



Universitat d'Alacant  
Universidad de Alicante



A Three-Dimensional Representation method  
for Noisy Point Clouds based on Growing  
Self-Organizing Maps accelerated on GPUs

Xavier del Pozo Somoza

Tesis Doctorales

[www.eltallerdigital.com](http://www.eltallerdigital.com)

UNIVERSIDAD de ALICANTE



University of Alicante

Phd Thesis

A Three-Dimensional  
Representation method for  
Noisy Point Clouds based on  
Growing Self-Organizing Maps  
accelerated on GPUs

Sergio Orts Escolano

Supervisors

Dr. José García Rodríguez  
Dr. Miguel Ángel Cazorla Quevedo





UNIVERSIDAD DE ALICANTE

Phd Thesis

A Three-Dimensional  
Representation method for  
Noisy Point Clouds based on  
Growing Self-Organizing Maps  
accelerated on GPUs

Universitat d'Alacant  
Sergio Orts Escolano  
Universidad de Alicante

Supervisors

Dr. José García Rodríguez  
Dr. Miguel Ángel Cazorla Quevedo

Departamento de Tecnología Informática y Computación  
TECNOLOGÍAS PARA LA SOCIEDAD DE LA  
INFORMACIÓN

Alicante, December 3, 2013



*Learn from yesterday, live for today, hope for tomorrow.  
The important thing is not to stop questioning.*

**Albert Einstein**



Universitat d'Alacant  
Universidad de Alicante



# Agradecimientos

---

En primer lugar, me gustaría agradecer enormemente a mis directores de tesis José y Miguel, por su inagotable dedicación durante el desarrollo de este trabajo, sin ellos nada de esto hubiera sido posible. Gracias por compartir conmigo todos vuestros conocimientos y experiencias, por haberme enseñado a mejorar cada día en mi trabajo. También agradecerlos vuestra relación más allá de lo profesional y por haberme tratado como un amigo.

Quiero agradecer a Patricia por su paciencia y comprensión, gracias por estar a mi lado durante este camino, se puede decir que esta tesis lleva un trozo de ti.

Gracias a mi familia por creer siempre en mí, por apoyarme en todo lo que he hecho sin cuestionar mis decisiones. Gracias especialmente a vosotros, Papá y Mamá, sin vosotros no sería la persona que soy a día de hoy.

Agradezco sinceramente a mis compañeros y amigos del Departamento de Tecnología Informática y Computación de la Universidad de Alicante, del grupo *I<sup>2</sup>RC* y de la Escuela Politécnica Superior de Alicante, quienes han aportado significativamente a esta tesis con sus ánimos, consejos, críticas y ante todo amistad. Gracias Vicente, Marcelo, Alexandros, Felipe, Paco, Rafa, Óscar, José, Jorge, David, Diego, Toni, José Luis, Virgilio, Diego Marcos, Mora, Higinio, Paco, Sergio, Antonio y por supuesto a todo el equipo de personal técnico, directivo y administrativo.

Agradecer también a los grupos de investigación UNICAD y RoViT por crear un nexo de colaboración y permitir que distintas personas puedan aunar sus conocimientos para avanzar en el campo de la investigación.

Muchas gracias a los amigos de siempre, por poder contar siempre con

vosotros cuando os he necesitado, también habéis formado parte de estos últimos años de trabajo que finalmente concluyen con esta tesis.

I would also like to thank Professor Robert B. Fisher from the University of Edinburgh for his advises during my collaboration stay at the Institute of Perception Action and Behaviour (IPAB). Thanks to all the good people I met there: Bas, Steven, Cigdem, Phoenix, Vladimir, Pete, etc.

Alicante, 3 de Diciembre de 2013

Sergio Orts Escolano



Universitat d'Alacant  
Universidad de Alicante

# Abstract

---

Artificial Vision or Computer Vision is a computer science field that since the 80s has been constantly evolving. This field is focused on the ability to "understand" an observation of a scene using a computer. Recognition of objects in a scene, where it was captured, the reconstruction of a 3D model from multiple shots, etc. would be some examples of computer vision objectives. These scenes have been traditionally captured using 2D image sensors, allowing image processing and analysis on 2D color images. 3D sensors have also been historically used but as these were very expensive compared to image sensors these have been less used. Therefore, most works related to Machine Vision have focused on the use of colour images as the main data source. With the recent emergence of low-cost 3D sensors equipped with image and 3D sensors the trend is changing. Last years, most research efforts in the area of computer vision are being focused on the use of 3D data as the main source of information. Moreover, these new sensors are able to provide both streams at high frame rates, similar to ones provided by traditional image sensors. The use of 3D data has allowed to face challenging problems that remained unresolved. However, processing 3D geometric information comes along with a higher computational cost, which is a current issue being researched.

