejecución al analizar grandes bases de datos. Los sistemas heterogéneos CPU-GPU pueden clasificarse en dos grupos, según la disposición de las unidades: ambas en el mismo chip o en piezas de hardware independientes (hardware dedicado). La primera de estas alternativas es ampliamente utilizada en dispositivos pequeños de uso cotidiano, como smartphones, portátiles y ordenadores de sobremesa de oficina. Los procesadores que integran la CPU y la GPU en el mismo chip suelen ser menos potentes, debido en gran medida a las limitaciones existentes en términos de espacio y energía. Por otro lado, el hardware dedicado suele instalarse en ordenadores de sobremesa más potentes diseñados para videojuegos o diseño gráfico, por ejemplo, y sistemas de computación (Figura B.1).

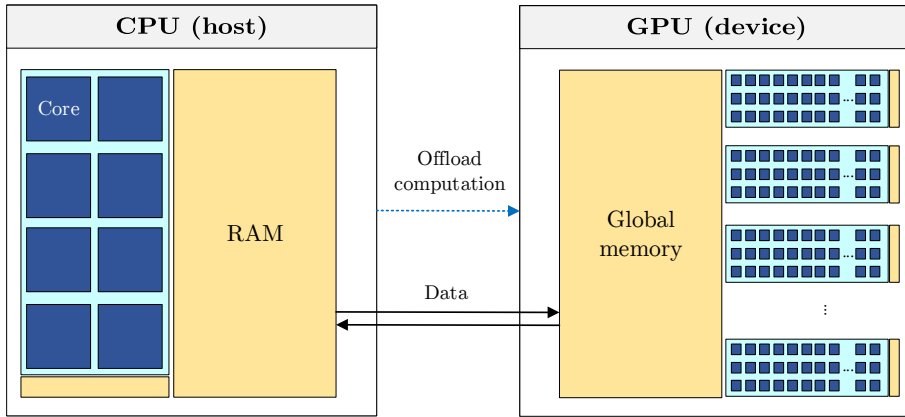


Figura B.1: Sistema heterogéneo CPU-GPU con GPU dedicada.

Este enfoque tiene un gran impacto en la dificultad de la tarea de programación, convirtiéndose en un reto cuya complejidad aumenta con el número de dispositivos considerados. Esto se debe a que la programación en paralelo aumenta la complejidad por la sincronización que se tiene que implementar en las zonas de código donde se realizan accesos a datos compartidos. Este problema, normalmente relacionado con las CPUs multinúcleo, se extiende también a los sistemas heterogéneos en la misma medida. Sin embargo, en términos generales, las ventajas de la computación heterogénea, que incluyen una mayor velocidad de procesamiento, una gran capacidad para manejar grandes bases de datos y un menor coste, superan las desventajas. Hoy en día, son muchas las aplicaciones que se benefician de ser rediseñadas para ejecutarse en sistemas heterogéneos, consiguiendo grandes reducciones en el tiempo de procesamiento. Algunas herramientas muy conocidas que ayudan en la tarea de optimizar los algoritmos existentes o desarrollar otros nuevos son OpenCL, SYCL y CUDA.

La computación heterogénea con GPU ha experimentado un gran crecimiento en tres áreas de gran importancia para la comunidad científica, como son la Inteligencia Artificial (IA) [18, 19], el Aprendizaje Automático [20] y el Aprendizaje Profundo [21, 22]. Estos tres conceptos están estrechamente relacionados, siendo el primero el que engloba a los dos siguientes. También, existen otras aplicaciones pertenecientes a diversas áreas de conocimiento donde se sigue esta metodología de procesamiento heterogéneo: análisis de series temporales [23, 24], procesamiento de datos médicos [25, 26, 27], máquinas de aprendizaje extremo [28, 29], integración numérica [30] y simulación del crecimiento urbano [31]. En consecuencia, los problemas relacionados con el análisis espacial también cumplen los

requisitos para ser ejecutados en sistemas heterogéneos, ya que son aplicaciones computacionalmente intensivas que tienen que manejar grandes bases de datos.

Es por ello que el presente trabajo se centra en el aprovechamiento del paralelismo intrínseco existente en tres algoritmos espaciales pertenecientes a diversas áreas de investigación punteras: la identificación de huellas dactilares latentes, el análisis de la visibilidad total y la planificación de trayectorias en terrenos complejos.

En primer lugar, la identificación de huellas dactilares latentes tiene que analizar una base de datos que puede dividirse fácilmente en partes más pequeñas, donde hay que lidiar con las complejas dependencias que hacen imposible la paralelización mediante tareas independientes. Por otro lado, los otros dos problemas, relacionados con el cálculo de vistas y la planificación de trayectorias, tienen que analizar una “menor cantidad” de datos, pero siendo mucho más difícil, casi imposible, la división de la base de datos en porciones más pequeñas e independientes. Por lo tanto, en las tres secciones siguientes se propondrán diferentes soluciones para lograr un procesamiento eficiente de los datos en los problemas de análisis espacial mencionados anteriormente mediante la inclusión de la GPU en las etapas de procesamiento, ya sea por sí sola, o conjuntamente con la CPU en sistemas heterogéneos.

B.4. ALFI: Identificación de Huellas Dactilares Latentes

Uno de los procedimientos más esenciales realizados en las investigaciones criminales está relacionado con la identificación de un sospechoso a partir de las huellas dactilares parciales dejadas en la superficie de un objeto de la escena del crimen. Este proceso se denomina identificación de huellas dactilares latentes y abordar esta tarea supone un reto ya que (i) requiere analizar bases de datos masivas en un tiempo razonable y (ii) la identificación del sospechoso normalmente se logra empleando diferentes metodologías con dependencias de datos muy complejas [32]. Estos dos hechos hacen que aprovechar plenamente la potencia de los sistemas heterogéneos CPU-GPU sea una tarea muy compleja. La mayoría de los esfuerzos en este contexto se centran en mejorar la precisión de los enfoques y descuidan la reducción del tiempo de procesamiento. De hecho, el algoritmo de la bibliografía con mayor precisión en la identificación de huellas latentes fue diseñado para un solo hilo de ejecución [33].

En este capítulo se presenta una nueva metodología de computación distribuida y paralela para la identificación de huellas dactilares latentes denominada procesamiento asíncrono para la identificación de huellas dactilares latentes (ALFI, del inglés *Asynchronous processing for Latent Fingerprint Identification*). Esta metodología se desarrolla teniendo en cuenta las características técnicas de la CPU—unidad huésped—y la GPU—unidad dispositivo—para aprovechar al máximo todos los recursos hardware disponibles en los sistemas heterogéneos. Esto se logra combinando un enfoque eficiente de comunicación asíncrona CPU-GPU con un paralelismo de grano fino a nivel de descriptor de huella dactilar. En la práctica, ALFI es capaz de analizar grandes bases de datos en menos tiempo que el algoritmo más moderno recogido en la literatura [33], proporcionando resultados de precisión muy similares. ALFI puede ser de gran ayuda en la resolución de crímenes por parte de las autoridades locales, ya que los tiempos de procesamiento de las huellas dactilares se reducen drásticamente utilizando, para ello, el hardware disponible en casi cualquier ordenador actual.

B.4.1. DMC: agrupación deformable de minucias

El algoritmo más preciso de la bibliografía en la identificación de huellas dactilares latentes sigue una metodología propia de agrupación deformable de minucias (en inglés, *Deformable Minutiae Clustering*), utilizando el descriptor MCC (*Minutia Cylinder-Code*) [70], por lo que se denominará de aquí en adelante DMC-CC [33]. DMC-CC se desarrolló fusionando los siguientes cuatro métodos independientes y una etapa final de cálculo de la similitud:

- a) La comparación local emplea el descriptor MCC como estructura de entrada del proceso para encontrar el primer grupo de parejas de minucias coincidentes. Este descriptor se basa en estructuras de datos 3D generadas considerando las posiciones y los ángulos de las minucias [65].
- b) El método de discriminación [77] calcula un valor de calidad para cada minucia en la huella dactilar latente y en la impresión de la huella dactilar basándose en la localización de las minucias vecinas.
- c) El método DMC [33, 78] encuentra grupos de parejas de minucias, junto con un valor de peso para cada uno, a partir del conjunto de parejas de minucias coincidentes encontrado en la etapa inicial. Tras fusionar las agrupaciones, se descubre un conjunto final de parejas de minucias con el que se calcula una puntuación inicial de similitud entre las huellas dactilares.

- d) Se aplica el método *Thin Plate Spline* (TPS) [76] para evitar la pérdida de datos debida a la deformación de las huellas dactilares y encontrar nuevas parejas de minucias coincidentes. Estas parejas podrían haber sido descartadas en los pasos anteriores debido a los efectos de la deformación y pueden mejorar el valor de similitud previamente calculado.
- e) En el último paso se obtienen los diferentes resultados estadísticos relativos al cálculo de la similitud.

Como punto negativo, el algoritmo DMC-CC presenta varias dependencias inevitables y complejas que obligan a ejecutar sus pasos de forma secuencial. Este hecho provoca una importante pérdida de rendimiento cuando se ejecuta en sistemas multinúcleo y heterogéneos.

B.4.2. Metodología

B.4.2.1. Procesamiento asíncrono de datos

ALFI se inspira en el algoritmo DMC-CC para, tras un rediseño completo de este, conseguir un procesamiento más rápido y eficiente en sistemas heterogéneos. Se desarrollan nuevos métodos para que la comparación local, la discriminación y parte del método DMC sean ejecutados en diferentes kernels en el dispositivo. El resto de etapas también son modificadas para recibir los resultados del dispositivo para que el huésped los ejecute y así equilibrar la carga computacional entre el huésped y el dispositivo. Estos últimos pasos calculados en el huésped se denominarán conjuntamente etapa de evaluación final multihilo (FES, *Final Evaluation Stage*) a lo largo de este capítulo. Además, el huésped también coordina el lanzamiento de todas las operaciones que se ejecutan en el dispositivo.

Sean L y T_{DB} la huella latente utilizada como caso de estudio y la base de datos de impresiones de huellas dactilares, respectivamente. En primer lugar, dado que todos los datos de la base de datos no pueden almacenarse por completo en la memoria del dispositivo de una sola vez, ésta debe dividirse en varios lotes de igual tamaño. Entonces, cada huella perteneciente a un determinado lote $T \in T_{DB}$ es comparada con la huella latente L en la unidad huésped, pero siempre después de que varios procesos se hayan completado en el dispositivo. Estos procesos incluyen la transferencia de datos de la unidad huésped al dispositivo $H2D$, la ejecución de los kernels K y las operaciones de transferencia de resultados del dispositivo al huésped $D2H$. ALFI solapa y sincroniza dichas operaciones y las que se llevan a cabo en la unidad huésped de forma eficiente gracias al

uso de eventos de sincronización dentro de un proceso de segmentación (conocido generalmente como *data pipeline* en inglés). Todo ello aplicado durante el procesamiento CPU-GPU, reduciendo al máximo los tiempos muertos.

El comportamiento de ALFI se muestra en la Figura B.2 para el caso particular de T_{DB} dividido en cuatro lotes de huellas $T_i \in T_{DB}, i = 1..4$, en aras de la simplicidad. En primer lugar, la asignación en memoria de la huella latente $A(|L|)$ y de la base de datos completa $A(|T_{DB}|)$ se realiza en la unidad huésped H , en particular utilizando memoria no paginable (*pinned memory*). La asignación de la base de datos completa es posible en el huésped pero no en el dispositivo, ya que normalmente el espacio de memoria disponible en el primero es significativamente superior. Además, se utiliza la memoria no paginable en el huésped ya que de este modo se evita que el espacio de memoria sea intercambiado/movido, mejorando la velocidad de las transferencias de memoria. En cuanto a la gestión de la memoria en el dispositivo D , la asignación de la huella latente $A(|L|)$ se realiza en el arranque. El resto del espacio de memoria disponible se divide en dos grandes subespacios. Cada uno de ellos, denotado como $A(n_m)$, incluirá un lote diferente de huellas dactilares, de forma que las transferencias de memoria y el cálculo puedan solaparse. Asimismo, cada espacio (incluyendo el lote dentro de ella) es gestionado por uno de los dos streams no predeterminados de CUDA str_0 y str_1 en el dispositivo.

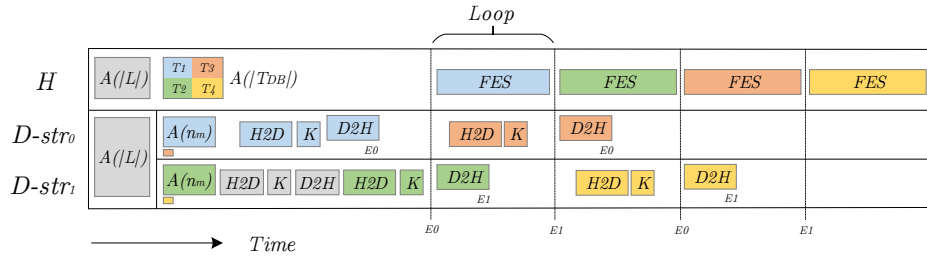


Figura B.2: Flujo de operaciones ejecutadas por ALFI a lo largo del tiempo en la unidad huésped (H) y los dos streams de CUDA en el dispositivo ($D-str_0$, $D-str_1$). Se considera una huella latente L y cuatro lotes T_{1-4} , resultantes de la división de la base de datos T_{DB} . A estos lotes se le asignan (A) los espacios de memoria necesarios en el dispositivo y huésped, donde n_m es el número de huellas dactilares procesadas por stream de CUDA en el dispositivo. Los streams de CUDA str_0 y str_1 y sus correspondientes eventos de sincronización E_0 y E_1 coordinan las operaciones solicitadas en el dispositivo. Estas operaciones conllevan transferencias de datos del huésped al dispositivo ($H2D$), del dispositivo al huésped ($D2H$) y la ejecución de los kernels (K). Estas operaciones se solapan con la etapa de evaluación final multihilo (FES) ejecutada en el huésped.

Un ejemplo del flujo de operaciones de ALFI podría ser el siguiente: cada espacio de memoria en el dispositivo se llena con los lotes T_1 y T_2 al inicio, que son gestionados por str_0 y str_1 , respectivamente. En las siguientes iteraciones, T_3 se almacenará en el primer espacio de memoria para el stream str_0 mientras se realiza el procesamiento de T_2 en el stream str_1 y después de que finalice la operación $D2H$ que contiene los resultados del procesamiento T_1 (evento E_0). Del mismo modo, T_4 se almacenará en el segundo espacio de memoria para str_1 mientras el procesamiento de T_3 está teniendo lugar en str_0 y después de que la operación $D2H$ que contiene los resultados de T_2 haya terminado (evento E_1).

De este modo, los datos se almacenan siempre en el dispositivo antes de iniciar el procesamiento, reduciéndose así los tiempos de inactividad. Todas estas operaciones ocurren mientras el huésped ejecuta la función FES sobre el lote correspondiente de la cola de tareas.

B.4.2.2. Paralelismo de grano fino en el procesamiento

Una vez que los datos se han asignado correctamente en la memoria del dispositivo, se lanzan cuatro kernels diferentes K encargados del procesamiento de los lotes de huellas dactilares.

La comparación local de minucias se realiza en $K_{1,2}$ con un número fijo de posibles parejas de minucias coincidentes, es decir, para cada minucia de la huella latente se le asigna la minucia más similar de la huella impresión. Después, la puntuación de calidad de cada minucia dentro de la huella dactilar latente y del lote de impresiones de huellas dactilares se realiza ejecutando dos veces el kernel K_3 (una para cada huella) con ligeras variaciones para la huella dactilar latente. Por último, en el kernel K_4 se realiza, para un lote específico de huellas dactilares, la alineación de las parejas de minucias para encontrar grupos de estas estructuras que sumarán al cómputo global. Los kernels anteriormente mencionados siguen una estrategia común de cómputo, exceptuando el kernel K_1 que implementa una versión modificada del algoritmo descrito en [15].

Dentro de cada kernel, ALFI establece que cada hilo de ejecución se encarga de procesar una determinada minucia de un lote de huellas dactilares, según su número de identificación de hilo tid (Figura B.3). Por ejemplo, considerando un lote de huellas T que contiene m minucias en total y un kernel K , el hilo con $tid = s$ elegirá y analizará la similitud de la minucia con índice s con las minucias de la huella latente. Este hilo realizará todas las operaciones solicitadas teniendo en cuenta los límites de la huella dactilar (índices de las minucias iniciales s y finales e) a los que pertenece la minucia elegida. Después de ejecutar los

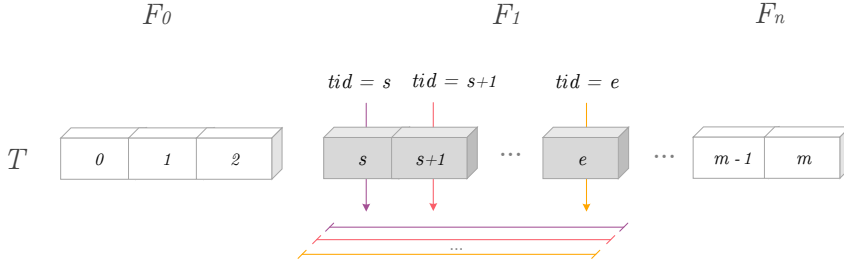


Figura B.3: Procesamiento de las impresiones dactilares realizado por ALFI en un stream de CUDA en el dispositivo. Cada hilo tid realiza todas las operaciones requeridas en un kernel K sobre su correspondiente minucia de una huella F_i dentro del lote de huellas T .

cuatro kernels en el dispositivo, se generará un conjunto de resultados parciales vT para cada lote de huellas dactilares procesado. Este resultado se transfiere a la unidad huésped, siendo utilizado como dato de entrada para realizar la etapa FES multihilo, que se encarga de obtener el valor final de similitud entre la huella dactilar latente y las impresiones dactilares.

B.4.3. Experimentos y resultados

Esta sección analiza y compara nuestra propuesta con respecto al estado del arte en términos de precisión y rendimiento computacional empleando bases de datos ampliamente utilizadas.

B.4.3.1. Diseño de los experimentos

Para realizar un análisis exhaustivo, se utilizan tres sistemas heterogéneos CPU-GPU (S_{1-3}) cuyas características se recogen en las Tablas B.1 y B.2.

Con respecto a la implementación, ALFI se ha desarrollado utilizando los lenguajes de programación C++ y CUDA C++. Estos son lenguajes compilados, lo que quiere decir que son traducidos a lenguaje máquina durante el proceso de compilación y siempre antes de ser ejecutados, permitiendo probar con mayor facilidad el rendimiento de ALFI tanto en sistemas operativos Linux como Windows empleando la misma implementación. Además, según una investigación

previa presentada en [82], la utilización de C++ como lenguaje de programación mejora significativamente el rendimiento computacional en la identificación de huellas latentes.

Tabla B.1: Especificaciones de las unidades huésped utilizadas en los experimentos.

Parámetro	S ₁ (Linux)	S ₂ (Ambos)	S ₃ (Windows)
Tipo de procesador	Intel Xeon	Intel Core	AMD Ryzen
Modelo de procesador	E5-2698 v3	i5-8600K	7-1700x
Número de núcleos CPU	16	6	8
Número de hilos	32	6	16
Frecuencia (GHz)	2,3	3,6	3,4
Memoria RAM (GB)	256	8	16
Caché L1 (kB)	8x64	6x64	8x96
Caché L2 (kB)	8x256	6x256	8x512
Caché L3 (MB)	1x40	1x9	2x8

Tabla B.2: Especificaciones de los dispositivos utilizados en los experimentos.

Parámetro	S ₁ (Linux)	S ₂ (Ambos)	S ₃ (Windows)
Modelo	GTX 980	GT 1030	GTX 1050-Ti
Arquitectura	Maxwell	Pascal	Pascal
Número de núcleos CUDA	2 048	384	768
Número de SMs	16	3	6
Frecuencia base (MHz)	1,12	1,22	1,12
Memoria global (GB)	4	2	4
Memoria por bloque (kB)	48	48	48
Max. número de hilos por bloque	1 024	1 024	1 024
Hilos por <i>warp</i>	32	32	32
Ancho de banda de memoria (GB/s)	224	48	112
Rendimiento (TFLOPs)	4,6	1,13	2,14

B.4.3.2. Bases de datos

La base de datos NIST SD27 [85], que incluye las huellas dactilares junto a sus minucias, se utiliza en los experimentos para probar el rendimiento de ALFI en identificación. Esta base de datos ampliamente utilizada en la literatura contiene 258 huellas dactilares latentes recogidas de casos reales con las imágenes disponibles a 500 dpi. Cada caso incluye la imagen de la huella dactilar latente y la huella dactilar rodada coincidente, donde los expertos han validado todas las minucias. Además, se han diseñado seis bases de datos ampliadas siguiendo diferentes combinaciones de huellas dactilares, como se muestra en la Tabla B.3.

Entonces, la base de datos NIST SD27 se amplía con huellas dactilares rodadas de NIST SD4 [86] y NIST SD14 [87] para obtener bases de datos de tamaño pequeño (B_{1-3}) y medio (B_{4-5}). Para obtener una base de datos ampliada más grande, se incluyeron en B_6 huellas dactilares planas sintéticas generadas con el software de demostración SFinGe versión 4.1 (build 1746). Las huellas dactilares generadas con este último software se han utilizado en varias competiciones de verificación de huellas dactilares demostrando que los resultados obtenidos bajo estas condiciones son similares a los obtenidos en bases de datos reales [88]. No obstante, las huellas dactilares planas contienen menos información que las rodadas, lo que puede afectar a los experimentos en términos de precisión y rendimiento computacional. En cuanto a las minucias de cada huella dactilar, estas se extraen utilizando el SDK VeriFinger [89].

Por otro lado, se analiza el rendimiento computacional de ALFI en condiciones lo más parecidas posibles a un caso real. En particular, varias de las seis bases de

Tabla B.3: Bases de datos ampliadas utilizadas en los experimentos, junto al número de huellas dactilares. Las dos columnas de la derecha muestran el número total de huellas dactilares y el número medio de minucias extraídas por huella dactilar, respectivamente.

Base de datos	NIST SD27	NIST SD4	NIST SD14	Sintéticas	Nº huellas	Nº minucias/huella
B_1	258	-	-	-	258	21
B_2	258	-	2 000	-	2 258	149
B_3	258	2 000	-	-	2 258	101
B_4	258	-	27 000	-	27 258	163
B_5	258	2 000	27 000	-	29 258	159
B_6	258	2 000	27 000	357 985	387 243	38



datos ampliadas (de la Tabla B.3) tienen un número similar de huellas dactilares y, por tanto, algunas de ellas pueden descartarse para eliminar la redundancia. Por consiguiente, se consideran las bases de datos ampliadas de tamaño medio y grande relacionadas con B_3 , B_5 y B_6 , ya que tienen un número representativo de huellas dactilares.

B.4.3.3. Análisis de precisión en identificación

Este experimento presenta los resultados de precisión de ALFI utilizando bases de datos de identificación. Para la comparación de resultados, se ha seleccionado el algoritmo DMC empleando los siguientes descriptores de minucias: *Minutia Cylinder-Code* (DMC-CC), *M-Triplets* (DMC-MT) y *Neighboring Minutiae-based Descriptor* (DMC-NMD).

Las curvas CMC (*Cumulative Match Characteristic*), descritas en [53], son los métodos estándar utilizados para evaluar la precisión de los algoritmos de identificación que producen una lista ordenada de posibles coincidencias. Este tipo de resultado muestra la probabilidad de que se produzca una identificación correcta (rango- k de precisión) dentro de un grupo de k candidatos, donde $k = 1 \dots 20$. En la práctica, los examinadores de huellas dactilares latentes pueden solicitar (i) todos los candidatos que posean una puntuación de coincidencia superior a un determinado umbral o (ii) un número específico de candidatos con la puntuación más alta. En cualquier caso, los examinadores normalmente comienzan el análisis con el candidato que tiene el rango más alto (rango-1 de precisión) y continúan con los restantes si no tienen éxito [93]. La búsqueda finaliza cuando se encuentra la huella coincidente o se han analizado todos los candidatos. Por lo tanto, la clave de la evaluación de la identificación no está únicamente en el valor de precisión rango-1, sino también en el rango-20 y la curva CMC completa de forma que el experimento sea lo más parecido a un caso real.

En este experimento, cada huella latente en el NIST SD27 se compara con todas las huellas de tipo impresión de la base de datos. Así se obtienen los correspondientes valores de precisión rango-1 y rango-20 que se presentan en las Tablas B.4 y B.5. A partir de estos resultados, pueden hacerse las siguientes observaciones:

- En la mayoría de los casos, el algoritmo DMC-CC es el mejor clasificado; sin embargo, la diferencia de precisión entre esta versión y ALFI es insignificante.

- En comparación con el algoritmo DMC-MT, ALFI lo supera en aproximadamente 1,6 %, 1,6 % y 1,2 % para las bases de datos B_1 , B_2 y B_3 , respectivamente, considerando los valores de rango-1. Para los valores de rango-20, ALFI supera al mismo algoritmo en aproximadamente 2,7 %, 0,8 %, 3,9 % y 3,9 % para las bases de datos B_2 , B_3 , B_4 y B_5 , respectivamente.
- En comparación con el algoritmo DMC-NMD, ALFI lo supera en aproximadamente 0,8 %, 1,2 %, 1,2 % y 0,8 % para las bases de datos B_1 , B_2 , B_4 y B_5 , respectivamente, considerando los valores de rango-1. Para los valores de rango-20, ALFI supera al mismo algoritmo en aproximadamente 0,4 %, 2,3 %, 0,4 %, 1,9 % y 2,3 % para las bases de datos B_1 , B_2 , B_3 , B_4 y B_5 , respectivamente.

Tabla B.4: Precisión de rango-1 del algoritmo DMC y ALFI mostrados en porcentajes.

Algoritmo	B ₁	B ₂	B ₃	B ₄	B ₅	B ₆
DMC-CC	87,21	81,01	80,62	70,16	69,77	62,79
DMC-MT	82,95	76,74	76,36	69,38	69,38	66,67
DMC-NMD	83,72	77,13	79,46	66,67	66,67	62,79
ALFI	84,50	78,29	77,52	67,83	67,44	61,24

Tabla B.5: Precisión de rango-20 del algoritmo DMC y ALFI mostrados en porcentajes.

Algoritmo	B ₁	B ₂	B ₃	B ₄	B ₅	B ₆
DMC-CC	94,57	91,47	91,09	85,27	84,50	78,68
DMC-MT	93,41	87,60	89,15	79,84	79,84	77,13
DMC-NMD	92,25	87,98	89,53	81,78	81,40	78,29
ALFI	92,64	90,31	89,92	83,72	83,72	75,58

B.4.3.4. Análisis de rendimiento computacional

En esta sección se busca comparar el rendimiento computacional de ALFI con el algoritmo DMC. En concreto, se elige el algoritmo DMC-CC debido a su mejor desempeño en detrimento de DMC-MT y DMC-NMD, como se indica en [33].

En este experimento, una huella latente aleatoria perteneciente a la base de datos NIST SD27 es comparada con las bases de datos ampliadas B_3 , B_5 y B_6

(descritas en la Sección B.4.3.2) utilizando los sistemas S_{1-3} (Sección B.4.3.1). Medimos el tiempo necesario para completar esta tarea y el rendimiento medio en el procesamiento. Este último parámetro se mide en KMPS (*Kilo Matches Per Second*), que significa miles de comparaciones por segundo. Los resultados de este experimento se presentan en función del sistema operativo, ya sea Linux o Windows, en el que se mide el rendimiento del procesamiento.

- Linux

Los resultados se muestran en la Tabla B.6 y en la Figura B.4 en términos de tiempo de ejecución y rendimiento, respectivamente. El algoritmo DMC-CC tomado como referencia se ha portado de C# a C++ para garantizar una comparación justa en Linux. La razón principal de esta elección es que el código original en C# no podía ejecutarse de forma eficiente en Linux, lo que hacía imposible comparar de forma justa ALFI con la presentada por los autores en [33]. Los casos estudiados revelan los siguientes hechos:

- **En S_1 :** ALFI es de media 28,9x más rápido que el algoritmo DMC-CC para la base de datos B_5 . El valor máximo de rendimiento es de 31,13 KMPS, alcanzado por ALFI utilizando la base de datos B_6 .
- **En S_2 :** ALFI es de media 20,6x más rápido que el algoritmo DMC-CC para la base de datos B_6 . El valor máximo de rendimiento es de 43,66 KMPS con la misma base de datos.

Tabla B.6: Resultados de tiempo de ejecución y aceleración de ALFI y DMC-CC (algoritmo de referencia) en Linux. Una huella latente aleatoria de la base de datos NIST SD27 [85] es comparada con tres bases de datos ampliadas (Sección B.4.3.2) utilizando dos sistemas con Linux (Tablas B.1 y B.2).

		S_1		S_2	
Base de datos	Algoritmo	Tiempo (s)	Aceleración	Tiempo (s)	Aceleración
B_3	DMC-CC	4,36	-	2,63	-
	ALFI	0,66	6,6	0,36	7,3
B_5	DMC-CC	91,69	-	57,37	-
	ALFI	3,17	28,9	3,57	16,1
B_6	DMC-CC	292,85	-	183,08	-
	ALFI	12,44	23,5	8,87	20,6

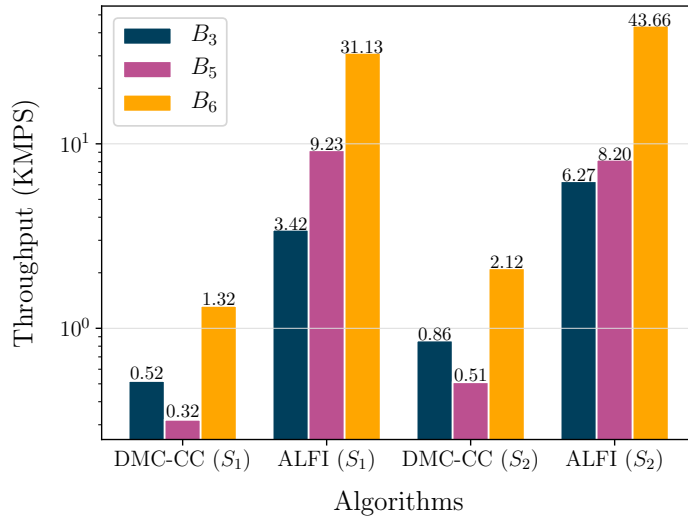


Figura B.4: Resultados de rendimiento de ALFI y DMC-CC en Linux. Una huella latente aleatoria de la base de datos NIST SD27 [85] es comparada con tres bases de datos ampliadas (Sección B.4.3.2) utilizando dos sistemas con Linux (Tablas B.1 y B.2).

- Windows

Los resultados de este experimento se muestran en la Tabla B.7 y en la Figura B.5. El algoritmo DMC-CC usado como referencia para Windows es la implementación C# presentada por sus autores. Cabe destacar lo siguiente:

- **En S_2 :** ALFI es de media 29,2x más rápido que el algoritmo DMC-CC para la base de datos B_6 . El valor máximo de rendimiento es de 23,89 KMPS, alcanzado por ALFI para la misma base de datos.
- **En S_3 :** ALFI es de media 44,7x más rápido que el algoritmo DMC-CC para la base de datos B_6 . El valor máximo de rendimiento es de 24,29 KMPS con la misma base de datos.

Tabla B.7: Resultados de tiempo de ejecución y aceleración de ALFI y DMC-CC (algoritmo de referencia) en Windows. Una huella latente aleatoria de la base de datos NIST SD27 [85] es comparada con tres bases de datos ampliadas (Sección B.4.3.2) utilizando dos sistemas con Windows (Tablas B.1 y B.2).

Base de datos	Algoritmo	S ₂		S ₃	
		Tiempo (s)	Aceleración	Tiempo (s)	Aceleración
B ₃	DMC-CC	6,61	-	9,48	-
	ALFI	0,61	10,8	0,66	14,4
B ₅	DMC-CC	146,98	-	209,22	-
	ALFI	6,71	21,9	5,46	38,3
B ₆	DMC-CC	472,66	-	712,78	-
	ALFI	16,21	29,2	15,94	44,7

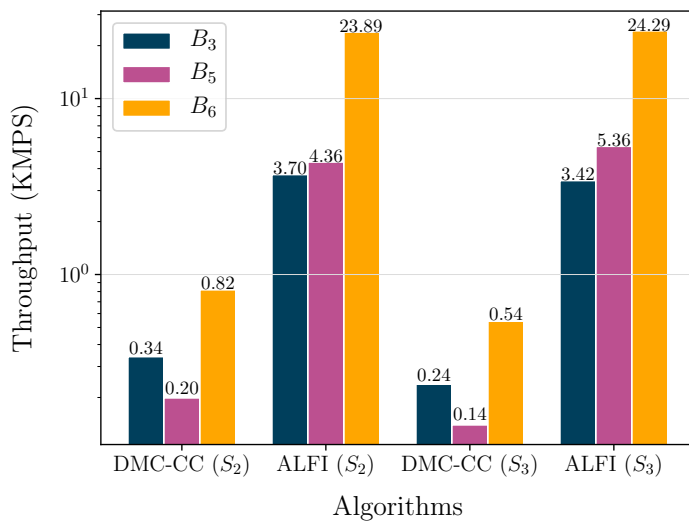


Figura B.5: Resultados de rendimiento de ALFI y DMC-CC en Windows. Una huella latente aleatoria de la base de datos NIST SD27 [85] es comparada con tres bases de datos ampliadas (Sección B.4.3.2) utilizando dos sistemas con Windows (Tablas B.1 y B.2).

B.5. sDEM: Cálculo de la Cuenca Visual Total

El análisis de la cuenca visual es un proceso que consiste en encontrar el área del terreno visible desde la ubicación de un observador. Esta información tiene un gran interés en muchos ámbitos, como en la planificación medioambiental [94] y la lucha contra incendios [95] al encontrarse las mejores ubicaciones para los asentamientos y las torres de vigilancia. Este tipo de problemas tiene como dato de entrada una matriz con valores de elevación regularmente espaciados, denominado Modelo Digital de Elevación (DEM, del inglés *Digital Elevation Model*).

Cabe destacar que estos algoritmos requieren un gran número de accesos a memoria a las posiciones donde se encuentran las matrices 2D que, a pesar de ser regulares, dan lugar a una mala localidad en memoria. Esta afirmación se aplica a cualquier algoritmo destinado a resolver uno de los problemas de cuencas visuales más exigentes desde el punto de vista computacional, denominado cuenca visual total (en inglés, *total viewshed*). Este comprende el cálculo de la cuenca visual para cada uno de los puntos que conforman el DEM, es decir, cada punto del DEM “hace” de observador. Para entender su envergadura, hay que tener en cuenta que un DEM suele contener varios millones de puntos.

En este capítulo, se presenta una nueva metodología denominada modelo digital de elevaciones sesgado (sDEM, *skewed Digital Elevation Model*), considerando el problema de la cuenca visual total como caso de estudio. sDEM aplica una reestructuración completa de los datos del DEM en memoria antes de llevar a cabo el cálculo de la cuenca visual total. En la práctica, el DEM se transforma en una nueva estructura denominada skwDEM en la que los datos se alinean en memoria para mejorar la localidad durante el acceso a los mismos, aumentando así la velocidad de procesamiento.

Con este enfoque, se hace innecesario el uso de técnicas muy comunes que buscan reducir el coste computacional para el problema de la cuenca visual total. Por ejemplo, el considerar una distancia de visibilidad máxima dentro de un área circular alrededor de cada ubicación objetivo. Por otro lado, sDEM también podría mejorar el rendimiento de otros algoritmos que analizan características topográficas relevantes como la pendiente y la elevación.

B.5.1. Conceptos básicos sobre la visibilidad

Dado que en la presente sección se explica la propuesta para acelerar el cálculo de la visibilidad, especialmente en el problema de la cuenca visual total, a continuación se muestra un resumen de los conceptos básicos de este campo:

- **POV:** significa punto de vista (*Point of View*). Es el punto de referencia en el que se encuentra el observador y desde el que se pretende calcular la visibilidad.
- **Cuenca visual:** área del terreno visible desde un único POV (el caso más sencillo), estando compuesta dicha área por un conjunto de puntos visibles.
- **Visibilidad:** estado de un punto objetivo que indica si es visto o no desde una localización de referencia.

Por otra parte, los diferentes problemas de cuencas visuales más estudiados pueden resumirse como sigue (Figura B.6):

- **Cuenca visual singular:** toma un único punto del DEM como POV. El resultado es un mapa booleano que representa puntos visibles y no visibles del terreno.
- **Cuenca visual múltiple:** toma varios puntos del DEM como POVs. El resultado es un mapa booleano acumulado de puntos visibles y no visibles del terreno.
- **Cuenca visual total:** toma cada uno de los puntos del DEM como POV. El resultado es un mapa de calor (*heatmap*) en el que cada celda contiene el valor de área vista desde la ubicación correspondiente en el terreno.

Como se ha mencionado anteriormente, el cálculo de la cuenca visual total es el problema más complicado en términos de requerimiento computacional de entre los problemas de visibilidad estudiados, siendo ésta la principal razón por la que se ha considerado como objeto de estudio en este capítulo.

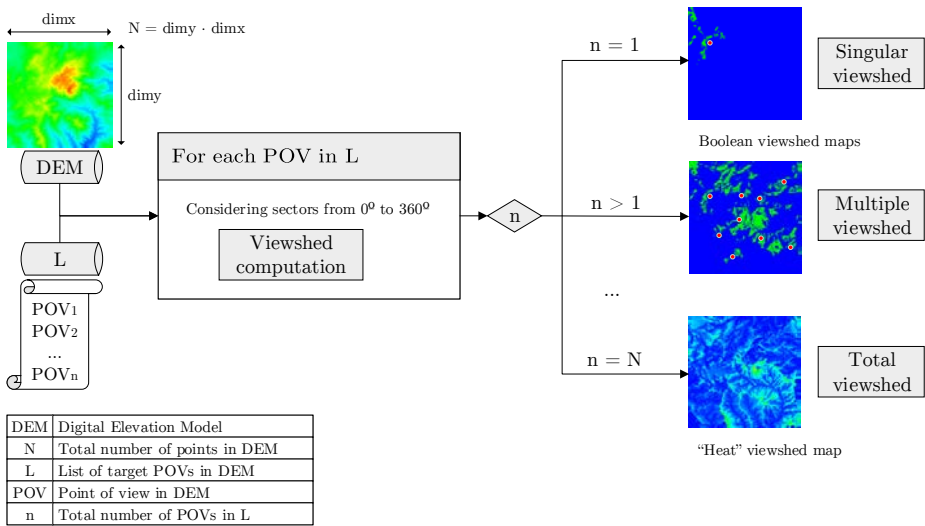


Figura B.6: Diagrama representando los principales tipos de problemas de cuencas visuales: singular, múltiple y total. Esta clasificación depende del número de puntos de vista (POVs) elegido sobre el DEM, cada uno produciendo diferentes resultados.

B.5.2. Cuenca visual total

Un análisis completo de la visibilidad en una zona determinada implica conocer la cuenca visual de cada punto y en todas las direcciones desde cada uno de ellos. Este problema, conocido como cuenca visual total, implica el calcular la cuenca visual de cada punto del DEM como POV (Figura B.7a) y, a continuación, acumular los resultados en un nuevo mapa de calor en el que cada celda contiene el valor de la cuenca visual de la ubicación correspondiente medida, por ejemplo, en km^2 (Figura B.7b).

Son pocos los estudios que abordan el problema del cálculo de la cuenca visual total y, la mayoría de ellos, se centran en una versión simplificada. Por ejemplo, la cuenca visual total calculada en [128] se obtiene reduciendo drásticamente el número de puntos de la malla a procesar. Asimismo, el enfoque utilizado en [129] calcula la visibilidad de áreas pequeñas y no para puntos específicos.

Hasta ahora, el único algoritmo que aborda el problema de la cuenca visual total en DEMs de alta resolución es el algoritmo TVS propuesto en [130, 110]. Este considera los puntos más cercanos a la línea de visión como un conjunto de puntos de muestra almacenados en una estructura llamada banda de visibilidad (BoS,

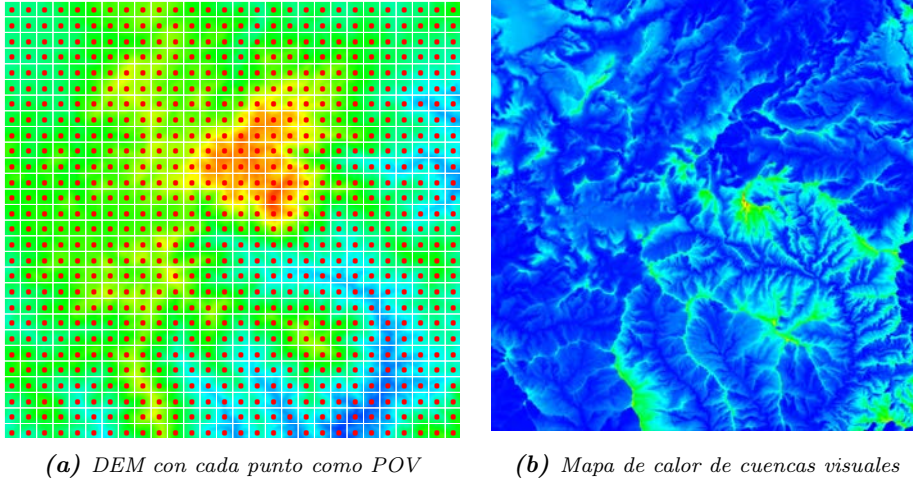


Figura B.7: Ilustración del problema de la cuenca visual total. (a) presenta un DEM discretizado con cada punto como POV donde por claridad no se dibuja ningún sector por POV, aunque en la práctica se considerarían todos los sectores para cada POV. Los valores mínimos de elevación se muestran en azul, mientras que las celdas rojas representan máximos. (b) muestra el mapa de calor de cuencas visuales resultante donde se muestran los puntos con mayor visibilidad en rojo y los de menor en azul.