Furthermore, traditional computer vision techniques have used intermediate representations or high level abstractions capable of dealing with tracking and recognition problems of entities in a more efficient way. Also, these representations allowed to abstract sensor information in higher level units of representation, e.g. regions of special interest, graphs and labels. As we move into a 3D space, new representation techniques are necessary

to continue applying similar techniques. Until now, most existing 3D representation techniques are too focused towards the video game (virtual reality) or the manufacturing industries. Most of the existing techniques are computationally too expensive or are not able to deal with high levels of noise in the observation which is common in these new sensors. As just introduced, these new sensors have very high noise levels especially when these start operating in their ranging limits. So a representation model capable of dealing with high noise levels and possible artifacts is demanded.

The research described in this thesis was motivated by the need of a robust model capable of representing 3D data distributions obtained with low-cost 3D sensors. In addition, modelling of noise produced by these sensors must be considered. As these sensors are capable of providing a 3D data stream in real time, the representation model has also to be able to face this constraint. Therefore, a time constraint is incorporated to the search of a 3D representation model for computer vision applications.

This thesis proposes the use of Self-Organizing Maps (SOMs) as a 3D representation model. In particular, we propose the use of the Growing Neural Gas (GNG) method, which has been successfully used for clustering, pattern recognition and topology representation of various kind of data. Until now, Self-Organizing Maps have been primarily computed offline and their application in 3D data has mainly focused on free noise models without considering time constraints. The Growing neural gases have the ability to adequately represent the input space provided to them. Self-organising neural models have also the ability to provide a good representation of the input space. In particular, the Growing Neural Gas (GNG) is a suitable model because of its flexibility, rapid adaptation and excellent quality of representation. However, this type of learning is time consuming, specially for high-dimensional input data. Since real applications often work under time constraints, it is necessary to adapt the learning process in order to complete it in a predefined time. This thesis proposes a hardware implementation leveraging the computing power of modern GPUs which takes advantage of a new paradigm coined as General-Purpose Computing on Graphics Processing Units (GPGPU). Finally it is also proposed a GPU

implementation of a local shape descriptor capable of extracting geometric information from the scene that can be subsequently used to describe it. The proposed methods need to be applicable to different problems or applications in the area of computer vision such as the recognition and localization of objects, visual surveillance or 3D reconstruction.



Universitat d'Alacant  
Universidad de Alicante



# Resumen

---

La visión artificial o visión por computador es un campo en constante evolución. Desde los años 80 esta área ha tratado la capacidad de “entender” una escena de forma automática mediante la utilización de un computador. Esta capacidad de entender se desglosa en varias etapas, por ejemplo, la localización y el reconocimiento de objetos que aparecen en una escena, dónde fue capturada, la reconstrucción de un modelo 3D a partir de varias capturas, etcétera. Estas escenas han sido tradicionalmente capturadas utilizando sensores de imagen, permitiendo analizar características sobre el color o la disposición 2D de los elementos en la escena. Los sensores 3D también han sido utilizados pero en menor medida debido a que tradicionalmente su coste ha sido muy elevado, siendo inaccesible para muchos grupos de investigación. Por lo tanto, la mayoría de trabajos en visión artificial se han centrado en la utilización de imágenes RGB como principal fuente de datos. Con la reciente aparición de sensores 3D de bajo coste dotados de sensores de imagen y de información 3D la tendencia está cambiando. Ahora la mayoría de esfuerzos de investigación en el área de visión por computador se están dirigiendo hacia la utilización de datos 3D como principal fuente de información. Además, estos nuevos sensores son capaces de ofrecer información a una frecuencia de vídeo. Destacar que la utilización de datos 3D está permitiendo resolver muchos problemas que hasta ahora permanecían sin solución: reconocimiento de gestos en 6 grados de libertad, reconstrucción e interacción con entornos 3D, o la localización y mapeado simultáneo de robots en exteriores. Por otro lado, el procesamiento de información geométrica 3D acarrea un mayor coste computacional por lo que varias de las aplicaciones mencionadas ante-

riamente han sido implementadas sobre dispositivos aceleradores como la GPU. Kinect Fusion es un claro ejemplo de este tipo de implementaciones donde Graphics Processing Unit (GPU) actúa como procesador principal de cómputo.

Tradicionalmente, en visión por computador se han utilizado técnicas de representación intermedias capaces de resolver problemas de reconocimiento y seguimiento de entidades en la observación. Además, estas representaciones permitían abstraer la información de los sensores en unidades abstractas de alto nivel, así como zonas de la imagen de especial interés (ROI), grafos y etiquetas. Al movernos a un espacio 3D, son necesarios nuevos modelos de representación para poder seguir aplicando técnicas similares. Sin embargo, la mayoría de técnicas de representación 3D existentes están demasiado enfocadas hacia la industria del videojuego (realidad virtual) o la fabricación de modelos 3D siendo la mayoría demasiado costosas computacionalmente o poco robustas para modelar el ruido de estos nuevos sensores. Como acabamos de introducir, estos nuevos sensores presentan niveles de ruido elevados, sobre todo cuando se empieza a operar en los límites del rango de los mismos, por lo que es necesario un modelo de representación capaz de manejar ruido y posible información errónea.

La investigación descrita en esta tesis está motivada por la necesidad de plantear un modelo robusto de representación 3D capaz de representar la información obtenida con sensores 3D de bajo coste, así como el modelado del ruido que estos sensores producen. Además, debido a que estos sensores son capaces de proveer un flujo de datos 3D en tiempo real, a semejanza de las cámaras de vídeo tradicionales, surge un nuevo problema asociado al coste computacional de los nuevos métodos de computación geométrica: la computación en tiempo real. Por lo tanto, una nueva restricción se incorpora en la búsqueda de un modelo de representación 3D para visión por computador, la necesidad de encontrar un modelo computacionalmente robusto y eficiente.

En esta tesis se propone la utilización de mapas auto-organizativos (SOM) como modelo de representación 3D. En concreto proponemos la utilización de gases neuronales crecientes (GNG) , que han sido utiliza-

dos con éxito tanto para clustering y reconocimiento de patrones como para la representación topológica de conjuntos de datos principalmente en 2 dimensiones. Hasta ahora, los mapas auto-organizativos han sido principalmente utilizados sin restricciones temporales y su utilización sobre datos 3D se ha centrado en modelos sin ruido y sin restricciones temporales. Los gases neuronales crecientes tienen la capacidad de representar de forma adecuada el espacio de entrada que se les proporcione. Además, son un modelo eficaz debido a su flexibilidad, rápida adaptación y excelente calidad de representación. Sin embargo, este tipo de modelos son muy costosos computacionalmente, especialmente para datos de entrada con alta dimensionalidad. Debido a que muchas aplicaciones de visión por computador se ejecutan bajo restricciones temporales, es necesario adaptar el proceso de aprendizaje de la red neuronal para ser completado bajo un tiempo predefinido. En esta tesis se propone una implementación hardware aprovechando la capacidad de cómputo de las GPUs actuales bajo el paradigma GPGPU. Finalmente se propone también la implementación de un descriptor característico capaz de extraer información geométrica de la escena que pueda ser posteriormente utilizado para describirla. El método propuesto debe ser aplicable a distintos problemas o aplicaciones dentro del área de visión por computador tales como el reconocimiento, localización de objetos, la vigilancia visual o la reconstrucción 3D.

Universidad de Alicante



# Contents

---

<b>List of Figures</b>	<b>xxiii</b>
------------------------	--------------

<b>List of Tables</b>	<b>xxxvii</b>
-----------------------	---------------

<b>1 Introduction</b>	<b>1</b>
1.1 Introduction . . . . .	1
1.2 Motivation . . . . .	3
1.3 Related works . . . . .	4
1.3.1 Representation models . . . . .	5
1.3.1.1 Explicit forms . . . . .	6
1.3.1.2 Parametric forms . . . . .	7
1.3.1.3 Generalized cylinders . . . . .	8
1.3.1.4 Superquadrics . . . . .	9
1.3.1.5 Octree-based representation . . . . .	9
1.3.1.6 Polygon mesh . . . . .	10
1.3.2 Surface fitting methods . . . . .	13
1.3.2.1 Spatial subdivision methods . . . . .	14
1.3.2.2 Distance functions methods . . . . .	16
1.3.2.3 Warping methods . . . . .	17
1.3.2.4 Incremental reconstruction methods . . . . .	18
1.4 Proposal . . . . .	19
1.5 Goals . . . . .	20
1.6 Structure of the dissertation . . . . .	21

<b>2</b>	<b>3D Representation with Growing Self-organizing Networks</b>	<b>23</b>
2.1	Introduction . . . . .	24
2.2	Self-Organizing Maps applied to 3D representation . . . . .	27
2.2.1	Related works . . . . .	27
2.2.2	Neural Gases . . . . .	31
2.2.2.1	Neural Gas Algorithm . . . . .	31
2.2.2.2	Growing Neural Gas Algorithm . . . . .	34
2.3	GNG-based method to represent real-world noisy observations	38
2.3.1	Data acquisition . . . . .	38
2.3.2	Topology preservation . . . . .	45
2.3.2.1	Topology preservation measures . . . . .	46
2.3.3	Noise removal . . . . .	47
2.3.3.1	Improving normal estimation . . . . .	49
2.3.4	Filtering quality: input space adaptation . . . . .	50
2.3.4.1	Growing Neural Gas input space adaptation	52
2.3.4.2	Neural Gas input space adaptation . . . . .	56
2.3.5	Colour interpolation . . . . .	62
2.3.6	Point cloud sequences management . . . . .	67
2.3.7	3D surface reconstruction . . . . .	68
2.3.7.1	Extended CHL . . . . .	70
2.3.7.2	Inserting and deleting neurons . . . . .	73
2.3.7.3	Experiments . . . . .	73
2.3.8	Conclusions . . . . .	78
<b>3</b>	<b>Improving keypoint detection from noisy 3D observations</b>	<b>79</b>
3.1	Introduction . . . . .	80
3.2	Keypoint detectors/descriptors . . . . .	82
3.2.1	Keypoint detectors . . . . .	82
3.2.2	Feature Descriptors . . . . .	88
3.2.3	Feature matching . . . . .	92
3.3	Improving Keypoint detection . . . . .	93
3.3.1	Experiments . . . . .	95
3.4	Conclusions . . . . .	101