Band of Sight). En este enfoque, la distancia al eje determina el número de puntos incluidos en la BoS [118]. El máximo aprovechamiento de la memoria durante el procesamiento se logró mediante la reutilización de los puntos contenidos en la lista y la obtención de la cuenca visual para cada punto alineado en sectores. Sin embargo, este algoritmo tiene importantes limitaciones:

- Para un POV dado, el análisis de los puntos dentro de la BoS se realiza de forma secuencial ya que es imposible saber si un punto objetivo es visible sin conocer el estado del anterior.
- La implementación de la reutilización de datos de la BoS produce una sobrecarga significativa causada por la selección de los puntos para cada dirección.
- No es apropiado para la implementación en sistemas de alto rendimiento como GPUs y arquitecturas multinúcleo porque el paralelismo está limitado a nivel de sector.

B.5.3. Metodología

B.5.3.1. Reutilización de los datos

La reutilización de datos es clave para la optimización del cálculo de la cuenca visual total, por lo que primero se introduce la estructura que gestionará este proceso. Esta estructura, también denominada *Band of Sight* (BoS), sirve como base para el proceso de reestructuración del DEM para cada POV y sector. En este caso, la BoS se utiliza para encontrar los puntos más cercanos a la línea de visión para cualquier POV de referencia y sector determinado. Es por ello que la elección de la anchura adecuada para esta estructura es vital para mejorar la localidad de los datos en el acceso a la memoria.

Las Figuras B.8a y B.8b muestran dos anchos de BoS de $2,5\sqrt{N}$ y \sqrt{N} , respectivamente, considerando un sector $s = 45^\circ$ por simplicidad. El amplio estudio

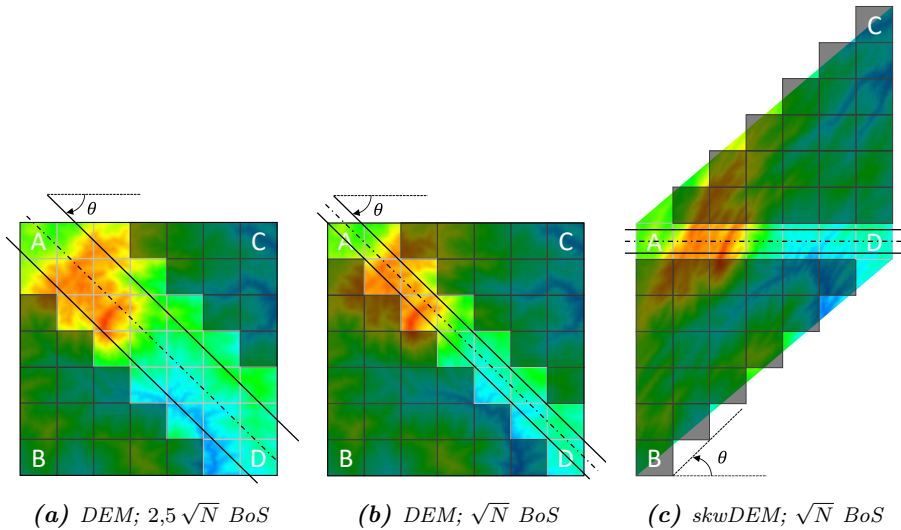


Figura B.8: Tres ejemplos de estructuras de banda de visibilidad (BoS) en el plano, considerando un sector $s = 45^\circ$ para simplificar. Las celdas de la malla en color oscuro no se utilizan para el cálculo de la visibilidad. (a) y (b) muestran dos anchos de BoS diferentes de $2,5\sqrt{N}$ y \sqrt{N} , respectivamente, con la misma disposición en el DEM $dim_y \times dim_x$; mientras que (c) presenta la reestructuración del skwDEM $2 \cdot dim_y \times dim_x$ considerando un ancho de BoS de \sqrt{N} . En aras de la claridad, las etiquetas A-D están situadas en los puntos de esquina del DEM para que se pueda visualizar el proceso de reestructuración. Obsérvese que sólo se muestra una BoS.

estadístico realizado en [110] demuestra que la anchura de esta estructura no es un factor determinante, siempre que sea del orden de \sqrt{N} . Por lo tanto, la propuesta de sDEM utiliza esta anchura de BoS para abordar el problema de la repetición de datos.

B.5.3.2. Reorganización de la malla de puntos

Una vez fijada la anchura de la BoS, se realiza la reubicación completa de los datos para transformar el DEM (Figura B.8b) en el skwDEM (Figura B.8c). Este último es un nuevo DEM cuya forma está sesgada en menor o mayor medida según la anchura de la BoS. La estructura skwDEM permite el aprovechamiento del paralelismo existente sin afectar negativamente a la precisión de los resultados en base a las siguientes consideraciones:

- Se aplica el método de barrido de Stewart [111] que establece que durante el procesamiento el bucle exterior itera sobre los sectores y el interior sobre los puntos del DEM. Es el único modelo que garantiza la reutilización de los datos alineados en todas las direcciones.
- Dado un sector, se construyen simultáneamente todas las posibles bandas visuales paralelas que atraviesan el DEM. Se aplica el método de interpolación basado en una versión simplificada del algoritmo de líneas de Bresenham [131], que se utiliza habitualmente para la rasterización de líneas. Este algoritmo se eligió por su alta velocidad, manteniendo al mismo tiempo suficiente fidelidad al problema considerado.
- Para cada sector, la reubicación se aplica una sola vez a todo el DEM. Por ejemplo, en el caso particular de considerar 180 sectores, la reubicación de los datos se realiza 180 veces, y siempre antes de iniciar el cálculo de la visibilidad. Así, la reubicación sólo depende del sector seleccionado. Otra ventaja es que este método es especialmente apropiado para el procesamiento en la GPU ya que su objetivo es reducir al máximo las estructuras condicionales, evitando la conocida penalización por divergencia de hilos.

En cuanto al método de reubicación, en la Figura B.9 se muestra la disposición escogida de filas y columnas para construir la estructura del skwDEM. Los datos del DEM situados en las mismas latitudes se almacenan de forma contigua en memoria (Figura B.9a); es decir, el bucle exterior itera de norte a sur, mientras que el bucle interior lo hace de oeste a este. Utilizando el método de interpolación, todos los segmentos paralelos del DEM (Figura B.9a) se proyectan en la

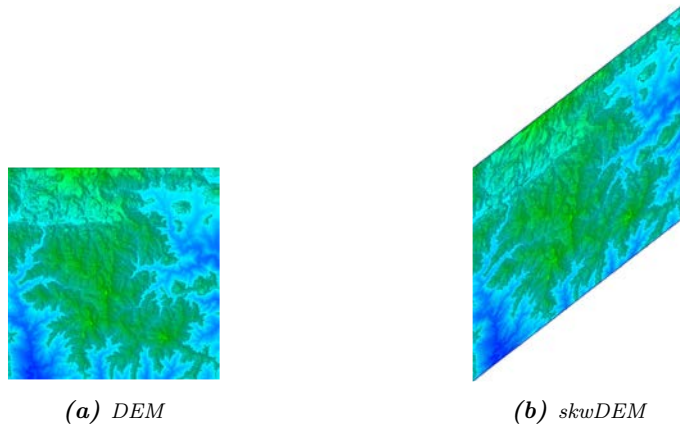


Figura B.9: El DEM y el resultado obtenido del proceso de reubicación de datos propuesto considerando el sector $s = 45^\circ$, en aras de la simplicidad. El DEM se corresponde con la zona del Parque Natural de los Montes de Málaga (Málaga, España). (a) presenta el DEM de entrada, (b) muestra el skwDEM utilizado en este trabajo.

estructura skwDEM (Figura B.9b) para que el número de elementos no nulos de ambas estructuras coincida. En este conjunto de datos reorganizado, a diferencia del original, todos los puntos de un sector dado (45° a modo de ejemplo) se colocan en la misma fila y, por lo tanto, los accesos a la memoria se realizan de forma secuencial aumentando la localidad y así el rendimiento.

B.5.3.3. Cálculo de la visibilidad

Dados los detalles de las principales estructuras BoS y skwDEM, la metodología de sDEM puede dividirse en los siguientes pasos (Figura B.10):

1. Para cada sector $s \in [0, ns/2]$, hacer:
 - a) Crear el $2 \cdot dim_y \times dim_x$ skwDEM, que es único para cada sector, a partir del $dim_y \times dim_x$ DEM y s .
 - b) Calcular los límites horizontales (A, D) y verticales (B, C) de la estructura skwDEM, que dependen del sector s .
 - c) Sea $POV_{i,j}$ el punto de vista elegido con coordenadas i y j . Para cada punto $POV_{i,j} \in skwDEM$ con $i \in [A, D]$ y $j \in [B, C]$, hacer:
 - 1) Calcular la cuenca visual lineal considerando los sectores s y $s + 180^\circ$, es decir, analizar a la derecha y a la izquierda los puntos de

- la fila (de ahí el término lineal) a la que pertenece $POV_{i,j}$ en el $skwDEM$.
- 2) Acumular el resultado de la cuenca visual en una nueva estructura llamada $skwVS$, que tiene una forma similar a la estructura $skwDEM$.
 - d) Transformar $skwVS$ en VS (cuenca visual sobre el DEM) deshaciendo las operaciones realizadas en el paso 1a. Este procedimiento considera el teorema de Pappus y también corrige la deformación introducida por el $skwDEM$.
 - e) Acumular los resultados en VS .

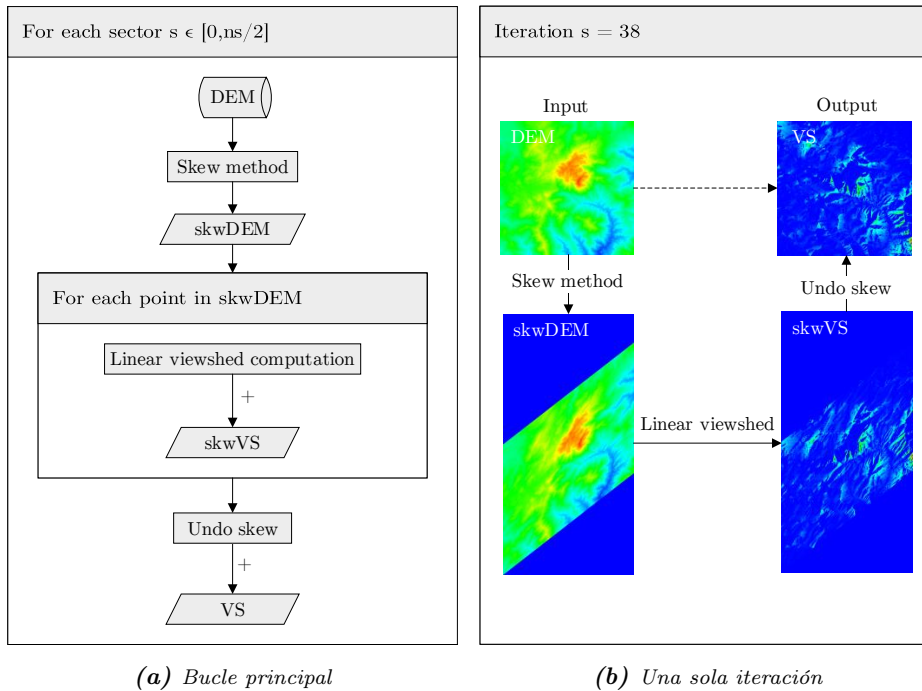


Figura B.10: Diagrama de flujo de $sDEM$ en el cálculo de la cuenca visual total que muestra (a) los pasos dentro del bucle principal encargado de recorrer todos los sectores y (b) los resultados obtenidos de una sola iteración del mismo bucle. En este último, los valores nulos y bajos se muestran en azul, mientras que las celdas rojas representan los valores máximos. El símbolo matemático $(+)$ representa el proceso de acumulación.

El cálculo de la cuenca visual en cada sector es una tarea vergonzosamente paralela porque el cálculo de cada uno de ellos es independiente. Esto reduce el problema de la cuenca visual total al análisis de $ns/2$ sectores considerando todos los puntos del skwDEM. Además, al reubicar los datos antes de realizar el cálculo de la cuenca visual para un sector determinado, sDEM asegura que cada BoS que necesita cada punto (como POV) ha sido previamente construido e incluido en el skwDEM.

Particularizando, cada fila del skwDEM corresponde a la BoS estática de cada punto incluido en ella para un sector determinado; es decir, si un punto es un POV, el resto de los puntos de su fila formarán la BoS para ese punto y sector. Como el skwDEM depende del sector, sólo tiene que reconstruirse $ns/2$ veces. En cambio, la estructura de BoS utilizada en [110] debe reconstruirse para cada punto y sector, lo que corresponde a $N \cdot ns$ veces.

B.5.3.4. Implementación multi-GPU

Aunque las implementaciones desarrolladas para plataformas mononúcleo y multinúcleo logran excelentes resultados (más adelante en la Sección B.5.4), se utilizaron principalmente como puntos de partida para el desarrollo del algoritmo final que se ejecuta en sistemas multi-GPU.

Dado que el problema de la cuenca visual total tiene ciertas similitudes con los problemas de procesamiento de matrices, la propuesta para acelerar este cálculo se centra en aprovechar el paralelismo intrínseco de este procedimiento mediante el uso de GPUs. En la práctica, los $ns/2$ sectores a procesar se distribuyen entre todos los dispositivos disponibles de forma que cada dispositivo se encarga de procesar un número similar de sectores, que viene dado por la programación elegida. Entonces, cada dispositivo lanza secuencialmente tres kernels para obtener la cuenca visual de todos los puntos del DEM para los sectores correspondientes.

De este modo, cada dispositivo contiene resultados parciales de las cuencas visuales, que son más tarde acumulados en el sistema huésped en una etapa final para obtener la cuenca visual total. Este método de acumulación en el huésped se utiliza para evitar las dependencias entre hilos mientras se realiza el cálculo de las cuencas visuales.

B.5.4. Experimentos y resultados

En esta sección se evalúa el rendimiento de sDEM con respecto a los sistemas de información geográfica (GIS) y el algoritmo más destacado de la bibliografía.

B.5.4.1. Diseño de los experimentos

Para los experimentos, se utilizan dos sistemas diferentes según sus requisitos:

- **SO Windows:** Windows 10 con una CPU Intel(R) Core(TM) i5-6500 a 3,20GHz con 4 núcleos (4 hilos) y 8GB de RAM DDR4.
- **SO Linux:** Ubuntu 16.04.5 LTS con un Intel(R) Xeon(R) E5-2698 v3 a 2,30 GHz con 16 núcleos (32 hilos) y 256 GB de RAM DDR4, junto con cuatro GPUs GTX 980 Maxwell con 2048 núcleos CUDA, 16 SMs, 1,12GHz y 4GB GDDR5 cada una.

El experimento presentado en la Sección B.5.4.2 se ejecuta en el sistema operativo (SO) Windows porque las aplicaciones GIS suelen estar desarrolladas para este SO; mientras que el experimento descrito en la Sección B.5.4.3 se ejecuta en el SO Linux para obtener una medida óptima del rendimiento computacional. En cuanto a los datos de entrada, se consideraron tres DEMs del Parque Natural de los Montes de Málaga (Málaga, España), cada uno con 10 metros de resolución y diferentes extensiones (Tabla B.8)

En cuanto a la implementación, se utiliza la API OpenMP para permitir la ejecución en paralelo del análisis de visibilidad para cada sector seleccionado, empleando programación dinámica ya que ha demostrado obtener el mejor rendimiento. Los códigos del huésped se compilan utilizando el compilador de código abierto g++ 5.4 con los indicadores de optimización ffast-math, fopenmp y O2. Los archivos CUDA utilizan el compilador NVIDIA NVCC de las herramientas de compilación CUDA V10.0.130. La implementación multinúcleo de la propuesta se lanza con el máximo número de hilos disponibles en el sistema, del mismo modo que las implementaciones de una y varias GPU se configuran para que los dispositivos funcionen a pleno rendimiento.

Tabla B.8: DEMs utilizados como datos de entrada en los experimentos.

DEM		UTM		
Base de datos	Dimensión	Zona ^a	Este	Norte
DEM10m-2k	2000x2000(10m)	30S	0310000 mE	4070000 mN
DEM10m-4k	4000x4000(10m)	30S	0360000 mE	4100000 mN
DEM10m-8k	8000x8000(10m)	30S	0360000 mE	4140000 mN

^aDesignador de banda de latitud.



Las aplicaciones GIS consideradas en esta comparación son Google Earth Pro 7.3, QGIS-GRASS 3.10.2 y ArcGIS 10.7, concretamente la extensión *Spatial Analyst* que contiene las herramientas *Viewshed 2* (VS-2), *Viewshed* (VS) y *Visibility* (VI).

B.5.4.2. Comparación con las aplicaciones GIS

Este experimento evalúa el rendimiento computacional de sDEM y de las aplicaciones GIS más utilizadas para resolver el problema de la cuenca visual múltiple. En particular, el objetivo es calcular la cuenca visual múltiple considerando 10 POVs localizados aleatoriamente en el DEM10m-2k, empleando un generador de números aleatorios para determinar las coordenadas.

La Figura B.11 junto con la Tabla B.9 muestran el tiempo que cada aplicación necesita para completar el cálculo de la cuenca visual múltiple. Se utilizaron implementaciones secuenciales para sDEM, QGIS-GRASS y Google Earth; mientras que ArcGIS tenía que ejecutarse utilizando todos los núcleos disponibles. Además, el tiempo de ejecución obtenido con Google Earth es el resultado de extrapolar

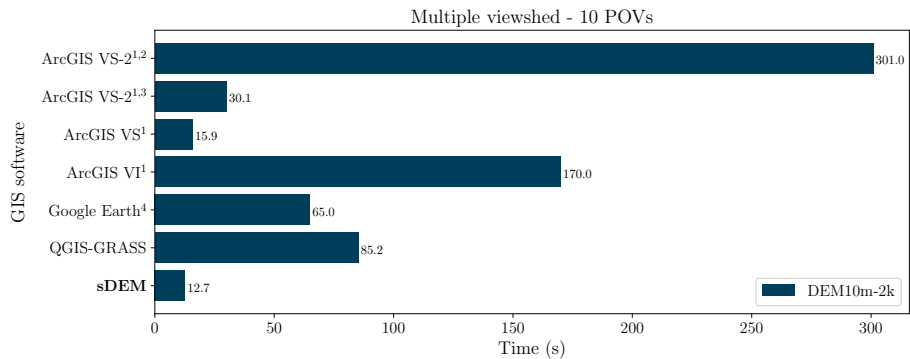


Figura B.11: Comparación de rendimiento computacional entre sDEM y las aplicaciones GIS más conocidas para calcular la cuenca visual múltiple, considerando 10 POVs localizados aleatoriamente en el DEM10m-2k. Se considera la ejecución en serie para aquellos programas que no se indica lo contrario. VS, VS-2 y VI son las herramientas de ArcGIS Spatial Analyst Viewshed, Viewshed 2 y Visibility, respectivamente. QGIS-GRASS utiliza el módulo r.viewshed. ¹ ejecución multihilo requerida utilizando el máximo número de núcleos disponibles. ² utilizando el parámetro PERIMETER_SIGHTLINES. ³ utilizando el parámetro ALL_SIGHTLINES. ⁴ Google Earth no tiene capacidad de cálculo de cuenca visual múltiple/total, entonces se ha multiplicado el tiempo medio de cálculo de la cuenca visual singular por el número de POVs considerados.

Tabla B.9: *Análisis de los resultados de la comparación de rendimiento computacional entre sDEM y las aplicaciones GIS más conocidas (Figura B.11). Se ha utilizado el tiempo de ejecución de sDEM como referencia para los cálculos.*

Aplicación GIS	Tiempo de cálculo (s)	Diferencia de tiempo (s)
ArcMap SA-2 ^{1,2}	301,0	288,3
ArcMap SA-2 ^{1,3}	30,1	17,4
ArcGIS VS ¹	15,9	3,2
ArcGIS VI ¹	170,0	157,3
Google Earth ⁴	65,0	52,3
QGIS-GRASS	85,2	72,5
sDEM	12,7	-

el cálculo de la cuenca visual singular al caso actual de 10 POVs, ya que esta aplicación no soporta esta operación.

En este experimento, sDEM supera a todos los programas de GIS analizados: ArcGIS VS-2 con dos configuraciones diferentes (23,7x, 2,4x), ArcGIS VS (1,3x), ArcGIS VI (13,4x), Google Earth (5,1x) y QGIS-GRASS (6,7x). No obstante, hay que señalar que aunque sDEM consigue resultados muy significativos en este experimento, la mayor ganancia se daría en el cálculo de la cuenca visual total.