<b>4 GPGPU parallel Implementations</b>	<b>103</b>
4.1 Introduction . . . . .	104
4.2 GPGPU architecture . . . . .	106
4.3 GPU-based implementation of the GNG algorithm . . . . .	108
4.3.1 Estimating the upper bound of the acceleration factor	109
4.3.2 GPU-based implementation . . . . .	111
4.3.2.1 Euclidean distance calculation . . . . .	112
4.3.2.2 Parallel reduction . . . . .	113
4.3.2.3 Complexity . . . . .	113
4.3.2.4 Other optimizations . . . . .	115
4.3.2.5 Minimizing transfers approach . . . . .	115
4.3.3 Experiments . . . . .	119
4.3.3.1 Number of threads per block . . . . .	119
4.3.3.2 Speed-up 2 Min Parallel Reduction . . . . .	120
4.3.3.3 GNG learning speed-up factor . . . . .	122
4.3.3.4 GNG hybrid version . . . . .	124
4.3.3.5 Rate of adjustments per second . . . . .	124
4.3.3.6 Discussion . . . . .	126
4.4 Neural Gas implementation using GPUs . . . . .	126
4.4.1 Experiments . . . . .	128
4.4.1.1 Adjustments per second . . . . .	130
4.4.2 Discussion . . . . .	130
4.5 GPU-based tensor extraction algorithm . . . . .	131
4.5.1 RGB-D processing on the GPU . . . . .	135
4.5.1.1 Noise removal: Bilateral filtering . . . . .	136
4.5.1.2 Normal estimation . . . . .	138
4.5.2 Surface triangulation on the GPU . . . . .	138
4.5.3 Tensor computation on the GPU . . . . .	142
4.5.4 Experimental results . . . . .	147
4.5.4.1 Performance . . . . .	148
4.6 Conclusions . . . . .	151
<b>5 Applications</b>	<b>153</b>
5.1 Introduction . . . . .	154

<b>5.2</b>	Robotics: 6DoF pose registration . . . . .	154
5.2.1	Accelerating 6DoF egomotion using GNG . . . . .	154
<b>5.3</b>	Computer Vision: 3D object recognition . . . . .	159
5.3.1	Offline learning . . . . .	160
5.3.2	Online recognition . . . . .	160
5.3.3	3D object recognition results . . . . .	162
5.3.3.1	Recognition on scenes with a single object	163
5.3.3.2	Recognition on scenes with multiple objects and occlusions . . . . .	163
5.3.3.3	Performance . . . . .	168
<b>5.4</b>	Computer vision: Multi-GPU Based Camera Network Sys- tem using GNG . . . . .	171
5.4.1	Related works . . . . .	171
5.4.2	Multi-source Information Processing with GPU . . .	173
5.4.2.1	Privacy and security. Real time constraint	173
5.4.2.2	Multicore CPU and multiGPU implemen- tation . . . . .	174
5.4.3	Experiments . . . . .	176
<b>5.5</b>	CAD/CAM: Growing Neural Gas Landmark . . . . .	178
5.5.1	Modified GNG for Landmarking and 3D Reconstruc- tion . . . . .	180
5.5.2	Experiments . . . . .	181
5.5.2.1	2D filtered quality results . . . . .	187
5.5.2.2	3D reconstruction quality . . . . .	188
<b>5.6</b>	CAD/CAM: 3D Model Reconstruction using Neural Gases .	190
5.6.1	Object reconstruction using low cost sensors . . . . .	192
<b>5.7</b>	Conclusions . . . . .	193
<b>6</b>	<b>Conclusions</b>	<b>195</b>
6.1	Conclusions . . . . .	195
6.2	Contributions of the Thesis . . . . .	198
6.3	Publications . . . . .	200
6.4	Future Work . . . . .	204
<b>Appendices</b>		<b>207</b>

<b>A Computed RMS errors</b>	<b>209</b>
<b>B Resumen</b>	<b>215</b>
B.1 Antecedentes y estado actual . . . . .	215
B.2 Objetivos de la investigación . . . . .	218
B.3 Algoritmo red GNG . . . . .	219
B.4 Mapas auto-organizativos crecientes: Información 3D . . . . .	221
B.4.1 Eliminación de ruido . . . . .	222
B.4.2 Filtrado: adaptación al espacio de entrada . . . . .	223
B.4.3 Aprendizaje información de color . . . . .	229
B.5 Implementación GPU algoritmo GNG . . . . .	234
B.5.1 Cálculo distancias Euclídeas . . . . .	235
B.5.2 Reducción paralela . . . . .	237
B.5.3 Factor de aceleración . . . . .	239
B.5.4 Versión Híbrida GNG . . . . .	240
B.5.5 Número de ajustes por segundo . . . . .	241
B.6 Extracción de características 3D en tiempo real . . . . .	242
B.6.1 Procesamiento información RGBD en la GPU . . . . .	242
B.6.2 Eliminación de ruido: filtro bilateral . . . . .	244
B.6.3 Estimación de normales . . . . .	245
B.6.4 Triangulación de superficies en la GPU . . . . .	246
B.6.5 Computación del tensor 3D en la GPU . . . . .	248
B.6.6 Rendimiento . . . . .	252
B.7 Conclusiones . . . . .	253
<b>Bibliography</b>	<b>259</b>
<b>List of Acronyms</b>	<b>285</b>