B.5.4.3. Comparación con el algoritmo de referencia

sDEM y el algoritmo TVS [110] son los únicos algoritmos en la literatura capaces de realizar el cálculo de la cuenca visual total en grandes conjuntos de datos sin llevar a cabo reducciones previas de la carga de trabajo. Se comparan las versiones en serie, de varios hilos, de una GPU y de varias GPU para sDEM con la implementación en serie del algoritmo TVS. Además, tres escenarios diferentes son utilizados para evaluar el rendimiento de ambos algoritmos, todos ellos enfocados al problema de la cuenca visual total.

- Cuenca visual total considerando un sector aleatorio

Se analiza el rendimiento computacional de sDEM comparándolo con el algoritmo TVS en el cálculo de la cuenca visual total considerando un único sector aleatorio, donde se selecciona el sector 10°. En la Figura B.12 y en las Tablas B.10 y B.11 se muestran las curvas de aceleración y los resultados de rendimiento (POVs procesados por segundo) utilizando tres DEMs.



sDEM supera al algoritmo TVS logrando, en el mejor de los casos, una aceleración máxima de hasta 233,5x utilizando la implementación de 1-GPU sobre el DEM10m-4k. Los resultados de rendimiento muestran que esta variable es de aproximadamente $2,11 \cdot 10^7$ POV/s para la misma implementación sobre el DEM10m-2k, en comparación con el valor de $1,18 \cdot 10^5$ POV/s alcanzado por el algoritmo TVS en serie.

No se han tenido en cuenta las implementaciones multi-GPU debido a la escasa carga de trabajo existente al distribuir un sector entre más de un dispositivo.

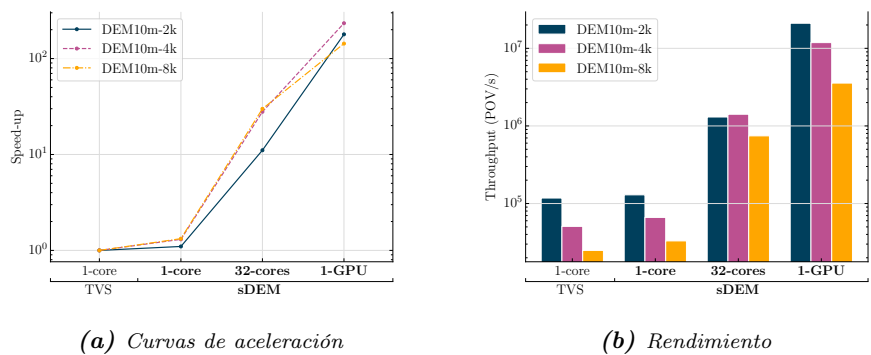


Figura B.12: Curvas de aceleración y resultados de rendimiento para el algoritmo TVS [110] y las diferentes implementaciones de la propuesta sDEM utilizando plataformas de un solo núcleo, multinúcleo, y una GPU para calcular la cuenca visual total considerando un sector seleccionado aleatoriamente, $s = 10^\circ$ (tamaño BoS de $dim \times dim$ puntos para el algoritmo TVS). Cada color corresponde a un conjunto de datos concreto. Se utiliza escala logarítmica.

Tabla B.10: Tiempo de ejecución y resultados de aceleración (Acl) de sDEM y TVS en el cálculo de la cuenca visual total considerando un sector aleatorio $s = 10^\circ$ (datos extraídos de la Figura B.12a).

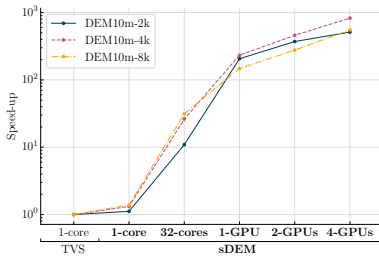
Algoritmo	Plataforma	DEM10m-2k		DEM10m-4k		DEM10m-8k	
		Tiempo (s)	Acl	Tiempo (s)	Acl	Tiempo (s)	Acl
TVS	1-núcleo	33,99	-	315,26	-	2571,66	-
sDEM	1-núcleo	30,92	1,1	241,68	1,3	1939,99	1,3
	32-núcleos	3,07	11,1	11,31	27,9	85,96	29,9
	1-GPU	0,19	178,9	1,35	233,5	17,95	143,3

Tabla B.11: Resultados de rendimiento (Thp) de *sDEM* y TVS en el cálculo de la cuenca visual total considerando un sector aleatorio $s = 10^\circ$ (datos extraídos de la Figura B.12b divididos por 10^5).

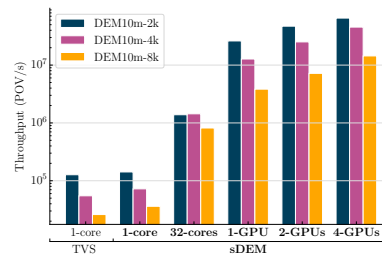
		DEM10m-2k	DEM10m-4k	DEM10m-8k
Algoritmo	Plataforma	Thp (POV/s)	Thp (POV/s)	Thp (POV/s)
TVS	1-núcleo	1,18	0,51	0,25
sDEM	1-núcleo	1,29	0,66	0,33
	32-núcleos	13,03	14,15	7,45
	1-GPU	210,53	118,52	35,65

- Cuenca visual total considerando valores medios

En este experimento, a diferencia del anterior, se selecciona el rango de dirección desde 0 hasta 45° para obtener valores medios por sector. Esto se debe a que las ejecuciones en serie de TVS y *sDEM* requerían varias semanas para completarse si se utilizaba un rango mayor, por ejemplo, hasta los 180° . Además, los resultados dentro de este rango son representativos y pueden extrapolarse a cualquier rango objetivo. En la Figura B.13 y las Tablas B.12 y B.13 se recogen las curvas de aceleración, el rendimiento alcanzado y los resultados de tiempo de ejecución.



(a) Curvas de aceleración



(b) Rendimiento

Figura B.13: Curvas de aceleración y resultados de rendimiento para el algoritmo TVS [110] y las diferentes implementaciones de la propuesta *sDEM* utilizando plataformas de un solo núcleo, multinúcleo, una GPU, y multi-GPU para calcular la cuenca visual total considerando los sectores $0^\circ < s < 45^\circ$ (tamaño BoS de $\dim x$ puntos para el algoritmo TVS). Cada color corresponde a un conjunto de datos concreto. Se utiliza escala logarítmica.

Tabla B.12: Tiempo de ejecución y resultados de aceleración (Acl) de sDEM y TVS en el cálculo de la cuenca visual total considerando los sectores $0^\circ < s < 45^\circ$ (datos extraídos de la Figura B.13a).

		DEM10m-2k		DEM10m-4k		DEM10m-8k	
Algoritmo	Plataforma	Tiempo (s)	Acl	Tiempo (s)	Acl	Tiempo (s)	Acl
TVS	1-núcleo	1417,61	-	13121,14	-	110671,86	-
sDEM	1-núcleo	1268,50	1,1	9951,31	1,3	79642,97	1,4
	32-núcleos	130,02	10,9	501,34	26,2	3526,20	31,4
	1-GPU	6,88	206,0	56,55	232,0	750,75	147,4
	2-GPUs	3,82	371,1	28,60	458,8	400,18	276,6
	4-GPUs	2,77	511,8	15,86	827,3	200,00	553,4

Tabla B.13: Resultados de rendimiento (Thp) de sDEM y TVS en el cálculo de la cuenca visual total considerando los sectores $0^\circ < s < 45^\circ$ (datos extraídos de la Figura B.13b divididos por 10^5).

		DEM10m-2k	DEM10m-4k	DEM10m-8k
Algoritmo	Plataforma	Thp (POV/s)	Thp (POV/s)	Thp (POV/s)
TVS	1-núcleo	1,27	0,55	0,26
sDEM	1-núcleo	1,42	0,72	0,36
	32-núcleos	13,84	14,36	8,17
	1-GPU	261,63	127,32	38,36
	2-GPUs	471,20	251,75	71,97
	4-GPUs	649,82	453,97	144,00

Los mejores casos estudiados muestran que la máxima aceleración alcanzada es de 827,3x con la implementación de 4-GPUs respecto al algoritmo TVS sobre el DEM10m-4k. Los resultados de rendimiento muestran que esta variable tiene un valor de aproximadamente $6,50 \cdot 10^7$ para la misma implementación sobre el DEM10m-2k, frente al valor de $1,27 \cdot 10^5$ del algoritmo TVS en serie.

- Mapa de la cuenca visual total generado con sDEM

El resultado final del mapa de la cuenca visual total del Parque Natural de los Montes de Málaga (Málaga, España) obtenido mediante el algoritmo sDEM se presenta en la Figura B.14. No se han encontrado diferencias sustanciales tras



analizar los valores de las diferencias absolutas y relativas al comparar los resultados obtenidos con TVS y sDEM. Sobre el DEM10m-2k, la diferencia absoluta encontrada es de un 1,18 %, mientras que la diferencia relativa es de 4,21 %. Todos estos valores están dentro de los límites recomendados en este campo [130].

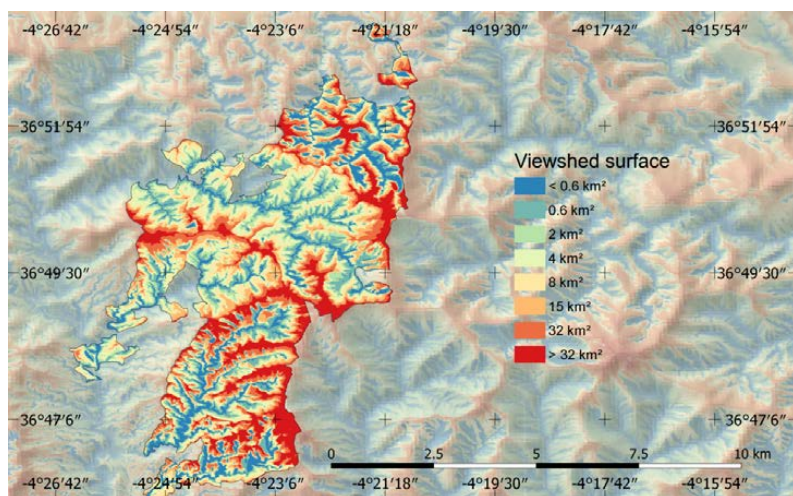


Figura B.14: Mapa de la cuenca visual total del Parque Natural de los Montes de Málaga y su entorno en la provincia de Málaga (España) generado con sDEM.

B.6. VPP: Planificación de Trayectorias con Máxima Visibilidad

El uso de vehículos aéreos no tripulados (UAV, del inglés *Unmanned Aerial Vehicles*) para tareas de teledetección ha crecido enormemente en los últimos años. Algunas de ellas son la vigilancia medioambiental y civil, la gestión de catástrofes y la lucha contra los incendios forestales. En todas estas situaciones es necesaria una respuesta rápida y temprana y, por ello, hay que detectar el peligro a tiempo. Esta condición hace que sea muy importante maximizar el terreno monitorizado durante el vuelo, sobre todo cuando la zona a vigilar es irregular, amplia y con muchos puntos ciegos. En esta situación, los últimos algoritmos de cálculo de la cuenca visual total pueden ser de gran ayuda. Estos algoritmos pueden utilizarse para analizar áreas extensas y generar trayectorias alternativas que proporcionen una monitorización total a partir de los datos obtenidos del análisis de visibilidad.

Este capítulo muestra cómo el cálculo de la cuenca visual total es una herramienta de valor incalculable para la generación de trayectorias capaces de proporcionar la máxima visibilidad posible durante el vuelo. Para demostrarlo, introducimos una nueva heurística llamada planificación de trayectorias basada en la visibilidad (VPP, *Visibility-based Path Planning*) que ofrece una solución diferente al problema de planificación de trayectorias. VPP puede utilizarse para generar trayectorias seguras desde las que vigilar zonas difíciles en terrenos complejos utilizando la cámara de un solo UAV. En este contexto, se ha elegido la prevención de incendios forestales como caso de estudio. VPP es capaz de encontrar con éxito varias trayectorias que proporcionan una visibilidad completa a partir de los datos obtenidos de un análisis exhaustivo de la visibilidad que expone las zonas más ocultas.

Por otro lado, también establecemos la dirección de la cámara del UAV para cada punto de paso incluido en el recorrido, evitando la superposición de zonas ya monitorizadas. Las imágenes capturadas durante el vuelo podrían ser analizadas en tiempo real para alertar a las autoridades competentes en caso de incendio. Todo lo comentado será evaluado considerando el Parque Natural de los Montes de Málaga (Málaga, España) que tiene una superficie aproximada de 48 km^2 .

B.6.1. Planificación de trayectorias

La planificación de trayectorias, también conocida como planificación del movimiento, es un problema clásico en robótica que ha ido ganando importancia desde mediados del siglo pasado (Figura B.15). Se trata de un problema de optimización multiobjetivo que trata de encontrar la trayectoria óptima o casi óptima para que el vehículo se desplace desde un punto inicial A hasta un punto objetivo B en base a la optimización de una o varias de las variables implicadas (por ejemplo, el tiempo de recorrido) y evitando obstáculos.

La mayoría de las investigaciones realizadas hasta la fecha para el cálculo de trayectorias utilizando UAVs se han centrado en evitar los obstáculos durante el recorrido, localizados de forma estática o dinámica en el entorno. Estos algoritmos son muy útiles para entornos dinámicos, como por ejemplo durante vuelos de vigilancia sobre zonas militares y vuelos de reconocimiento a baja altura en presencia de edificios y otros obstáculos; sin embargo, no suelen ser adecuadas para la vigilancia de incendios forestales donde el entorno puede considerarse estático. Esto se debe a que no se producen cambios significativos a lo largo del tiempo y la única restricción para la altitud del vuelo viene dada por la normativa vigente.

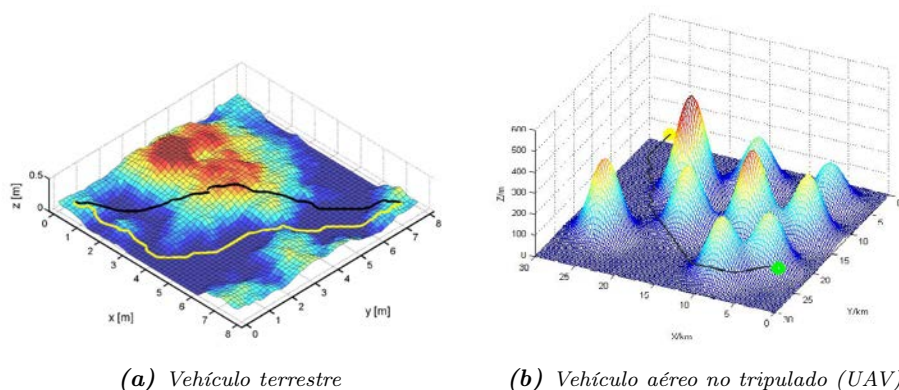


Figura B.15: Dos estrategias diferentes de planificación de trayectorias en zonas de terreno complejo en función del medio de transporte por el que se desplaza el vehículo: terrestre y aéreo [142, 143].

Son pocos los enfoques de planificación de trayectorias para UAVs que se centran en maximizar la cobertura visual con el objetivo de detectar la aparición de incendios en zonas forestales y, en la práctica, suelen tener los siguientes problemas:

- Ninguna de las soluciones propuestas hasta ahora tiene en cuenta la visibilidad desde el conjunto de puntos iniciales para los cálculos posteriores. Estos incluirían, como mínimo, los resultados de visibilidad desde los puntos de partida y de llegada. Estos deberían ser datos iniciales de los métodos heurísticos utilizados para resolver el problema de planificación de trayectorias.
- Las alternativas que utilizan la discretización del territorio en subregiones (cuadrículas) para calcular los puntos del recorrido tienen el inconveniente de obligar al UAV a volar sobre cada una de ellas. De esta forma, no se tiene en cuenta si una determinada subregión ya ha sido cubierta desde puntos adyacentes.
- Muchos algoritmos consideran visibilidad restringida, es decir, el área máxima de terreno capturado por la cámara está limitado por un círculo preestablecido de radio fijo justo debajo del UAV. Por lo tanto, el UAV puede visitar muchas ubicaciones redundantes dentro del territorio, a pesar de disponer de una cámara de alta resolución. Estos enfoques normalmente están motivados por alguna restricción de rendimiento computacional.

- Todos estos enfoques pueden ser válidos para territorios pequeños, pero difícilmente aplicables en escenarios grandes y complejos ya que no realizan un análisis exhaustivo de la visibilidad en la zona.

En el presente trabajo, se incluye el análisis de la cuenca visual total (sin restricciones de visibilidad) al problema de la planificación de trayectorias en grandes regiones de orografía compleja. En este sentido, nuestro enfoque no requiere la aplicación de una técnica de discretización del espacio más allá de la proporcionada por el DEM.

B.6.2. Metodología

B.6.2.1. Caso práctico: Parque Natural Montes de Málaga

Desgraciadamente, los incendios forestales son cada vez más frecuentes y ponen en peligro nuestros ecosistemas. Cada año arden millones de hectáreas de bosque, lo que supone un coste considerable para los países afectados y daña el medio ambiente [167]. La detección precoz de los incendios es de vital importancia para minimizar las consecuencias y, por ello, se han invertido importantes esfuerzos en la detección de los incendios forestales antes de que sean incontrolables.

Esta sección introduce VPP, una nueva heurística para abordar eficazmente el problema de la monitorización de grandes regiones de orografía compleja a partir de un análisis exhaustivo de la visibilidad. A lo largo de los siguientes apartados se mostrarán algunas capturas de los resultados parciales obtenidos con VPP. La simulación se ha realizado considerando una zona especialmente afectada por el problema de los incendios en los últimos 10 años: el Parque Natural de los Montes de Málaga [168, 169, 170]. Esta zona está situada en el sur de España y tiene un importante interés ambiental con un relieve extremadamente variado, lo que provoca notables dificultades para su monitorización. Declarado espacio natural protegido en 1989, este parque localizado cerca de Málaga capital (España) es una región geográfica de 4 800 hectáreas de extensión con zonas que van desde los 80 metros sobre el nivel del mar hasta algo más de 1 000 metros. Todos estos hechos hacen que el parque natural sea adecuado para presente análisis.

B.6.2.2. Heurística propuesta para generar la trayectoria

En primer lugar, para utilizar la heurística VPP hay que fijar dos umbrales iniciales de diseño α y γ . Cada uno de estos umbrales está relacionado con un

porcentaje de cobertura visual (Cv) a alcanzar para la región de interés: α representa el porcentaje mínimo de área que debe cubrirse como resultado del proceso; γ indica el aumento porcentual mínimo de cobertura de área que debe producirse entre dos mediciones. Particularizando, estas variables tendrán un valor de $\alpha = 90\%$ y $\gamma = 5\%$, obtenidos tras realizar varias pruebas de optimización. En otros casos, es posible que haya que ajustar estos valores para los entornos en función del objetivo y el relieve de la región.

Por otro lado, los puntos de la trayectoria se van añadiendo secuencialmente a los ya presentes indicados por las condiciones iniciales del problema. Se distinguen dos tipos según su disposición a lo largo de la trayectoria:

- **Puntos de paso primarios (W_p):** este grupo lo conforman las ubicaciones iniciales de seguimiento obligatorio y también las encontradas después de haber aplicado la heurística VPP. Estos puntos se conectan mediante líneas rectas para formar el camino.
- **Puntos de paso secundarios (W_s):** a este conjunto pertenecen aquellos puntos situados en la línea recta que une dos puntos de paso primarios consecutivos. El UAV también puede fotografiar el terreno desde este tipo de localizaciones.

- Etapa 1: evaluar la aportación de los puntos iniciales

En nuestro caso, el análisis que podemos realizar en esta primera fase está restringido por las condiciones iniciales del problema. El UAV debe pasar por cinco puntos logísticos (L), entre los que se encuentran tres ubicaciones para el seguimiento de zonas naturales importantes y los puntos inicial y final del recorrido. Las cuencas visuales de cada uno de ellos deben ser calculadas y acumuladas para obtener una primera aproximación de la cobertura visual (Cv).¹ En concreto, se obtiene el mapa de la cuenca visual singular (SVS, *Singular Viewshed*) para cada $P_i \in W_p, i = 1...|W_p|$ de forma que, posteriormente, se sumen todos estos mapas y se forme el mapa de la cuenca visual acumulada del conjunto de puntos (PS-CVS, *Point Set Cumulative Viewshed*). Este último mapa representa la cantidad de terreno cubierto al final de esta primera fase, proporcionando una visión global interesante de la cobertura al principio del proceso.

En las siguientes figuras, utilizaremos el mapa de áreas ocultas acumuladas (CHAR, *Cumulative Hidden Areas*), que es la estructura complementaria del mapa PS-CVS, para lograr una mejor apreciación de las zonas del terreno que

¹Para este cálculo y los sucesivos, la altura del observador se fija en $h = 30\text{ m}$.

quedan sin monitorizar. El mapa CHAR destaca las zonas no monitorizadas y es el más adecuado para encontrar y definir las áreas remotas. Colocando este mapa sobre el modelo del terreno se obtiene el resultado mostrado en la Figura B.16. En las próximas etapas, añadir más puntos de paso al conjunto W_p garantizará el éxito de la monitorización de la región.



Figura B.16: Mapa de áreas ocultas acumuladas (CHAR) tras la etapa 1 de VPP que ilustra el área restante sin cubrir (capa de color púrpura) después de haber analizado las cuencas visuales de los cinco puntos logísticos iniciales incluidos en W_p . La línea blanca representa los límites del Parque Natural de los Montes de Málaga (Málaga, España).

- Etapa 2: encontrar nuevos puntos de máxima visibilidad

Naturalmente, deducimos que los puntos que proporcionan el valor máximo de cuenca visual son buenas elecciones en esta etapa. Dichos puntos se denominan POVs de máxima visibilidad (V) y se obtienen mediante la resolución del problema de la cuenca visual total empleando la herramienta M-TVS. Esta herramienta, en lugar de considerar todos los puntos del DEM, puede dejar fuera del análisis algunos puntos o zonas concretas. Esto se hace para encontrar siempre los puntos con la máxima cuenca visual sobre la zona objetivo (no vigilada), excluyendo las zonas redundantes (ya vigiladas o sin interés) [112]. Por todo ello, el hecho de incluir estos puntos progresivamente en W_p aumentará gradualmente el área cubierta de modo que en algún momento se pueda superar el umbral α .

En nuestro caso de estudio, la inclusión de las dos mejores ubicaciones de máxima visibilidad al conjunto actual de puntos ha demostrado ser la mejor

estrategia, garantizando una cobertura visual superior al 80 % en la región de interés (Figura B.17). Aunque esta inclusión al conjunto actual W_p ha mejorado la cobertura, todavía quedan muchas zonas ocultas sin monitorizar.



Figura B.17: Mapa de áreas ocultas acumulativas (CHAR) tras la etapa 2 de VPP que ilustra el área restante sin cubrir (capa de color púrpura) después de haber analizado las cuencas visuales de los cinco puntos logísticos iniciales y dos nuevos puntos de máxima visibilidad incluidos en W_p . La línea blanca representa los límites del Parque Natural de los Montes de Málaga (Málaga, España).

- Etapa 3: identificar las zonas aisladas

Como se ha explicado en la sección introductoria, no existe una solución única para el problema de la planificación de trayectorias buscando la máxima cobertura visual. Teniendo en cuenta la bibliografía reciente, algunas estrategias como las presentadas en [113] y [165] habrían garantizado que el vuelo del UAV alcanzara todas las zonas aisladas pero siguiendo un enfoque codicioso en el que el UAV se ve obligado a sobrevolar casi cualquier punto del terreno. Por el contrario, VPP proporciona una solución mejor: identificar las zonas visualmente aisladas y hacer que el UAV vuele únicamente sobre sus centroides, evitando así la necesidad de barrer toda la región. Siguiendo esta línea se proponen dos métodos para identificar dichas zonas aisladas según la perspectiva desde la que se analice:

- Desde el punto de vista del observador, definiríamos una zona como aislada si no es visible desde un número determinado de puntos de máxima visibilidad.

- Desde el punto de vista de la zona objetivo, consideraremos que una zona está aislada si la superficie del terreno que se ve desde su centro es inferior a un parámetro fijo λ . Este resultado se consigue utilizando curvas de nivel.

En este trabajo, las zonas aisladas se identifican utilizando el último método, ya que es el enfoque más objetivo. El parámetro $\lambda = 1 \text{ km}^2$ es elegido tras haber realizado varias pruebas de optimización. Con este enfoque, las áreas resultantes se encuentran más delimitadas en la mayoría de los casos, constituyendo espacios simplemente conectados que son fácilmente identificables por sus centroides (Figura B.18). Estos centroides se incluyen entonces en el conjunto actual de puntos W_p . Por último, se genera el mapa PS-CVS para comprobar si el porcentaje de cobertura del terreno está por encima del umbral α . Si se cumple esta condición, podemos pasar a la etapa final.

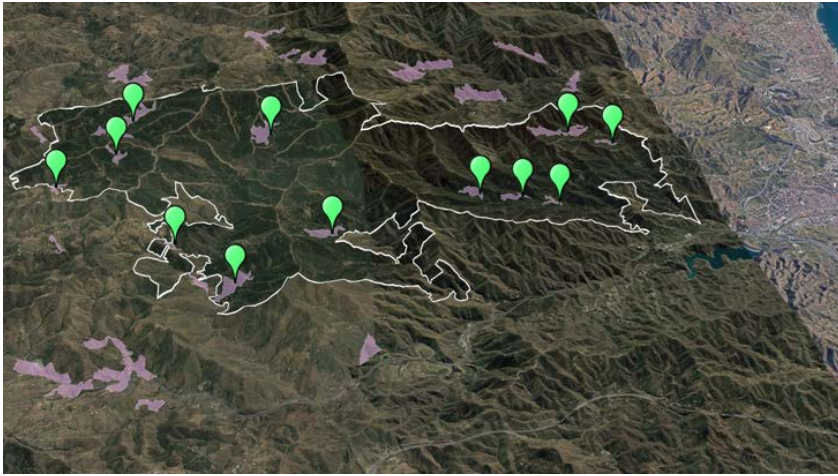


Figura B.18: Zonas aisladas identificadas mediante curvas de nivel con un área de visión inferior a $\lambda = 1 \text{ km}^2$ en la etapa 3. Sus centroides se representan con marcadores verdes dentro de los límites del Parque Natural de los Montes de Málaga (línea blanca).

- Etapa 4: generar la trayectoria de vuelo

El último paso consiste en generar la trayectoria a seguir conectando los puntos de paso primarios dentro el conjunto final W_p . Después de haber realizado las tres etapas anteriores de la heurística VPP en la zona del Parque Natural de los Montes de Málaga, los puntos de paso en W_p incluyen:

- 5 puntos logísticos que incluyen los puntos iniciales y finales del recorrido, así como tres localizaciones fácilmente accesibles de alto valor medioambiental que podrían servir como puntos de recarga para el UAV.
- 2 puntos de máxima visibilidad obtenidos tras calcular el mapa M-TVS.
- 12 centroides de las zonas aisladas ($\lambda = 1 \text{ km}^2$).

La Figura B.19 muestra el resultado de conectar todos los puntos de paso primarios considerando la zona del Parque Natural de los Montes de Málaga (Málaga, España). Este camino, con una longitud de aproximadamente 34 kilómetros, ofrece una visibilidad prácticamente completa y, por lo tanto, la zona objetivo puede ser monitorizada con éxito utilizando la cámara instalada en el UAV.



Figura B.19: Trayectoria de vuelo resultante (línea roja) generada mediante la heurística VPP en la zona del Parque Natural de los Montes de Málaga (Málaga, España). El punto logístico situado más a la derecha se corresponde con la ubicación base. El área monitorizada se representa mediante una capa blanca sobre el modelo del terreno.

Cabe señalar que el mapa PS-CVS calculado aquí puede diferir del de un vuelo real porque se ha considerado una cámara de 360 grados de apertura. Esto se analizará con más detalle en la siguiente sección donde se buscará la mejor dirección de la cámara para cada uno de los puntos de paso con la finalidad de maximizar el terreno cubierto.

B.6.2.3. Método de rechazo de la visibilidad

En la sección anterior, se ha presentado la heurística VPP para generar trayectorias de vuelo basadas en un análisis exhaustivo de la visibilidad. Ese análisis consideraba el rango completo de visibilidad para cada punto de paso primario a lo largo de la trayectoria, lo que significa que la visibilidad desde cualquier punto en cualquiera de las cuatro etapas anteriores se calculaba como si se utilizara una cámara de 360 grados. Sin embargo, en la vida real, el uso de este tipo de cámaras en UAVs no es una práctica habitual, por lo que entra en juego el parámetro denominado ángulo de visión de la cámara (AOV, *Angle of View*).

En fotografía, el AOV de la cámara delimita el ángulo de una determinada escena captada por la cámara, medida en horizontal, vertical o diagonal. Esta variable puede considerarse como el número máximo de sectores consecutivos (es decir, de ángulos) que la cámara es capaz de captar. Por lo tanto, este parámetro afecta en gran medida a la elección de la dirección de la cámara para cada punto de paso, que se denominará centro de interés (COI, *Center of Interest*).

En el problema de visibilidad que nos ocupa, entre cada dos puntos de paso primarios consecutivos, el UAV tomará una imagen cada 150 metros aproximadamente, siendo estos nuevos puntos los denominados como puntos de paso secundarios (W_s). En el caso de estudio, este método conduce a un total de 235 puntos a lo largo de la trayectoria de vuelo (W_p y W_s), por lo que el parámetro AOV de la cámara se convierte en un parámetro crucial para maximizar la cobertura de la región.

El flujo de trabajo a seguir, que puede considerarse como una única iteración del proceso sobre el conjunto total de puntos de paso, consiste en (i) procesar la cuenca visual desde el punto de paso objetivo siguiendo una técnica de optimización de memoria, (ii) obtener el mejor COI, (iii) simular el resultado de la cobertura y (iv) repetir el proceso con el siguiente punto de paso en la trayectoria.

- Encontrar los mejores COIs

Calcular el mejor COI para cada punto de paso de la trayectoria es clave para maximizar la visibilidad durante el vuelo. Una vez iniciado el vuelo, hay que evitar apuntar la cámara en una dirección en la que el terreno ya haya sido cubierto en su mayoría desde puntos anteriores de la trayectoria.

Teniendo en cuenta esta cuestión, es necesario definir una tabla de consulta (en inglés, *look-up table*) llamada sectores de anillos visibles (RS , *Ring-Sectors*) que contiene el número de puntos cubiertos desde un determinado punto de paso y

para cada sector. La Figura B.20 muestra cómo se rellena la tabla de consulta RS para un solo sector, por simplicidad. En la práctica, RS tiene tantas filas como sectores (desde $s = 0^\circ$ hasta $n_s = 360^\circ$) y columnas como la máxima distancia posible entre dos puntos (medida en celdas del DEM). Si la ubicación del objetivo es visible desde el punto de paso y aún no se ha cubierto—comprobando el valor del estado dentro de $PL-CVS$, se marca su celda correspondiente en RS . El número total de celdas marcadas por cada fila se almacena en la primera columna de RS , representando también el peso de cada sector.

Como se ha explicado anteriormente, el uso de cámaras de 360 grados en los UAVs no es una práctica habitual y sólo un número fijo de sectores consecutivos (n_{aov}) serán capturados por la cámara en cada punto de paso de la trayectoria en un proceso representado mediante la estructura AOV . Esta estructura de datos funciona como una máscara que permite el procesamiento de la visibilidad para algunos sectores, mientras que lo deshabilita en los restantes no capturados por la cámara. Así, sólo aquellos sectores consecutivos que estén marcados en AOV contribuirán al resultado de la cobertura, donde el COI es el sector central.

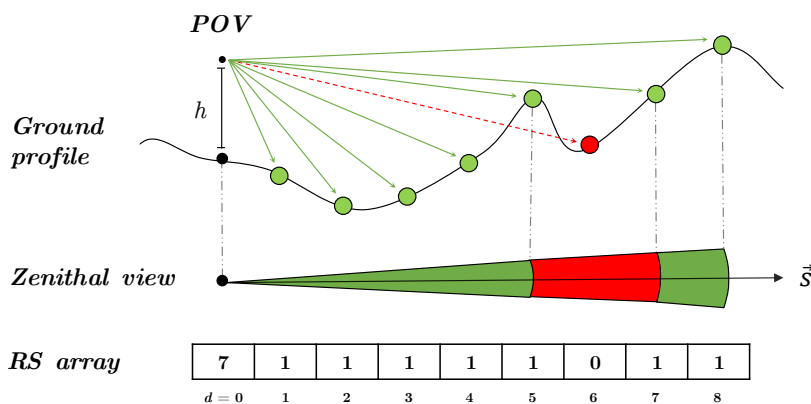


Figura B.20: Perfil del terreno, vista cenital y la estructura RS para un POV de referencia de altura h y un sector s . Los puntos visibles en el perfil del terreno se representan en verde, mientras que los puntos rojos se corresponden con ubicaciones ocultas. La vista cenital muestra el sector de anillos visibles correspondiente al perfil del suelo mostrado. En la zona inferior, se rellena la estructura RS a modo de ejemplo.

El mejor conjunto de sectores consecutivos para cada punto de paso se elige en función de sus pesos y siempre buscando maximizar la cobertura del terreno evitando el solapamiento visual. En concreto, el peso de cada sector se mide como el número total de puntos visibles en esa dirección y aún no cubiertos (previamente almacenados en la primera columna de RS , $d = 0$). Después, el conjunto de sectores consecutivos con el mayor peso global se marcará en la estructura AOV para el punto de paso en cuestión. Siguiendo un proceso iterativo, el peso global de cada posible agrupación de n_{aov} sectores consecutivos se obtiene sumando los pesos individuales de cada sector incluido en ellos, por ejemplo, sumando cada peso desde $s = 0^\circ$ hasta $s = n_{aov}$, luego desde $s = 1^\circ$ hasta $s = n_{aov} + 1^\circ$, y así sucesivamente hasta encontrar la mayor suma de pesos.

El último paso consiste en calcular los mapas $CC-SVS^2$ y $PL-CVS$ del punto de paso objetivo para obtener los resultados de cobertura del terreno individuales y acumulados, respectivamente.

- Ajuste de la dirección de la cámara considerando la variable AOV

Dada cualquier trayectoria y DEM, el flujo de trabajo para simular el comportamiento de la cámara durante el vuelo sería el siguiente:

1. Para cada punto de paso en la trayectoria de vuelo, hacer:
 - a) Rellenar RS según la visibilidad actual y el mapa $PL-CVS$.
 - b) Rellenar AOV según el conjunto de sectores consecutivos con el mayor peso global.
 - c) Para cada punto en el DEM, hacer:
 - 1) Calcular la diferencia angular (A_d) y la distancia euclídea (E_d) entre este punto y el punto de paso.
 - 2) Indexar la matriz booleana AOV utilizando A_d para comprobar si el sector actual está incluido en el ángulo de visión de la cámara desde el punto de paso actual.
 - 3) Si se cumple esta condición, hacer:
 - a' Indexar la tabla de consulta RS —utilizando A_d como índice de fila y E_d como índice de columna—para recuperar el estado del punto objetivo, es decir, si ha sido capturado por la cámara.

² $CC-SVS$ (*Camera Captured Singular Viewshed*) es la cuenca visual capturada por la cámara. Este mapa muestra la cuenca visual desde un determinado punto de paso de forma similar a la estructura SVS pero analizando únicamente los sectores marcados en la estructura AOV (sectores capturados por la cámara).

b' En caso afirmativo, se marca la celda correspondiente en el mapa *CC-SVS*.

$d)$ Acumular el resultado en el mapa *PL-CVS*.

La Figura B.21 muestra cuatro iteraciones de la metodología propuesta, donde el ángulo de visión de la cámara y el valor de la altura para el UAV se establecen en 84° y $h = 30\text{ m}$, respectivamente. Con este método, conseguimos evitar el solapamiento visual entre los puntos de paso incluidos en la trayectoria de vuelo, maximizando así el terreno cubierto a lo largo de la misma.

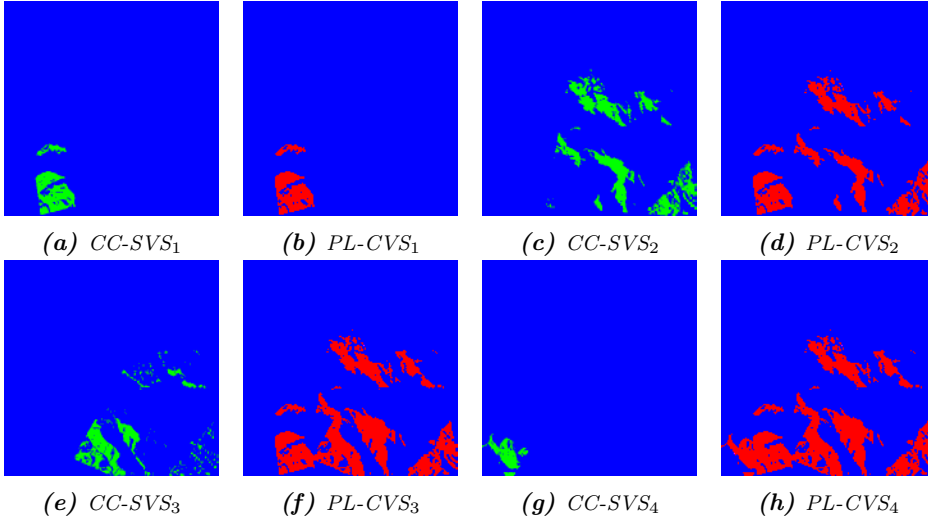


Figura B.21: Secuencia simplificada de operaciones en la que el UAV visita cada uno de los cuatro primeros puntos de paso de la trayectoria de vuelo. Los subíndices se refieren a la posición que ocupan los puntos de paso dentro de la trayectoria.

B.6.3. Experimentos y resultados

En esta sección se evalúa el rendimiento de VPP utilizando la cobertura del terreno como unidad de medida.

B.6.3.1. Diseño de los experimentos

En cuanto al problema de la monitorización, las características de la cámara empleada en el presente caso de estudio son las mismas que la instalada en el

conocido dron DJI Phantom 4 Pro [172]. Se establece el AOV de la cámara en 84° y la altura de vuelo en $h = 30\text{ m}$.

Las coordenadas UTM y las estadísticas de elevación de la zona del Parque Natural de los Montes de Málaga (Málaga, España) consideradas para este análisis se presentan en la Tabla B.14. Esta tabla refleja la abrupta orografía de la zona, que se puede apreciar debido al alto valor de la desviación estándar de la elevación respecto al valor medio.

Tabla B.14: *Análisis estadístico básico del conjunto de datos utilizado en los experimentos. Los valores se muestran en metros.*

DEM	UTM			Estadísticas de elevación			
Dimensión	Zona ^a	Este	Norte	Min.	Máx.	Rango	Media ± SD
2000x2000(10m)	30S	0370000 mE	4090000 mN	76	1041	965	560,7 ± 504,9

^aValor de la banda de latitud.

B.6.3.2. Simulación del vuelo del UAV

La planificación de trayectorias y la cobertura visual son problemas difíciles ya que dependen en gran medida de la complejidad y la diversificación de los terrenos. Nuestro caso de estudio comprende una zona amplia y variada con muchos lugares remotos/aislados en los que la vigilancia es una tarea casi imposible de realizar por observadores terrestres. Sin embargo, este problema puede solventarse utilizando un único UAV que sobrevuele la zona siguiendo una trayectoria que proporcione una visibilidad de toda la región objetivo.

En el diseño de esta trayectoria no se ha considerado la visibilidad restringida como aparece, por ejemplo, en [113], porque la tecnología actual de las cámaras permite el procesamiento de imágenes de alta definición con una pérdida de definición mínima. Tampoco se ha tenido en cuenta el problema de la evitación de obstáculos ya que la zona objetivo es un parque natural en el que no se encontrarán obstáculos entre dos localizaciones si se considera una altura de vuelo suficientemente elevada.

Una vez fijada la trayectoria y la dirección de la cámara para cada punto de paso, se puede obtener la cobertura final del terreno (Figura B.22). En ella se muestra el área cubierta por el UAV tras volar siguiendo la trayectoria y capturando imágenes de la zona del Parque Natural de los Montes de Málaga (Málaga, España). La escala de colores indica el terreno fotografiado por el UAV

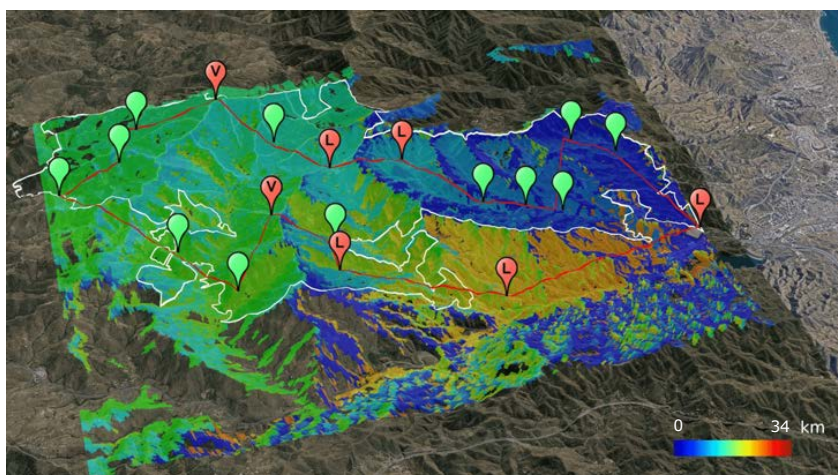


Figura B.22: Cuenca visual acumulada desde la trayectoria (PL-CVS) de la zona del Parque Natural de los Montes de Málaga (Málaga, España). La escala de colores indica el terreno fotografiado por el UAV y la distancia desde el punto de partida a la que se tomó la fotografía, teniendo la trayectoria una longitud de 34 km e incluyendo 235 puntos de paso. El punto más a la derecha se corresponde con el punto de partida y el UAV recorre la trayectoria en sentido contrario a las agujas del reloj.

y la distancia desde el punto de partida a la que se tomó la fotografía, teniendo la trayectoria una longitud de 34 km e incluyendo 235 puntos de paso. Así, si el color de una determinada zona del mapa es azul, significa que ese punto ha sido fotografiado desde los primeros kilómetros del recorrido, donde el punto logístico más a la derecha se corresponde con el punto de partida y el UAV recorre la trayectoria en sentido contrario a las agujas del reloj. En cambio, el color rojo en el mapa indica que la fotografía se ha tomado hacia el final del trayecto [174]. Como resultado final se obtiene que el vuelo de un solo UAV siguiendo la trayectoria generada por VPP logra capturar el 98,7% de la superficie del parque natural durante el vuelo.