# List of Figures

---

1.1	Common steps that are necessary to solve Computer Vision (CV) problems. 3D data modelling is one of the key problems within research works in 3D machine vision applications.	2
1.2	(Left) Point cloud captured from a manufactured object (builder helmet). (Right) 3D mesh generated from the captured point cloud)	5
1.3	Scheme that shows how representation models can be classified: from the point of view of the application area and the point of view of their geometric description.	6
1.4	Different modelled forms using fully, partial and free-form constrained Generalized Cylinders	8
1.5	Stanford Bunny model representation using the Octree model.	10
1.6	(Left) 3D mesh representing a complete shoe last model, it contains 3,500 vertices and 7,000 faces. (Right) 3D mesh obtained from a single view of a scene. It is comprised of 21,941 vertices and 33,470 faces.)	12
2.1	Rectangular lattice commonly used by the Kohonen's map as initial topology.	24
2.2	The Self-Organizing Map structure. Selection of a node and adaptation of neighbouring nodes of the neural network to the input data.	25
2.3	Initial, intermediate and final states of the NG learning algorithm	34

2.4	(a) Delaunay triangulation, (b) Induced Delaunay triangulation . . . . .	34
2.5	Initial, intermediate and final states of the GNG learning algorithm . . . . .	37
2.6	GNG learning algorithm. . . . .	37
2.7	Mobile robots used for experiments. Left: Magellan Pro unit used for indoors. Right: PowerBot used for outdoors. . . . .	39
2.8	Left: Depth map. Center: RGB Map. Right: Projected point cloud. . . . .	41
2.9	3D sensors used for experiments. From left to right: Sick laser unit LMS-200, Time-Of-Flight SR4000 camera and Microsoft Kinect . . . . .	42
2.10	Various scenes represented using the GNG algorithm. From top to bottom: scene captured using the Time-Of-Flight SR4000 camera, the LMS-200 Sick laser and the Microsoft Kinect device . . . . .	43
2.11	Different objects represented using the GNG algorithm. Top object belongs to the public dataset [Tombari et al., 2010a]. Rest of objects were captured using the Kinect device in our own laboratory. . . . .	44
2.12	Noise causes error in the estimated normal . . . . .	50
2.13	Normal estimation comparison. Top: Normal estimation on raw point cloud. Bottom: Normal estimation on filtered point cloud produced by the GNG method . . . . .	51
2.14	Synthetic scene. Top: Ground truth without noise. Bottom: simulated Kinect view with added noise: $\sigma = 0.5$ . . . . .	53
2.15	Top: bunny model (34,834 points). Bottom: rhino model (79,934 points). . . . .	54
2.16	Filtering quality using 10,000 points. GNG vs Voxel Grid comparison. From top to bottom: Noisy model $\sigma = 0.4$ , Original CAD model, filtered model using GNG method and filtered model using Voxel Grid. . . . .	55

2.17	Reconstructed 3D models using NG method. From top to bottom: Bunny, armadillo and horse. The number of neurons of the reconstructed models is 5k, 8k and 15k. . . . .	57
2.18	NG parameters study using Stanford bunny model. Three different levels of noise $\sigma$ are applied . . . . .	59
2.19	Horse model reconstructed incrementally from an unorganized point cloud of 148k points. From left to right, the number of vertex of the reconstructed mesh is 1k,2.5k and 8k. . . . .	60
2.20	NG parameters study using Stanford bunny model. Three different levels of noise $\alpha$ are applied. . . . .	60
2.21	Filtering quality using 10,000 points. NG vs Voxel Grid comparison. Top left: original noise-free model. Top right: noisy model $\alpha=0.6$ mm. Bottom left: filtered model using NG method. Bottom right: filtered model using Voxel Grid. . . . .	61
2.22	Several objects and scenes are down-sampled using the Colour-GNG representation. (a),(b),(c) show original pointclouds. (d),(e),(f) show down-sampled point clouds using the proposed method. . . . .	63
2.23	Colour interpolation. Colour assigned to each neuron (large circles) is calculated as the averaged weighted sum of input space samples (small circles) within a search radius. . . . .	64
2.24	Mario figure is down-sampled using the Colour-GNG method. Results are similar to those obtained with the colour interpolation post-processing step. . . . .	65
2.25	Two different scenes captured using the Kinect sensor are represented using the Colour-GNG method. Results are similar to those obtained with the colour interpolation post-processing step. . . . .	66
2.26	An improved workflow to manage point cloud sequences using the GNG algorithm. . . . .	67
2.27	Different views of reconstructed models using an existing GNG-based method for surface reconstruction. Post-processing steps were avoided causing gaps and holes in the final 3D reconstructed models. . . . .	69