B.7. Conclusiones y Trabajo Futuro

El análisis espacial ha adquirido una gran importancia en los últimos años a raíz del crecimiento tan significativo que ha experimentado Internet, donde la cantidad de información espacial disponible ha ido aumentando de forma progresiva con los años. Además, la mejora de los sistemas utilizados hoy en día en

términos de capacidad computacional también ha contribuido a este hecho. A lo largo de esta tesis, se han abordado tres problemas espaciales de gran relevancia, como son (i) la identificación de huellas dactilares latentes, (ii) el cálculo de la cuenca visual total y (iii) la planificación de trayectorias de máxima visibilidad.

El Capítulo B.4 presentó una novedosa metodología denominada procesamiento asíncrono para la identificación de huellas dactilares latentes (ALFI) para sistemas heterogéneos CPU-GPU. ALFI solapa y sincroniza eficientemente dos tareas que se ejecutan en diferentes dispositivos. La primera tarea, relacionada con el procesamiento de datos en el dispositivo, implica el preprocesamiento y la búsqueda de parejas de minucias similares, el cálculo de la calidad de las minucias y la obtención de agrupaciones de parejas. La segunda tarea comprende la etapa de evaluación final multihilo ejecutada en el huésped que evalúa las agrupaciones de parejas de minucias y devuelve las posibles huellas dactilares coincidentes.

En el dispositivo, la novedosa estrategia aplicada al procesamiento de datos aprovecha el paralelismo intrínseco del proceso de identificación de huellas latentes. Cada hilo del dispositivo procesa una minucia concreta de un lote de impresiones dactilares y la compara con cada minucia de la huella latente. Por todo ello, ALFI reduce con éxito los tiempos muertos en el huésped y el dispositivo, consiguiendo resultados más rápidamente entre la huella latente y las impresiones.

ALFI ha sido probado en los sistemas operativos Linux y Windows utilizando tres equipos diferentes con sistema CPU-GPU. Se utilizaron bases de datos de identificación bien conocidas como NIST SD27, NIST SD14 y NIST SD4 para probar la precisión del algoritmo propuesto en la identificación de huellas dactilares latentes. De forma adicional, se utilizaron las bases de datos de verificación FVC 2002, FVC 2004 y FVC 2006 para comprobar el rendimiento de la verificación. Los experimentos han demostrado que ALFI supera al algoritmo DMC—el mejor hasta la fecha, logrando un aumento de velocidad promedio de hasta 22x, manteniendo los resultados de precisión dentro del mismo rango. Además, considerando el mejor caso estudiado, ALFI logra un aumento de velocidad de hasta 44,7X utilizando una base de datos ampliada que contiene 387 243 huellas dactilares.

ALFI es la primera metodología para la identificación de huellas dactilares latentes que está específicamente diseñada para aprovechar al máximo todos los recursos hardware de sistemas heterogéneos CPU-GPU. Sin embargo, este algoritmo no se está utilizando en investigaciones criminales actuales debido a problemas relacionados con la patente del código fuente, aunque esperamos que este incon-

veniente se resuelva pronto para que ALFI pueda ser utilizado por las autoridades policiales en la identificación de individuos implicados en crímenes reales.

El Capítulo B.4 se relaciona con las siguientes publicaciones:

Acelerando la comparación de huellas dactilares basadas en agrupaciones deformables de minucias

A.J. Sanchez-Fernandez, L.F. Romero, and S. Tabik

In *XXIX Jornadas de Paralelismo (Jornadas SARTECO)*, Teruel, Spain, September 2018

[82] A First Step to Accelerating Fingerprint Matching based on Deformable Minutiae Clustering

A.J. Sanchez-Fernandez, L.F. Romero, S. Tabik, M.A. Medina-Pérez, and F. Herrera

In *18th Conference of the Spanish Association for Artificial Intelligence (CAEPIA 2018)*, Granada, Spain, October 2018

Published in: *Advances in Artificial Intelligence. Book Series: Lecture Notes in Computer Science*, vol. 11160, pp. 361–371, 2018

[176] Asynchronous Processing for Latent Fingerprint Identification on Heterogeneous CPU-GPU Systems

A.J. Sanchez-Fernandez, L.F. Romero, D. Peralta, M.A. Medina-Pérez, Y. Saeys, F. Herrera, and S. Tabik

IEEE Access, vol. 8, pp. 124236–124253, 2020

El Capítulo B.5 introdujo una nueva metodología denominada modelo digital de elevación sesgado (sDEM) para acelerar el análisis de la superficie del terreno. En particular, se seleccionó el problema de la cuenca visual total como caso de estudio para evaluar el rendimiento de sDEM, desarrollado desde cero y diferente a los métodos recogidos en la literatura por la forma en que se ejecutan las operaciones.

sDEM se centra en aumentar el rendimiento de los accesos a la memoria aplicando una técnica de reestructuración de datos antes de iniciar el cálculo. La reorganización propuesta hace que los puntos cercanos en el terreno estén también localizados en posiciones contiguas de memoria, pero no siguiendo las

direcciones estándar norte-sur y este-oeste, sino considerando diferentes sectores angulares desde 0° hasta 180° . Gracias a ello, el cálculo de la cuenca visual sobre cualquier terreno es mucho menos costoso computacionalmente utilizando sDEM que con los enfoques descritos en la bibliografía.

Se han propuesto diferentes versiones de sDEM para plataformas mononúcleo, multinúcleo, con una GPU y multi-GPU, junto con estudios intensivos de rendimiento donde se compara sDEM con la literatura y el software GIS actual. sDEM ha sido probado en sistemas operativos Windows y Linux utilizando dos equipos diferentes y tres DEMs compuestos por hasta 64 millones de puntos del Parque Natural de los Montes de Málaga (Málaga, España).

Las diferentes implementaciones han demostrado tener un mejor rendimiento que el software GIS más utilizado en la actualidad en lo que respecta al cálculo de la cuenca visual múltiple. Además, sDEM supera ampliamente al algoritmo más avanzado en términos de velocidad y rendimiento para los tres DEMs evaluados en el cálculo de la cuenca visual total. De hecho, el enfoque propuesto acelera este cálculo hasta 827,3x para el mejor caso estudiado, con respecto a la implementación en serie, sobre un DEM formado por 16 millones de puntos.

Dado lo anterior, sDEM abre la puerta a aprovechar al máximo los sistemas multi-GPU para la optimización de muchos algoritmos para los que nunca se habían considerado. Esto se debía a que la irregularidad en el procesamiento de los datos del terreno causaba una baja eficiencia, como ocurría en el cálculo de la cuenca visual total.

El Capítulo B.5 se ha elaborado a partir de las siguientes publicaciones:

Reorganización de matrices en algoritmos de barrido radial sobre Modelos Digitales del Terreno

A.J. Sanchez-Fernandez, L.F. Romero, S. Tabik, and G. Bandera

In *XXX Jornadas de Paralelismo (Jornadas SARTECO)*, Cáceres, Spain, September 2019

[173] A data relocation approach for terrain surface analysis on multi-GPU systems: a case study on the total viewshed problem

A.J. Sanchez-Fernandez, L.F. Romero, G. Bandera, and S. Tabik

International Journal of Geographical Information Science, pp. 1–21, 2020

El Capítulo B.6 presentó una nueva heurística denominada denominada planificación de trayectorias basada en la visibilidad (VPP) que ofrece una solución diferente al problema de planificación de trayectorias basado en la explotación de los datos de visibilidad, donde elegimos la prevención de incendios forestales como caso de estudio. VPP muestra los beneficios del cálculo de la cuenca visual total: encontramos varias trayectorias seguras para el UAV que proporcionan una visibilidad total a partir de los resultados obtenidos de un análisis de visibilidad exhaustivo que expone las zonas más ocultas. Estos resultados muestran que es posible monitorizar regiones difíciles con terrenos complejos utilizando la cámara instalada en un solo UAV, proporcionando una vigilancia del territorio más eficaz en términos de reducción de la longitud de la trayectoria y aumento de la superficie cubierta.

Como la trayectoria generada se compone de varios puntos de paso, también proporcionamos un método para determinar la dirección de la cámara para cada uno de ellos, con el objetivo de evitar la superposición de áreas visuales de lugares ya vigilados para aumentar la cantidad de terreno cubierto. Estas imágenes—tomadas a lo largo del vuelo—pueden ser analizadas en tiempo real para alertar a las autoridades locales en caso de incendio. Por último, evaluamos el rendimiento de VPP simulando el vuelo de un UAV sobre la zona del Parque Natural de los Montes de Málaga (Málaga, España). En este caso, se ha logrado una cobertura del terreno del 98,7 % del área del parque natural.

Los resultados demostraron que nuestra propuesta cubre con éxito la zona estudiada, lo que supondría una alta protección contra los incendios forestales a un coste relativamente bajo. Importantes empresas y organismos, como la Junta de Andalucía, el Ayuntamiento de Málaga y AERTEC Solutions [177] han manifestado su interés por escrito en los hallazgos de este trabajo.

Los resultados del trabajo mostrado en el Capítulo B.6 han derivado en la siguiente publicación:³

[178] VPP: Visibility-based Path Planning Heuristic for Monitoring Large Regions of Complex Terrain using a UAV Onboard Camera

A.J. Sanchez-Fernandez, L.F. Romero, G. Bandera, and S. Tabik

IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 2021

³Este artículo ha sido aceptado para su publicación en un futuro número de IEEE J-STARS.

En resumen, esta tesis ha abordado tres problemas espaciales muy relevantes siguiendo diferentes puntos de vista:

- a) En el Capítulo B.4, comenzamos analizando el comportamiento del algoritmo de identificación de huellas dactilares latentes más avanzado disponible hasta la fecha. En especial, nos centramos en encontrar puntos débiles y patrones de procesamiento que pudiéramos aprovechar y optimizar. Nos dimos cuenta de que este algoritmo no aprovechaba la heterogeneidad de los sistemas actuales, por lo que desarrollamos una alternativa basada en el procesamiento asíncrono de datos entre la CPU y la GPU especialmente diseñada para sistemas heterogéneos.
- b) Posteriormente, en el Capítulo B.5 aplicamos lo aprendido a otro problema espacial de gran interés: el cálculo de la cuenca visual total en grandes áreas. Encontramos una alternativa que mejoraba en gran medida los tiempos de ejecución al aplicar una reestructuración completa de los datos del terreno antes de procesar la visibilidad. Probamos varias alternativas basadas en la computación heterogénea, pero los enfoques que utilizan únicamente el procesamiento en la GPU superaban con creces a las implementaciones heterogéneas debido a la naturaleza matricial del problema.
- c) Como parte final de este trabajo, en el Capítulo B.6 abordamos el conocido problema de la planificación de trayectorias pero considerando la prevención de incendios en regiones forestales como caso de estudio. En especial, tratamos de encontrar la trayectoria que proporcionara la máxima visibilidad del terreno durante el vuelo del UAV. Para el desarrollo de esta heurística, utilizamos los conceptos de programación heterogénea aprendidos durante el desarrollo de los dos algoritmos anteriores. Finalmente, la heurística que resultó de este estudio es capaz de generar una trayectoria eficaz para la monitorización completa del área objetivo.

Las líneas de investigación que constituirían la continuación natural del trabajo incluido en esta tesis podrían ser las siguientes:

En lo que respecta a ALFI, el siguiente paso sería evaluar la eficiencia energética (por ejemplo, en FLOPs/vatios) comparando este enfoque y el algoritmo DMC-CC. En otras palabras, queremos estudiar cómo afecta la ganancia de velocidad durante la ejecución al consumo de energía, siempre teniendo en cuenta que el consumo de energía es una variable secundaria en el problema en cuestión. Las variables más críticas en el problema de la identificación de individuos son la velocidad de procesamiento y la precisión. Por otro lado, podríamos también estudiar si la inclusión de técnicas de aprendizaje automático en el proceso

de emparejamiento de las minucias conduciría a un aumento en la precisión de identificación proporcionada por ALFI.

Para la segunda línea de investigación relativa a sDEM, el paso natural sería lanzar una herramienta multiplataforma con fines de investigación y ampliar esta a otros tipos de análisis topográfico. Algunos ejemplos podrían ser el análisis de pendientes y elevaciones, así como cualquier otra característica que tenga necesidades de procesamiento similares. También se podría explorar la aplicación de sDEM a otros problemas que impliquen el análisis de grandes cantidades de datos organizados de forma matricial.

Con respecto al problema de planificación de trayectorias, el futuro de la heurística VPP presentada aquí incluye la búsqueda de nuevas etapas de procesamiento que aumenten aún más el terreno cubierto—con menos puntos de paso en la trayectoria si es posible. Además, otro objetivo sería aplicar el estudio a más zonas forestales de diferente complejidad, explorando asimismo nuevas etapas basadas en el cálculo de la visibilidad.

Por último, muchas de las soluciones que se han propuesto pueden aplicarse a problemas espaciales completamente diferentes a los que se abordan en esta tesis, como se está haciendo actualmente. Como ejemplo de este hecho, se está trabajando en la adaptación del algoritmo sDEM (Capítulo B.5) para el procesamiento de datos requerido en uno de los problemas propuestos en el segundo *SKA Science Data Challenge* [179]: el análisis de un gran número de imágenes procedentes de una red mundial de radiotelescopios [180].

Bibliography

- [1] Christopher Lloyd. *Spatial data analysis: an introduction for GIS users*. Oxford university press, 2010. ISBN13: 9780199554324.
- [2] Stewart Fotheringham and Peter Rogerson. *Spatial analysis and GIS*. CRC Press, 2013. ISBN13: 9780748401048.
- [3] Kang-Tsung Chang. Geographic information system. *International Encyclopedia of Geography: People, the Earth, Environment and Technology*, pages 1–10, 2016. <https://doi.org/10.1002/9781118786352.wbieg0152.pub2>.
- [4] Khronos Group. OpenGL overview. Retrieved from <https://www.opengl.org/about/>, 2021. Accessed June 18, 2021.
- [5] NVIDIA Developer. CUDA zone. Retrieved from <https://developer.nvidia.com/cuda-zone>, 2021. Accessed June 23, 2021.
- [6] Hannes Fassold. Computer vision on the GPU – tools, algorithms and frameworks. In *2016 IEEE 20th Jubilee International Conference on Intelligent Engineering Systems (INES)*, pages 245–250. IEEE, 2016. <https://doi.org/10.1109/INES.2016.7555129>.
- [7] Khronos Group. OpenCL overview. Retrieved from <https://www.khronos.org/opencl/>, 2021. Accessed June 25, 2021.
- [8] Khronos Group. SYCL resources. Retrieved from <https://www.khronos.org/sycl/resources>, 2021. Accessed July 13, 2021.
- [9] Suejb Memeti, Lu Li, Sabri Pllana, Joanna Kołodziej, and Christoph Kessler. Benchmarking OpenCL, OpenACC, OpenMP, and CUDA: programming productivity, performance, and energy consumption. In *Proceedings of the 2017 Workshop on Adaptive Resource Management and Scheduling for Cloud Computing*, pages 1–6. Association for Computing Machinery, 2017. <https://doi.org/10.1145/3110355.3110356>.

- [10] NVIDIA Developer Zone. CUDA toolkit documentation. Retrieved from <https://docs.nvidia.com/cuda/cuda-compiler-driver-nvcc/index.html>, 2021. Accessed July 5, 2021.
- [11] John E Stone, Kirby L Vandivort, and Klaus Schulten. GPU-accelerated molecular visualization on petascale supercomputing platforms. In *Proceedings of the 8th International Workshop on Ultrascale Visualization*, pages 1–8. Association for Computing Machinery, 2013. <https://doi.org/10.1145/2535571.2535595>.
- [12] Michel Müller and Takayuki Aoki. New high performance GPGPU code transformation framework applied to large production weather prediction code. *ACM Transactions on Parallel Computing (TOPC)*, 5(2):1–42, 2018. <https://doi.org/10.1145/3291523>.
- [13] Faisal F Qarah and Yi-Cheng Tu. A fast exact viewshed algorithm on GPU. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 3397–3405. IEEE, 2019. <https://doi.org/10.1109/BigData47090.2019.9006353>.
- [14] Pablo David Gutierrez, Miguel Lastra, Francisco Herrera, and Jose Manuel Benitez. A high performance fingerprint matching system for large databases based on GPU. *IEEE Transactions on Information Forensics and Security*, 9(1):62–71, 2014. <https://doi.org/10.1109/TIFS.2013.2291220>.
- [15] Raffaele Cappelli, Matteo Ferrara, and Davide Maltoni. Large-scale fingerprint identification on GPU. *Information Sciences*, 306:1–20, 2015. <https://doi.org/10.1016/j.ins.2015.02.016>.
- [16] Miguel Lastra, Jesús Carabaño, Pablo D Gutiérrez, José M Benítez, and Francisco Herrera. Fast fingerprint identification using GPUs. *Information Sciences*, 301:195–214, 2015. <https://doi.org/10.1016/j.ins.2014.12.052>.
- [17] William J Dally, Yatish Turakhia, and Song Han. Domain-specific hardware accelerators. *Communications of the ACM*, 63(7):48–57, 2020. <https://doi.org/10.1145/3361682>.
- [18] Cong Hao, Atif Sarwari, Zhijie Jin, Husam Abu-Haimed, Daryl Sew, Yuhong Li, Xinheng Liu, Bryan Wu, Dongdong Fu, Junli Gu, et al. A hybrid GPU+FPGA system design for autonomous driving cars. In *2019 IEEE International Workshop on Signal Processing Systems (SiPS)*, pages 121–126. IEEE, 2019. <https://doi.org/10.1109/SiPS47522.2019.9020540>.