2.28 Considered situations for edge and face creation during the extended CHL . . . . .	72
2.29 Edge removal constraint based on the Tales sphere. Left: The triangle formed by these 3 neurons is close to a right triangle, therefore it is not removed. Right: The edge connecting $s_1$ and $n_i$ is removed as the angle formed by vectors $s_2 - s_1$ and $n_i - s_1$ is larger than $3/4\pi$ . Moreover, the triangle formed by these edges is also removed. . . . .	72
2.30 Face creation process during the insertion of new neurons. Left: neuron insertion between the neuron $q$ with highest error and its neighbour $f$ with highest error. Right: four new triangles and two edges are created considering $r$ , $q$ and $r$ . . . . .	73
2.31 Reconstructed models using our extended GNG method for face reconstruction and without applying post-processing steps. Top: Stanford bunny. Bottom: builder helmet. . . . .	74
2.32 The proposed method for 3D reconstruction was also applied to 3D models of people and a foot of a person. Top: 3D model of a person. Bottom: digitized foot. . . . .	75
2.33 The proposed method for 3D reconstruction was also applied to partial 3D views of scenes. Left: Noisy point clouds captured using the Kinect sensor. Right: 3D reconstruction using the proposed method. . . . .	76
2.34 Different 3D reconstructions of a the builder helmet model using various network sizes. Left: 3D reconstruction using 250 neurons and 200 input patterns. Middle: 3D reconstruction using 1000 neurons and 500 input patterns. Right: 3D reconstruction using 2500 neurons and 1000 input patterns. . . . .	77
2.35 Small gaps produced by our extended GNG method for 3D surface reconstruction. . . . .	77

3.1	Keypoints are detected based on the normal gradients i.e. curvature SIFT keypoint estimation. Detected keypoints are marked using blue dots. (Left) Complete Stanford bunny model. (Right) 3D partial view of Mario Bros model . . . . .	83
3.2	Scene captured using the Kinect sensor. Keypoints are detected based on the normal gradients i.e. curvature SIFT keypoint estimation. Detected keypoints are marked using blue dots. . . . .	84
3.3	Keypoints are detected using the 3D Harris detector and its variants. Detected keypoints are marked using blue dots. Two different models have been tested: on the left side the Stanford Bunny model and on the right side a builder helmet captured using the Kinect sensor. From top to bottom: Harris3D, Tomasi3D and Noble3D. . . . .	85
3.4	Same models than in Figure 3.3. From top to bottom: Lowe3D and Curvature3D. . . . .	86
3.5	Keypoints are detected using the 3D Harris detector and its variants. Detected keypoints are marked using blue dots. A partial view of the complete scene has been tested. From left to right and from top to bottom we used Harris3D, Tomasi3D, Noble3D and Lowe3D. . . . .	87
3.6	Detected keypoints using the ISS algorithm on different 3D observations. (Left) partial view of a scene. (Right) complete model of a builder helmet. Both were captured using the Kinect sensor. . . . .	89
3.7	Diagram of the PFH computation for a query point $p_q$ . The query point is marked red and it is in the middle of the diagram. Neighbour points $k$ are selected within a radius $r$ , which is marked using a dotted line. . . . .	90
3.8	Diagram of the FPFH computation for a query point $p_q$ . The query point is marked in red and it is in the middle of the diagram. Neighbour points $k$ are selected within a radius $r$ , which is marked using a dotted line. . . . .	91

3.9	Spherical grid used by the SHOT descriptor to calculate normals differences and therefore to build a weighted histogram of normals. . . . .	92
3.10	Correspondences matching between two different scenes are computed. Top: correspondences without performing rejection step. Bottom: After applying RANSAC based rejection step false positives are removed. . . . .	94
3.11	General scheme of most 3D computer vision applications. Proposed GNG-based algorithm presented in Chapter 2 is used to obtain a robust representation of the input space (highlighted step). This representation will improve following steps as the keypoint detection. . . . .	95
3.12	Registered views of the indoor scene. . . . .	96
3.13	Correspondences matching computed on different input data. Top: raw point clouds. Middle: reduced representation using the GNG (20,000 neurons). Bottom: reduced representation using the GNG (10,000 neurons) . . . . .	98
3.14	Registration example performed with the Lowe keypoint detector and the FPFH descriptor using as an input a GNG representation with 20,000 neurons. . . . .	100
4.1	CUDA compatible GPU Architecture. . . . .	106
4.2	Percentage of instructions that are executed each stage for 20000 neurons and 1000 input patterns per iteration. . . . .	109
4.3	GNG learning algorithm remarking the parallel stages. . . . .	112
4.4	Example of Parallel Reduction Algorithm execution. . . . .	114
4.5	First approach, CUDA workflow. . . . .	116
4.6	Approach that minimize transfers. CUDA workflow. . . . .	117
4.7	Percentage of time spent in execution of the algorithm for memory transfers. . . . .	118
4.8	Execution time depending on the number of threads per block.	120
4.9	Speed-up of <i>2MinParallelReduction</i> implementation using different graphic cards. . . . .	121

4.10	Example of GPU and CPU GNG runtime, and speed-up for different devices. . . . .	123
4.11	Example of CPU and Hybrid GNG runtime for different devices. . . . .	125
4.12	Rate of adjustments per second performed by different GPU devices and CPU. . . . .	125
4.13	(Top) Runtime execution (GPU GTX480) (lines) and speed-up (bars) for a fixed number of neurons (10,000) and different number of iterations. (Bottom) Runtime execution (GPU GTX480) (lines) and speed-up (bars) for a fixed number of iterations (100,000) and different number of neurons. . . . .	129
4.14	Computed iterations per second. NG CPU vs GPU.(GTX480). . . . .	131
4.15	General system overview. Steps coloured in dark are computed on the GPU. . . . .	134
4.16	(a) Point cloud obtained after transforming depth and color maps provided by the Kinect sensor. (b) Normal estimation. (c) Surface reconstruction. (d) Feature descriptor extraction: 3D tensor computed over a partial view . . . . .	135
4.17	Bottom row. Normal estimation using the original map. Top row. Normal estimation using the filtered map (bilateral filtering). It is observed that the normal estimation was improved resulting in more stable normal directions. This effect is visually observed on plane surfaces where the estimated normals using a noisy map were much less stable than normals computed over a filtered map. . . . .	139
4.18	Left: Point triangulation condition. This images shows how the condition to create an edge establishes that the angle $\theta_{pov}$ formed by vectors $v_{v_p,p}$ and $v_{p,q}$ must be over a threshold $\epsilon_{\theta_{pov}}$ . This threshold assures that points are not occluded among themselves. The Euclidean distance between $p$ and $q$ also must be smaller than an established threshold $T_d$ , dynamically calculated according to mesh resolution and its standard deviation. Right: Triangles are established by left cut checking constraints between points. . . . .	141

4.19 Point cloud meshing using the proposed method. Note that some holes and gaps still exist in the approximate surface reconstruction due to the noisy information obtained from the Kinect sensor. . . . .	142
4.20 Detecting a valid pair using mutual angle $\theta_d$ and mutual distance $d$ constraints. The midpoint of the pair will be used later to define a 3D grid for tensor computation. . . .	144
4.21 (a) Launching $Dim_x \times Dim_y \times Dim_z$ threads in parallel where each GPU thread represent a voxel of the grid. (b) Each thread with indexes $i, j, k$ calculates the area of intersection between the mesh and its corresponding voxel using Sutherland Hodgman's polygon clipping algorithm. Taking advantage of thread indexes, the calculated area is stored in a flattened vector. . . . .	147
4.22 Profiling computation of tensors using streams (top) and without streams (bottom). On the top of the figure (using streams) it can be seen how up to 6 kernels run simultaneously occupying all available resources on the GPU. . . . .	150
4.23 Tensors computation runtime using different number of streams. Number of tensors is fixed to 200 and the device used is the NVIDIA GTX 480. Using CUDA streams runtime was considerably improved. . . . .	151
4.24 Speed-up achieved compared to sequential CPU version for the computation of a different number of tensors. . . . .	151
5.1 Applying GNG to SR4000 data. The original data set is shown at the top right corner. The intensity image captured by the camera is shown at the bottom right corner. Left, the resulting GNG (green) can be observed over the original data set. . . . .	156
5.2 Top, planar patches extracted from SR4000 camera. Bottom, use of GNG to improve planar patches extraction. . . .	157

5.3	Planar based 6DoF egomotion results. Left image shows map building results without using GNG while the results shown on the right are obtained after computing a GNG mesh.	158
5.4	Model library consisted of 7 real models. Each object consists of several partial views. For every partial view of a model, tensors are computed (blue lines) describing the model by extracting 3D surface patches (tensors).	161
5.5	Confusion matrix for extracted tensors from different views of each object model contained in the dataset. The averaged recognition rate (TP) is 84%, wrong matches (FP) 16% and false negatives (FN) 0%. Cells numbers indicate the number of times (percentage) that an extracted tensor was successfully classified to the corresponding model. Presented results show that method provides high discrimination capability.	164
5.6	Object recognition is performed on scenes with different level of occlusion. The models are occluded by objects stored and non-stored in the library.	166
5.7	Scenes with multiple object and occlusions. These scenes were used in experiments presented in Section 5.3.3.2 achieving a recognition rate of 82%. Scenes presented a different level of clutter and occlusions caused by stored and non-stored objects.	167
5.8	3D object recognition in cluttered scenes. Different partial views of two scenes are showed. Multiple labels are shown as all computed tensors are evaluated and matched against the library model.	170
5.9	GNG graph representation	174
5.10	MultiGPU multi-camera workflow	175
5.11	Example of single image multi-target visual surveillance system	176
5.12	Execution times and speed-up for different GNG learning parameters and different number of cameras	177