- [19] Firas Al-Ali, Thilina Doremure Gamage, Hewa WTS Nanayakkara, Farhad Mehdipour, and Sayan Kumar Ray. Novel case study and benchmarking of AlexNet for Edge AI: From CPU and GPU to FPGA. In *2020 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, pages 1–4. IEEE, 2020. <https://doi.org/10.1109/CCECE47787.2020.9255739>.
- [20] Peilong Li, Yan Luo, Ning Zhang, and Yu Cao. Heterospark: a heterogeneous CPU/GPU Spark platform for machine learning algorithms. In *2015 IEEE International Conference on Networking, Architecture and Storage (NAS)*, pages 347–348. IEEE, 2015. <https://doi.org/10.1109/NAS.2015.7255222>.
- [21] Xuan Peng, Xuanhua Shi, Hulin Dai, Hai Jin, Weiliang Ma, Qian Xiong, Fan Yang, and Xuehai Qian. Capuchin: Tensor-based GPU memory management for deep learning. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 891–905. Association for Computing Machinery, 2020. <https://doi.org/10.1145/3373376.3378505>.
- [22] Shubham Chaudhary, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, and Srinidhi Viswanatha. Balancing efficiency and fairness in heterogeneous gpu clusters for deep learning. In *Proceedings of the Fifteenth European Conference on Computer Systems*, pages 1–16. Association for Computing Machinery, 2020. <https://doi.org/10.1145/3342195.3387555>.
- [23] Jun Zhao, Xiaoliang Zhu, Wei Wang, and Ying Liu. Extended Kalman filter-based Elman networks for industrial time series prediction with GPU acceleration. *Neurocomputing*, 118:215–224, 2013. <https://doi.org/10.1016/j.neucom.2013.02.031>.
- [24] Jose C Romero, Antonio Vilches, Andrés Rodríguez, Angeles Navarro, and Rafael Asenjo. ScrimpCo: scalable matrix profile on commodity heterogeneous processors. *The Journal of Supercomputing*, pages 1–22, 2020. <https://doi.org/10.1007/s11227-020-03199-w>.
- [25] Mohammad A Alsmirat, Yaser Jararweh, Mahmoud Al-Ayyoub, Mohammed A Shehab, and Brij B Gupta. Accelerating compute intensive medical imaging segmentation algorithms using hybrid CPU-GPU implementations. *Multimedia Tools and Applications*, 76(3):3537–3555, 2017. <https://doi.org/10.1007/s11042-016-3884-2>.
- [26] Xi Chen, Chen Wang, Shanjiang Tang, Ce Yu, and Quan Zou. CMSA: a heterogeneous CPU/GPU computing system for multiple similar RNA/DNA

- sequence alignment. *BMC bioinformatics*, 18(1):315, 2017. <https://doi.org/10.1186/s12859-017-1725-6>.
- [27] Lin-Ching Chang, Esam El-Araby, Vinh Q Dang, and Lam H Dao. GPU acceleration of nonlinear diffusion tensor estimation using CUDA and MPI. *Neurocomputing*, 135:328–338, 2014. <https://doi.org/10.1016/j.neucom.2013.12.035>.
- [28] Mark Van Heeswijk, Yoan Miche, Erkki Oja, and Amaury Lendasse. GPU-accelerated and parallelized ELM ensembles for large-scale regression. *Neurocomputing*, 74(16):2430–2437, 2011. <https://doi.org/10.1016/j.neucom.2010.11.034>.
- [29] Shijie Li, Xin Niu, Yong Dou, Qi Lv, and Yueqing Wang. Heterogeneous blocked CPU-GPU accelerate scheme for large scale extreme learning machine. *Neurocomputing*, 261:153–163, 2017. <https://doi.org/10.1016/j.neucom.2016.05.112>.
- [30] Alexander V Smirnov. FIESTA4: optimized feynman integral calculations with GPU support. *Computer Physics Communications*, 204:189–199, 2016. <https://doi.org/10.1016/j.cpc.2016.03.013>.
- [31] Qingfeng Guan, Xuan Shi, Miaoqing Huang, and Chenggang Lai. A hybrid parallel cellular automata model for urban growth simulation over GPU/CPU heterogeneous architectures. *International Journal of Geographical Information Science*, 30(3):494–514, 2016. <https://doi.org/10.1080/13658816.2015.1039538>.
- [32] Danilo Valdes-Ramirez, Miguel Angel Medina-Pérez, Raúl Monroy, Octavio Loyola-González, Jorge Rodríguez, Aythami Morales, and Francisco Herrera. A review of fingerprint feature representations and their applications for latent fingerprint identification: Trends and evaluation. *IEEE Access*, 7:48484–48499, 2019. <https://doi.org/10.1109/ACCESS.2019.2909497>.
- [33] Miguel Angel Medina-Pérez, Aythami Morales Moreno, Miguel Angel Ferrer Ballester, Milton García-Borroto, Octavio Loyola-González, and Leopoldo Altamirano-Robles. Latent fingerprint identification using deformable minutiae clustering. *Neurocomputing*, 175:851–865, 2016. <https://doi.org/10.1016/j.neucom.2015.05.130>.
- [34] Vinayak Rai, Kapil Mehta, Jatin Jatin, Dheeraj Tiwari, and Rohit Chaurasia. Automated biometric personal identification-techniques and applications. In *2020 4th International Conference on Intelligent Computing*

- and Control Systems (ICICCS), pages 1023–1030. IEEE, 2020. <https://doi.org/10.1109/ICICCS48265.2020.9120896>.
- [35] Kelly A Gates. *Our biometric future: Facial recognition technology and the culture of surveillance*, volume 2. NYU Press, 2011. ISBN13: 9780814732106.
- [36] David D Zhang. *Automated biometrics: Technologies and systems*, volume 7. Springer Science & Business Media, 2013. <https://doi.org/10.1007/978-1-4615-4519-4>.
- [37] Göran Lundborg. Handprints from the past. In *The Hand and the Brain*, pages 41–48. Springer, 2014. https://doi.org/10.1007/978-1-4471-5334-4_5.
- [38] LM Mallory-Greenough, JD Greenough, G Dobosi, and JV Owen. Fingerprinting ancient Egyptian quarries: Preliminary results using laser ablation microprobe-inductively coupled plasma-mass spectrometry. *Archaeometry*, 41(2):227–238, 1999. <https://doi.org/10.1111/j.1475-4754.1999.tb00979.x>.
- [39] Jeffery G Barnes et al. Fingerprint sourcebook-chapter 1: History, 2011. ISBN13: 9781477664766.
- [40] Simon A Cole. History of fingerprint pattern recognition. In *Automatic Fingerprint Recognition Systems*, pages 1–25. Springer, 2004. https://doi.org/10.1007/0-387-21685-5_1.
- [41] William James Herschel. *The origin of finger-printing*. H. Milford, Oxford University Press, 1916. <https://doi.org/10.1038/098268a0>.
- [42] Laura A Hutchins. Fingerprint sourcebook-chapter 5: Systems of friction ridge classification. *NCJRS*, 2011. ISBN13: 9781477664766.
- [43] Francis Galton. *Finger prints*. Macmillan and Company, 1892. ISBN13: 9781512072273.
- [44] United States. Federal Bureau of Investigation. *The Identification Division of the FBI: A Brief Outline of the History, the Services, and the Operating Techniques of the World's Greatest Repository of Fingerprints*. Federal Bureau of Investigation, US Department of Justice, 1982.
- [45] FBI and United States of America. Proceedings of the international forensic symposium on latent prints. *NCJRS*, 1988. ISBN13: 9780932115089.

- [46] Kenneth R Moses, Peter Higgins, Michael McCabe, Salil Probhakar, and Scott Swann. Fingerprint sourcebook-chapter 6: automated fingerprint identification system (afis). *NCJRS*, 2011. ISBN13: 9781477664766.
- [47] Qinghai Gao and Daniel Pinto. Some challenges in forensic fingerprint classification and interpretation. In *2016 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*, pages 1–7. IEEE, 2016. <https://doi.org/10.1109/LISAT.2016.7494096>.
- [48] Davide Maltoni, Dario Maio, Anil K Jain, and Salil Prabhakar. *Handbook of fingerprint recognition*. Springer Science & Business Media, 2009. <https://doi.org/10.1007/978-1-84882-254-2>.
- [49] Michael Kücken and Alan C Newell. Fingerprint formation. *Journal of theoretical biology*, 235(1):71–83, 2005. <https://doi.org/10.1016/j.jtbi.2004.12.020>.
- [50] Xunqiang Tao, Xinjian Chen, Xin Yang, and Jie Tian. Fingerprint recognition with identical twin fingerprints. *PLOS ONE*, 7(4):e35704, 2012. <https://doi.org/10.1371/journal.pone.0035704>.
- [51] Edward Richard Henry. *Classification and uses of finger prints*. HM Stationery Office, printed by Darling and son, Limited, 1913. ISBN13: 9780260310781.
- [52] Daniel Peralta, Isaac Triguero, Salvador García, Yvan Saeys, Jose M Benítez, and Francisco Herrera. On the use of convolutional neural networks for robust classification of multiple fingerprint captures. *International Journal of Intelligent Systems*, 33(1):213–230, 2018. <https://doi.org/10.1002/int.21948>.
- [53] Anil K Jain and Jianjiang Feng. Latent fingerprint matching. *IEEE Transactions on pattern analysis and machine intelligence*, 33(1):88–100, 2011. <https://doi.org/10.1109/TPAMI.2010.59>.
- [54] Anush Sankaran, Mayank Vatsa, and Richa Singh. Latent fingerprint matching: A survey. *IEEE Access*, 2:982–1004, 2014. <https://doi.org/10.1109/ACCESS.2014.2349879>.
- [55] Megha Chhabra, Manoj K Shukla, and Kiran Kumar Ravulakollu. State-of-the-art: A systematic literature review of image segmentation in latent fingerprint forensics. *Recent Advances in Computer Science and Communications (Formerly: Recent Patents on Computer Science)*, 13(6):1115–1125, 2020. <https://doi.org/10.2174/2213275912666190429153952>.

- [56] Daniel Peralta, Isaac Triguero, Raul Sanchez-Reillo, Francisco Herrera, and Jose M. Benitez. Fast fingerprint identification for large databases. *Pattern Recognition*, 47(2):588–602, 2014. <https://doi.org/10.1016/j.patcog.2013.08.002>.
- [57] Brian DeCann and Arun Ross. Can a ‘poor’ verification system be a ‘good’ identification system? a preliminary study. In *2012 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 31–36. IEEE, 2012. <https://doi.org/10.1109/WIFS.2012.6412621>.
- [58] Weixin Bian, Deqin Xu, Qingde Li, Yongqiang Cheng, Biao Jie, and Xintao Ding. A survey of the methods on fingerprint orientation field estimation. *IEEE Access*, 7:32644–32663, 2019. <https://doi.org/10.1109/ACCESS.2019.2903601>.
- [59] Soweon Yoon, Jianjiang Feng, and Anil K Jain. Latent fingerprint enhancement via robust orientation field estimation. In *2011 international joint conference on biometrics (IJCB)*, pages 1–8. IEEE, 2011. <https://doi.org/10.1109/IJCB.2011.6117482>.
- [60] Jian Li, Jianjiang Feng, and C-C Jay Kuo. Deep convolutional neural network for latent fingerprint enhancement. *Signal Processing: Image Communication*, 60:52–63, 2018. <https://doi.org/10.1016/j.image.2017.08.010>.
- [61] Lazaro J. Gonzalez-Soler, Marta Gomez-Barrero, Leonardo Chang, Airl Perez-Suarez, and Christoph Busch. On the impact of different fabrication materials on fingerprint presentation attack detection. In *2019 International Conference on Biometrics, ICB 2019*, pages 1–8. IEEE, 2019. <https://doi.org/10.1109/ICB45273.2019.8987425>.
- [62] Haozhe Liu, Wentian Zhang, Guojie Liu, and Feng Liu. A zero-shot based fingerprint presentation attack detection system, 2020. <https://arxiv.org/abs/2002.04908>.
- [63] Ruben Tolosana, Marta Gomez-Barrero, Christoph Busch, and Javier Ortega-Garcia. Biometric presentation attack detection: Beyond the visible spectrum. *IEEE Transactions on Information Forensics and Security*, 15:1261–1275, 2019. <https://doi.org/10.1109/TIFS.2019.2934867>.
- [64] Khin Nandar Win, Kenli Li, Jianguo Chen, Philippe Fournier Viger, and Keqin Li. Fingerprint classification and identification algorithms for criminal investigation: A survey. *Future Generation Computer Systems*, 2019. <https://doi.org/10.1016/j.future.2019.10.019>.

- [65] Xudong Jiang and Wei-Yun Yau. Fingerprint minutiae matching based on the local and global structures. In *Proceedings 15th international conference on pattern recognition. ICPR-2000*, volume 2, pages 1038–1041. IEEE, 2000. <https://doi.org/10.1109/ICPR.2000.906252>.
- [66] Hong Hai Le, Ngoc Hoa Nguyen, and Tri-Thanh Nguyen. Speeding up and enhancing a large-scale fingerprint identification system on GPU. *Journal of Information and Telecommunication*, 2(2):147–162, 2018. <https://doi.org/10.1080/24751839.2017.1404712>.
- [67] Cristobal A Navarro, Nancy Hitschfeld-Kahler, and Luis Mateu. A survey on parallel computing and its applications in data-parallel problems using GPU architectures. *Communications in Computational Physics*, 15(2):285–329, 2014. <https://doi.org/10.4208/cicp.110113.010813a>.
- [68] Kalle Karu and Anil K Jain. Fingerprint classification. *Pattern recognition*, 29(3):389–404, 1996. [https://doi.org/10.1016/0031-3203\(95\)00106-9](https://doi.org/10.1016/0031-3203(95)00106-9).
- [69] Arun Ross, Anil Jain, and James Reisman. A hybrid fingerprint matcher. *Pattern Recognition*, 36(7):1661–1673, 2003. [https://doi.org/10.1016/S0031-3203\(02\)00349-7](https://doi.org/10.1016/S0031-3203(02)00349-7).
- [70] Raffaele Cappelli, Matteo Ferrara, and Davide Maltoni. Minutia cylinder-code: A new representation and matching technique for fingerprint recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(12):2128–2141, 2010. <https://doi.org/10.1109/TPAMI.2010.52>.
- [71] Daniel Peralta, Mikel Galar, Isaac Triguero, Daniel Paternain, Salvador García, Edurne Barrenechea, José M Benítez, Humberto Bustince, and Francisco Herrera. A survey on fingerprint minutiae-based local matching for verification and identification: Taxonomy and experimental evaluation. *Information Sciences*, 315:67–87, 2015. <https://doi.org/10.1016/j.ins.2015.04.013>.
- [72] Anna Mikaelyan and Josef Bigun. Ground truth and evaluation for latent fingerprint matching. In *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 83–88. IEEE, 2012. <https://doi.org/10.1109/CVPRW.2012.6239220>.
- [73] Anil K Jain, Jianjiang Feng, Abhishek Nagar, and Karthik Nandakumar. On matching latent fingerprints. In *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–8. IEEE, 2008. <https://doi.org/10.1109/CVPRW.2008.4563117>.

- [74] Alessandra A Paulino, Jianjiang Feng, and Anil K Jain. Latent fingerprint matching using descriptor-based Hough transform. *IEEE Transactions on Information Forensics and Security*, 8(1):31–45, 2013. <https://doi.org/10.1109/TIFS.2012.2223678>.
- [75] Sunpreet S Arora, Eryun Liu, Kai Cao, and Anil K Jain. Latent fingerprint matching: performance gain via feedback from exemplar prints. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(12):2452–2465, 2014. <https://doi.org/10.1109/TPAMI.2014.2330609>.
- [76] Asker M Bazen and Sabih H Gerez. Fingerprint matching by thin-plate spline modelling of elastic deformations. *Pattern Recognition*, 36(8):1859–1867, 2003. [https://doi.org/10.1016/S0031-3203\(03\)00036-0](https://doi.org/10.1016/S0031-3203(03)00036-0).
- [77] Kai Cao, Eryun Liu, Liaojun Pang, Jimin Liang, and Jie Tian. Fingerprint matching by incorporating minutiae discriminability. In *2011 International Joint Conference on Biometrics (IJCB)*, pages 1–6. IEEE, 2011. <https://doi.org/10.1109/IJCB.2011.6117537>.
- [78] Miguel Angel Medina-Pérez, Milton García-Borroto, Andres Eduardo Gutierrez-Rodríguez, and Leopoldo Altamirano-Robles. Improving fingerprint verification using minutiae triplets. *Sensors*, 12(3):3418–3437, 2012. <https://doi.org/10.3390/s120303418>.
- [79] Siham Tabik, Maurice Peemen, Nicolas Guil, and Henk Corporaal. Demystifying the 16×16 thread-block for stencils on the GPU. *Concurrency and Computation: Practice and Experience*, 27(18):5557–5573, 2015. <https://doi.org/10.1002/cpe.3591>.
- [80] FVC-onGoing. On-line evaluation of fingerprint recognition algorithms. Retrieved from <https://biolab.csr.unibo.it/FVCOnGoing>, 2020. Accessed April 1, 2021.
- [81] Bernadette Dorizzi, Raffaele Cappelli, Matteo Ferrara, Dario Maio, Davide Maltoni, Nesma Houmani, Sonia Garcia-Salicetti, and Aurélien Mayoue. Fingerprint and on-line signature verification competitions at ICB 2009. In *International Conference on Biometrics*, pages 725–732. Springer, 2009. https://doi.org/10.1007/978-3-642-01793-3_74.
- [82] Andres J. Sanchez-Fernandez, Luis F. Romero, Siham Tabik, Miguel Angel Medina-Pérez, and Francisco Herrera. A first step to accelerating fingerprint matching based on deformable minutiae clustering. In *Conference of the Spanish Association for Artificial Intelligence*, pages 361–371. Springer, 2018. https://doi.org/10.1007/978-3-030-00374-6_34.

- [83] Conrad Sanderson and Ryan Curtin. Armadillo: a template-based C++ library for linear algebra. *Journal of Open Source Software*, 1(2):26, 2016. <https://doi.org/10.21105/joss.00026>.
- [84] Conrad Sanderson and Ryan Curtin. A user-friendly hybrid sparse matrix class in C++. In *International Congress on Mathematical Software*, pages 422–430. Springer, 2018. https://doi.org/10.1007/978-3-319-96418-8_50.
- [85] Michael D Garris and Michael D Garris. NIST special database 27: fingerprint minutiae from latent and matching tenprint images. *National Institute of Standards and Technology*, 2000. <https://doi.org/10.6028/NIST.IR.6534>.
- [86] Craig I Watson and CL Wilson. NIST special database 4. *National Institute of Standards and Technology*, 1992.
- [87] Craig I Watson. NIST special database 14: mated fingerprint cards pairs 2 version 2. *National Institute of Standards and Technology*, 2001.
- [88] Raffaele Cappelli, Dario Maio, Davide Maltoni, James L Wayman, and Anil K Jain. Performance evaluation of fingerprint verification systems. *IEEE transactions on pattern analysis and machine intelligence*, 28(1):3–18, 2006. <https://doi.org/10.1109/TPAMI.2006.20>.
- [89] Neurotechnology. Verifinger SDK. Retrieved from <http://www.neurotechnology.com/verifinger.html>, 2020. Accessed April 20, 2021.
- [90] Dario Maio, Davide Maltoni, Raffaele Cappelli, James L Wayman, and Anil K Jain. FVC2002: second fingerprint verification competition. In *Object recognition supported by user interaction for service robots*, volume 3, pages 811–814. IEEE, 2002. <https://doi.org/10.1109/ICPR.2002.1048144>.
- [91] Dario Maio, Davide Maltoni, Raffaele Cappelli, Jim L Wayman, and Anil K Jain. FVC2004: third fingerprint verification competition. In *International Conference on Biometric Authentication*, pages 1–7. Springer, 2004. https://doi.org/10.1007/978-3-540-25948-0_1.
- [92] Raffaele Cappelli, Matteo Ferrara, Annalisa Franco, and Davide Maltoni. Fingerprint verification competition 2006. *Biometric Technology Today*, 15(7-8):7–9, 2007. [https://doi.org/10.1016/S0969-4765\(07\)70140-6](https://doi.org/10.1016/S0969-4765(07)70140-6).
- [93] Nalini Ratha and Ruud Bolle. *Automatic fingerprint recognition systems*. Springer Science & Business Media, 2003. <https://doi.org/10.1007/b97425>.

- [94] Eric E Jones. Using viewshed analysis to explore settlement choice: A case study of the Onondaga Iroquois. *American Antiquity*, pages 523–538, 2006. <https://doi.org/10.2307/40035363>.
- [95] Marín Pompa-García, Raúl Solís-Moreno, Efraín Rodríguez-Téllez, Alfredo Pinedo-Álvarez, Diana Avila-Flores, Ciro Hernández-Díaz, and Efraín Velasco-Bautista. Viewshed analysis for improving the effectiveness of watchtowers, in the north of Mexico. *The Open Forest Science Journal*, 3(1), 2010. <https://doi.org/10.2174/1874398601003010017>.
- [96] MathWorks. The three main families of map projections. Retrieved from <https://www.mathworks.com/help/map/the-three-main-families-of-map-projections.html>, 2020. Accessed April 12, 2021.
- [97] D-ICE Engineering. Coordinate systems. Retrieved from http://theory.frydom.org/source/coordinate_systems.html, 2019. Accessed April 18, 2021.
- [98] Vortex FdC. An introduction to geographic coordinates. Retrieved from <https://vortexfdc.com/an-introduction-to-geographic-coordinates/>, 2019. Accessed April 27, 2021.
- [99] Jafar Aghajani, Parissa Farnia, Ali Akbar Velayati, et al. Impact of geographical information system on public health sciences. *Biomedical and Biotechnology Research Journal (BBRJ)*, 1(2):94, 2017. https://doi.org/10.4103/bbrj.bbrj_34_17.
- [100] Henry James. *Photo-zincography*. Forbes & Bennett, 1860. ISBN13: 97811113324283.
- [101] Aran J Cauchi-Saunders and Ian J Lewis. GPU enabled XDraw viewshed analysis. *Journal of Parallel and Distributed Computing*, 84:87–93, 2015. <https://doi.org/10.1016/j.jpdc.2015.07.001>.
- [102] Yiwen Wang and Wanfeng Dou. A fast candidate viewpoints filtering algorithm for multiple viewshed site planning. *International Journal of Geographical Information Science*, pages 1–16, 2019. <https://doi.org/10.1080/13658816.2019.1664743>.
- [103] Tomislav Hengl and Ian S Evans. Mathematical and digital models of the land surface. *Developments in soil science*, 33:31–63, 2009. [https://doi.org/10.1016/S0166-2481\(08\)00002-0](https://doi.org/10.1016/S0166-2481(08)00002-0).

- [104] Leonid Djinevski, Sonja Stojanova, and Dimitar Trajanov. Optimizing durkins propagation model based on TIN terrain structures. In *International Conference on ICT Innovations*, pages 263–272. Springer, 2013. https://doi.org/10.1007/978-3-319-01466-1_25.
- [105] A Nelson, HI Reuter, and P Gessler. DEM production methods and sources. *Developments in soil science*, 33:65–85, 2009. [https://doi.org/10.1016/S0166-2481\(08\)00003-2](https://doi.org/10.1016/S0166-2481(08)00003-2).
- [106] Environmental Systems Research Institute (ESRI). ArcGIS Desktop, Version 10.7. Retrieved from <https://desktop.arcgis.com/en/arcmap/10.7/get-started/setup/arcgis-desktop-quick-start-guide.htm>, 2021.
- [107] QGIS Development Team. QGIS Geographic Information System, Version 3.10. Retrieved from <https://www.qgis.org/en/docs/index.html#310>, 2021.
- [108] Wanfeng Dou, Yanan Li, and Yanli Wang. An equal-area triangulated partition method for parallel XDraw viewshed analysis. *Concurrency and Computation: Practice and Experience*, 31(17):e5216, 2019. <https://doi.org/10.1002/cpe.5216>.
- [109] Howie M Choset, Seth Hutchinson, Kevin M Lynch, George Kantor, Wolfram Burgard, Lydia E Kavradi, and Sebastian Thrun. *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005. ISBN13: 9780262033275.
- [110] Siham Tabik, Antonio R Cervilla, Emilio Zapata, and Luis F Romero. Efficient data structure and highly scalable algorithm for total-viewshed computation. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 8(1):304–310, 2014. <https://doi.org/10.1109/JSTARS.2014.2326252>.
- [111] A James Stewart. Fast horizon computation at all points of a terrain with visibility and shading applications. *Visualization and Computer Graphics, IEEE Transactions on*, 4(1):82–93, 1998. <https://doi.org/10.1109/2945.675656>.
- [112] Antonio Rodriguez Cervilla, Siham Tabik, and Luis F Romero. Siting multiple observers for maximum coverage: an accurate approach. *Procedia Computer Science*, 51:356–365, 2015. <https://doi.org/10.1016/j.procs.2015.05.255>.

- [113] Jie Li, Changwen Zheng, and Xiaohui Hu. An effective method for complete visual coverage path planning. In *2010 Third International Joint Conference on Computational Science and Optimization*, volume 1, pages 497–500. IEEE, 2010. <https://doi.org/10.1109/CSO.2010.167>.
- [114] Wm Randolph Franklin and Clark Ray. Higher isn't necessarily better: Visibility algorithms and experiments. In *Advances in GIS research: sixth international symposium on spatial data handling*, volume 2, pages 751–770. Taylor & Francis Edinburgh, 1994.
- [115] Mikhail J Atallah. Dynamic computational geometry. In *Foundations of Computer Science, 1983., 24th Annual Symposium on*, pages 92–99. IEEE, 1983. <https://doi.org/10.1109/SFCS.1983.13>.
- [116] Brian Cabral, Nelson Max, and Rebecca Springmeyer. Bidirectional reflection functions from surface bump maps. In *ACM SIGGRAPH Computer Graphics*, pages 273–281. Association for Computing Machinery, 1987. <https://doi.org/10.1145/37402.37434>.
- [117] Peter F Fisher. First experiments in viewshed uncertainty: simulating fuzzy viewsheds. *Photogrammetric engineering and remote sensing*, 58:345–345, 1992.
- [118] Leila De Floriani and Paola Magillo. Visibility algorithms on triangulated digital terrain models. *International Journal of Geographical Information Systems*, 8(1):13–41, 1994. <https://doi.org/10.1080/02693799408901985>.
- [119] W. Randolph Franklin, Jr Clark K Ray, and Shashank Mehta. Geometric algorithms for siting of air defense missile batteries. Technical Report DAAL03-86-D-0001, Delivery Order Number 2756, US Army Topographic Engineering Center - Scientific Services Program, Battelle, Columbus Division, 1994.
- [120] Branko Kaučič and Borut Zalik. Comparison of viewshed algorithms on regular spaced points. In *Proceedings of the 18th spring conference on Computer graphics*, pages 177–183. Association for Computing Machinery, 2002. <https://doi.org/10.1145/584458.584487>.
- [121] Yong Gao, Hao Yu, Yu Liu, Yuehu Liu, Mingchao Liu, and Yong Zhao. Optimization for viewshed analysis on GPU. In *2011 19th International Conference on Geoinformatics*, pages 1–5. IEEE, 2011. <https://doi.org/10.1109/GeoInformatics.2011.5980830>.

- [122] Natalija Stojanović and Dragan Stojanović. Performance improvement of viewshed analysis using GPU. In *2013 11th International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting Services (TELSIKS)*, volume 2, pages 397–400. IEEE, 2013. <https://doi.org/10.1109/TELSKS.2013.6704407>.
- [123] Andrej Osterman, Lucas Benedičič, and Patrik Ritoša. An IO-efficient parallel implementation of an R2 viewshed algorithm for large terrain maps on a CUDA GPU. *International Journal of Geographical Information Science*, 28(11):2304–2327, 2014. <https://doi.org/10.1080/13658816.2014.918319>.
- [124] Yanli Zhao, Anand Padmanabhan, and Shaowen Wang. A parallel computing approach to viewshed analysis of large terrain data using graphics processing units. *International Journal of Geographical Information Science*, 27(2):363–384, 2013. <https://doi.org/10.1080/13658816.2012.692372>.
- [125] Damjan Strnad. Parallel terrain visibility calculation on the graphics processing unit. *Concurrency and Computation: Practice and Experience*, 23(18):2452–2462, 2011. <https://doi.org/10.1002/cpe.1808>.
- [126] Xiao-Dong Song, Guo-An Tang, Xue-Jun Liu, Wan-Feng Dou, and Fa-Yuan Li. Parallel viewshed analysis on a PC cluster system using triple-based irregular partition scheme. *Earth Science Informatics*, 9(4):511–523, 2016. <https://doi.org/10.1007/s12145-016-0263-5>.
- [127] Wanfeng Dou, Yanan Li, and Yanli Wang. A fine-granularity scheduling algorithm for parallel XDraw viewshed analysis. *Earth Science Informatics*, 11(3):433–447, 2018. <https://doi.org/10.1007/s12145-018-0339-5>.
- [128] Katherine A Dungan, Devin White, Sylviane Déderix, Barbara J Mills, and Kristin Safi. A total viewshed approach to local visibility in the Chaco World. *Antiquity*, 92(364):905–921, 2018. <https://doi.org/10.15184/aqy.2018.135>.
- [129] Tom Brughmans, Mereke van Garderen, and Mark Gillings. Introducing visual neighbourhood configurations for total viewsheds. *Journal of Archaeological Science*, 96:14–25, 2018. <https://doi.org/10.1016/j.jas.2018.05.006>.
- [130] Siham Tabik, Emilio L Zapata, and Luis F Romero. Simultaneous computation of total viewshed on large high resolution grids. *International Journal of Geographical Information Science*, 27(4):804–814, 2013. <https://doi.org/10.1080/13658816.2012.677538>.

- [131] Jack Bresenham. A linear algorithm for incremental digital display of circular arcs. *Communications of the ACM*, 20(2):100–106, 1977. <https://doi.org/10.1145/359423.359432>.
- [132] Consejería de Agricultura, Ganadería, Pesca y Desarrollo Sostenible de la Junta de Andalucía. Modelos digitales del terreno. Retrieved from http://www.juntadeandalucia.es/medioambiente/mapwms/REDIAM_WCS_mdt?, 2020. Accessed October 22, 2021.
- [133] NVIDIA. NVIDIA Nsight Visual Studio Edition: Achieved occupancy. Retrieved from <https://docs.nvidia.com/gameworks/content/developertools/desktop/analysis/report/cudaexperiments/kernellevel/achievedoccupancy.htm>, 2015. Accessed December 27, 2021.
- [134] European Commission. Report: new technology for drone-based emergency response missions. Technical Report 415444, European Commission - Results in Brief, Brussels, 2020.
- [135] Guijun Yang, Jiangang Liu, Chunjiang Zhao, Zhenhong Li, Yanbo Huang, Haiyang Yu, Bo Xu, Xiaodong Yang, Dongmei Zhu, Xiaoyan Zhang, et al. Unmanned aerial vehicle remote sensing for field-based crop phenotyping: current status and perspectives. *Frontiers in plant science*, 8:1111, 2017. <https://doi.org/10.3389/fpls.2017.01111>.
- [136] Petros Petrides, Panayiotis Kolios, Christos Kyrkou, Theocharis Theocharides, and Christos Panayiotou. Disaster prevention and emergency response using unmanned aerial systems. In *Smart Cities in the Mediterranean*, pages 379–403. Springer, 2017. https://doi.org/10.1007/978-3-319-54558-5_18.
- [137] Milan Erdelj, Enrico Natalizio, Kaushik R Chowdhury, and Ian F Akyildiz. Help from the sky: Leveraging UAVs for disaster management. *IEEE Pervasive Computing*, 16(1):24–32, 2017. <https://doi.org/10.1109/MPRV.2017.11>.
- [138] Abdulla Al-Kaff, Francisco Miguel Moreno, Luis Javier San José, Fernando García, David Martín, Arturo de la Escalera, Alberto Nieva, and José Luis Meana Garcéa. VBII-UAV: vision-based infrastructure inspection-UAV. In *World Conference on Information Systems and Technologies*, pages 221–231. Springer, 2017. https://doi.org/10.1007/978-3-319-56538-5_24.

- [139] Oussama Bekkouché, Tarik Taleb, and Miloud Bagaa. UAVs traffic control based on multi-access edge computing. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2018. <https://doi.org/10.1109/GLOCOM.2018.8647421>.
- [140] Scott D Hamshaw, Tayler Engel, Donna M Rizzo, Jarlath O’Neil-Dunne, and Mandar M Dewoolkar. Application of unmanned aircraft system (UAS) for monitoring bank erosion along river corridors. *Geomatics, Natural Hazards and Risk*, 10(1):1285–1305, 2019. <https://doi.org/10.1080/19475705.2019.1571533>.
- [141] Wenli Li, W Randolph Franklin, Salles Viana Gomes de Magalhães, and Marcus VA Andrade. GPU-accelerated multiple observer siting. *Photogrammetric Engineering & Remote Sensing*, 83(6):439–446, 2017. <https://doi.org/10.14358/PERS.83.6.439>.
- [142] Genya Ishigami, Keiji Nagatani, and Kazuya Yoshida. Path planning for planetary exploration rovers and its evaluation based on wheel slip dynamics. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 2361–2366. IEEE, 2007. <https://doi.org/10.1109/ROBOT.2007.363672>.
- [143] Han Wang, Bingjing Yan, Xiaoxia Li, Xuejing Luo, Qiang Yang, and Wenjun Yan. On optimal path planning for UAV based patrolling in complex 3D topographies. In *2016 IEEE international conference on information and automation (ICIA)*, pages 986–990. IEEE, 2016. <https://doi.org/10.1109/ICInfA.2016.7831962>.
- [144] Alessandro Gasparetto, Paolo Boscariol, Albano Lanzutti, and Renato Vidoni. Path planning and trajectory planning algorithms: A general overview. *Motion and Operation Planning of Robotic Systems: Background and Practical Approaches*, pages 3–27, 2015. https://doi.org/10.1007/978-3-319-14705-5_1.
- [145] Gregor Klancar, Andrej Zdesar, Saso Blazic, and Igor Skrjanc. *Wheeled mobile robotics: from fundamentals towards autonomous systems*. Butterworth-Heinemann, 2017. ISBN13: 9780128042045.
- [146] Jiajun Gu and Qixin Cao. Path planning for mobile robot in a 2.5-dimensional grid-based map. *Industrial Robot: An International Journal*, 2011. <https://doi.org/10.1108/01439911111122815>.

- [147] Andrey V Savkin, Alexey S Matveev, Michael Hoy, and Chao Wang. *Safe robot navigation among moving and steady obstacles*. Butterworth-Heinemann, 2016. <https://doi.org/10.1016/C2014-0-04846-0>.
- [148] Masoud Fetanat, Sajjad Haghzad, and Saeed Bagheri Shouraki. Optimization of dynamic mobile robot path planning based on evolutionary methods. In *2015 AI & Robotics (IRANOPEN)*, pages 1–7. IEEE, 2015. <https://doi.org/10.1109/RIOS.2015.7270743>.
- [149] Bernhard Korte and Jens Vygen. The traveling salesman problem. In *Combinatorial Optimization: Theory and Algorithms*, pages 527–562. Springer, 2008. https://doi.org/10.1007/978-3-540-71844-4_21.
- [150] A Hanif Halim and IJAoCMiE Ismail. Combinatorial optimization: comparison of heuristic algorithms in travelling salesman problem. *Archives of Computational Methods in Engineering*, 26(2):367–380, 2019. <https://doi.org/10.1007/s11831-017-9247-y>.
- [151] Roneeta Purkayastha, Tanmay Chakraborty, Anirban Saha, and Debarka Mukhopadhyay. Study and analysis of various heuristic algorithms for solving travelling salesman problem – a survey. In *Proceedings of the Global AI Congress 2019*, pages 61–70. Springer, 2020. https://doi.org/10.1007/978-981-15-2188-1_5.
- [152] Yangguang Fu, Mingyue Ding, Chengping Zhou, and Hanping Hu. Route planning for unmanned aerial vehicle (UAV) on the sea using hybrid differential evolution and quantum-behaved particle swarm optimization. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 43(6):1451–1465, 2013. <https://doi.org/10.1109/TSMC.2013.2248146>.
- [153] Yucong Lin and Srikanth Saripalli. Sampling-based path planning for UAV collision avoidance. *IEEE Transactions on Intelligent Transportation Systems*, 18(11):3179–3192, 2017. <https://doi.org/10.1109/TITS.2017.2673778>.
- [154] Chenxi Huang, Yisha Lan, Yuchen Liu, Wen Zhou, Hongbin Pei, Longzhi Yang, Yongqiang Cheng, Yongtao Hao, and Yonghong Peng. A new dynamic path planning approach for unmanned aerial vehicles. *Complexity*, 2018, 2018. <https://doi.org/10.1155/2018/8420294>.
- [155] Ugur Cekmez, Mustafa Ozsiginan, and Ozgur Koray Sahingoz. Multi colony ant optimization for UAV path planning with obstacle avoidance. In *2016 international conference on unmanned aircraft systems (ICUAS)*, pages 47–52. IEEE, 2016. <https://doi.org/10.1109/ICUAS.2016.7502621>.

- [156] Xia Chen, Miaoyan Zhao, and Liyuan Yin. Dynamic path planning of the UAV avoiding static and moving obstacles. *Journal of Intelligent & Robotic Systems*, 99(3):909–931, 2020. <https://doi.org/10.1007/s10846-020-01151-x>.
- [157] Jie Chen, Fang Ye, and Tao Jiang. Path planning under obstacle-avoidance constraints based on ant colony optimization algorithm. In *2017 IEEE 17th International Conference on Communication Technology (ICCT)*, pages 1434–1438. IEEE, 2017. <https://doi.org/10.1109/ICCT.2017.8359869>.
- [158] Carmelo Di Franco and Giorgio Buttazzo. Energy-aware coverage path planning of UAVs. In *2015 IEEE international conference on autonomous robot systems and competitions*, pages 111–117. IEEE, 2015. <https://doi.org/10.1109/ICARSC.2015.17>.
- [159] Shaimaa Ahmed, Amr Mohamed, Khaled Harras, Mohamed Kholief, and Saleh Mesbah. Energy efficient path planning techniques for UAV-based systems with space discretization. In *2016 IEEE Wireless Communications and Networking Conference*, pages 1–6. IEEE, 2016. <https://doi.org/10.1109/WCNC.2016.7565126>.
- [160] Joo-Seok Lee and Kee-Ho Yu. Optimal path planning of solar-powered UAV using gravitational potential energy. *IEEE Transactions on Aerospace and Electronic Systems*, 53(3):1442–1451, 2017. <https://doi.org/10.1109/TAES.2017.2671522>.
- [161] Momena Monwar, Omid Semiari, and Walid Saad. Optimized path planning for inspection by unmanned aerial vehicles swarm with energy constraints. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2018. <https://doi.org/10.1109/GLOCOM.2018.8647342>.
- [162] German Gramajo and Praveen Shankar. An efficient energy constraint based UAV path planning for search and coverage. *International Journal of Aerospace Engineering*, 2017, 2017. <https://doi.org/10.1155/2017/8085623>.
- [163] Taua M Cabreira, Carmelo Di Franco, Paulo R Ferreira, and Giorgio C Buttazzo. Energy-aware spiral coverage path planning for UAV photogrammetric applications. *IEEE Robotics and Automation Letters*, 3(4):3662–3668, 2018. <https://doi.org/10.1109/LRA.2018.2854967>.
- [164] LH Nam, Loulin Huang, Xue Jun Li, and JF Xu. An approach for coverage path planning for UAVs. In *2016 IEEE 14th international workshop on*

- advanced motion control (AMC)*, pages 411–416. IEEE, 2016. <https://doi.org/10.1109/AMC.2016.7496385>.
- [165] Gerard Leng, Zhang Qian, and Vengatesan Govindaraju. Multi-UAV surveillance over forested regions. *Photogrammetric Engineering & Remote Sensing*, 80(12):1129–1137, 2014. <https://doi.org/10.14358/PERS.80.12.1129>.
- [166] Junzhong Zhou, Wei Zhang, Yizhe Zhang, Yuqiang Zhao, and Yiyuan Ma. Optimal path planning for UAV patrolling in forest fire prevention. In *Asia-Pacific International Symposium on Aerospace Technology*, pages 2209–2218. Springer, 2018. https://doi.org/10.1007/978-981-13-3305-7_178.
- [167] European Commission. Report: more countries than ever hit by forest fires. Technical Report IP/19/6176, European Commission - Press release, Brussels, 2019.
- [168] Inma Aljaro. Un incendio en los Montes de Málaga arrasa siete hectáreas. *La Opinión de Málaga*. Retrieved from <https://www.laopiniondemalaga.es/sucesos/2607/incendio-montes-malaga-arrasa-siete-hectareas/79671.html>, 2006. Accessed November 18, 2021.
- [169] Laura Álvarez. Estabilizado el incendio de los Montes de Málaga. *El Mundo*. Retrieved from <https://www.elmundo.es/andalucia/2014/08/10/53e78be1e2704e4c278b457c.html>, 2014. Accessed November 5, 2021.
- [170] Francisco Jiménez. Bajo control un incendio forestal entre olías y los Montes de Málaga. *Diario Sur*. Retrieved from <https://www.diariosur.es/malaga-capital/declarado-incendio-forestal-20190707141322-nt.html>, 2019. Accessed November 11, 2021.
- [171] ArcGIS Desktop. What is the ArcGIS Network Analyst extension? Retrieved from <https://desktop.arcgis.com/en/arcmap/latest/extensions/network-analyst/what-is-network-analyst-.htm>, 2020. Accessed November 7, 2021.
- [172] DJI. Phantom 4 Pro. Retrieved from <https://www.dji.com/es/phantom-4-pro/info#specs>, 2021. Accessed December 20, 2021.
- [173] Andres J. Sanchez-Fernandez, Luis F. Romero, Gerardo Bandera, and Siham Tabik. A data relocation approach for terrain surface analysis on multi-GPU systems: a case study on the total viewshed problem. *International Journal of Geographical Information Science*, pages 1–21, 2020. <https://doi.org/10.1080/13658816.2020.1844207>.

- [174] Andres J. Sanchez-Fernandez, Luis F. Romero, Gerardo Bandera, and Siham Tabik. Internal report from the Department of Computer Architecture (University of Malaga): optimal surveillance flight for UAVs in fire protection for the Montes de Malaga Natural Park. Retrieved from <https://vimeo.com/389926980>, 2020. Accessed November 3, 2021.
- [175] Andres J. Sanchez-Fernandez, Luis F. Romero, Gerardo Bandera, and Siham Tabik. Internal report from the Department of Computer Architecture (University of Malaga): UAV flight test. Retrieved from <https://youtu.be/cMQgl5phTws>, 2021. Accessed December 23, 2021.
- [176] Andres J. Sanchez-Fernandez, Luis F. Romero, Daniel Peralta, Miguel Angel Medina-Pérez, Yvan Saeys, Francisco Herrera, and Siham Tabik. Asynchronous processing for latent fingerprint identification on heterogeneous CPU-GPU systems. *IEEE Access*, 8:124236–124253, 2020. <https://doi.org/10.1109/ACCESS.2020.3005476>.
- [177] AERTEC Solutions. Projects/solutions. Retrieved from <https://aertecsolutions.com/en/proyectos/>, 2021. Accessed December 18, 2021.
- [178] Andres J. Sanchez-Fernandez, Luis F. Romero, Gerardo Bandera, and Siham Tabik. VPP: visibility-based path planning heuristic for monitoring large regions of complex terrain using a UAV onboard camera. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 2021. <https://doi.org/10.1109/JSTARS.2021.3134948>.
- [179] Square Kilometre Array. SKA science data challenge 2. Retrieved from <https://sdc2.astronomers.skatelescope.org/>, 2021. Accessed December 19, 2021.
- [180] Square Kilometre Array. SDC2 dataset. Retrieved from <https://sdc2.astronomers.skatelescope.org/sdc2-challenge/data>, 2021. Accessed December 19, 2021.