5.13 Rate of adjustments per second for different GNG learning parameters and different number of cameras using MultiGPU-MultiCPU and single threaded CPU versions . . . . .	178
5.14 Mechanical (left) and optical (right) digitizers for shoe lasts. . . . .	179
5.15 Typical sequence of sections on a shoe last and detail. . . . .	180
5.16 Shoe lasts used in the experiments. On the left, last 1; center, last 2; on the right, last 3 . . . . .	181
5.17 Error calculation process. . . . .	182
5.18 Voxel Grid versus GNG: Mean error and max error (millimetres). Different number of input signals $\lambda$ for the GNG method. Left: mean error for all the sections. Right: maximum error for all the sections. . . . .	184
5.19 Voxel Grid versus GNG mean errors. . . . .	185
5.20 Voxel grid versus GNG distribution of error. . . . .	185
5.21 Comparison of reconstruction times without filtering (left) and filtering (right). . . . .	186
5.22 Computation time for different models and sections. . . . .	187
5.23 Detail section: overlay (top) areas with noise. GNG sorting and filtering (bottom). . . . .	188
5.24 Landmark points using GNG for a section with 180 and 200 points. . . . .	189
5.25 GNG vs VG topological preservation comparison. . . . .	189
5.26 Voxel Grid (left) vs GNG (right). . . . .	189
5.27 Virtual Digitizing (left) vs GNG (right). . . . .	190
5.28 Builder helmet reconstructed using the proposed method. Left: Point cloud obtained using the Kinect sensor and the marker based-approach. Center and right: Different views of the 3D reconstructed surface using the NG method. . . . .	192
5.29 (Left) RGB image of the builder helmet. (Right) Different views of the final reconstructed builder helmet using the proposed method. . . . .	193

A.1	Mean, median, min and max RMS errors of the estimated transformations. These values have been extracted from results presented in Tables A.1, A.2, A.3 and A.4. . . . .	214
B.1	Comparativa estimación normales. Arriba: Estimación de normales sobre la nube de puntos sin filtrar. Abajo: estimación de normales sobre la representación generada por el método GNG. . . . .	224
B.2	Escena sintética. Arriba: Vista parcial generada sin ruido. Abajo: Vista parcial generada simulando un dispositivo Kinect con ruido Gaussiano $\sigma = 0.5$ y $\mu = 0.0$ . . . . .	226
B.3	Izquierda: modelo 3D conejo (34,834 puntos). Derecha: modelo 3D rinoceronte (79,934 puntos). . . . .	227
B.4	Calidad de filtrado utilizando 10,000 ppuntos. Comparativa GNG vs Voxel Grid. De arriba a abajo: Modelo original, modelo con ruido $\sigma = 0.4$ , modelo filtrado utilizando el método GNG y modelo filtrado utilizando VG. . . . .	228
B.5	Diferentes escenas representadas utilizando la red neuronal GNG. De arriba a abajo: escenas capturada utilizando la cámara de tiempo de vuelo SR4000, un laser LMS-200 Sick montado sobre un brazo robótico y el sensor RGBD Kinect de Microsoft . . . . .	230
B.6	Distintos objetos representado utilizando la red neuronal GNG. El modelo 3D de Mario pertenece al conjunto de datos presentado en [Tombari et al., 2010a]. El resto de objetos fueron capturados utilizando el dispositivo Kinect de Microsoft. . . . .	231
B.7	Representación de distintos objetos y escenas utilizando el método propuesto Colour GNG. En (a),(b),(c) se muestran las nubes de puntos originales. En (d),(e),(f) se presentan las versiones reducidas utilizando el método GNG con información de color. . . . .	233

B.8	Método de post-procesado para interpolación de color. El color asignado a cada neurona (círculos grandes) es calculado como la media ponderada de los patrones de entrada vecinos (circulos pequeños) en un radio de búsqueda. Radio de búsqueda vecinos más cercanos (circunferencias con línea discontinua). . . . .	234
B.9	Representación reducida con información de color utilizando el método Colour GNG. Los resultados son visualmente similares a los producidos utilizando una interpolación ponderada sobre el espacio de entrada. . . . .	235
B.10	Representación reducida con información de color de dos escenas diferentes. . . . .	236
B.11	Algoritmo GNG. Se han marcado las etapas paralelizadas. .	237
B.12	Ejemplo de reducción paralela. . . . .	238
B.13	Tiempo de ejecución en la CPU y GPU. Se muestran los factores de aceleración obtenidos sobre distintos modelos de GPU. . . . .	240
B.14	Tiempos de ejecución implementaciones CPU e híbrida sobre distintas GPUs. . . . .	241
B.15	Tasas de ajustes por segundo obtenidas para distintos procesadores. . . . .	242
B.16	Izquierda. Mapa de profundidad. Centro. Mapa RGB. Derecha. Nube de puntos proyectada. . . . .	244
B.17	Arriba: Estimación de normales sobre el mapa filtrado (filtro bilateral). Abajo: Estimación de normales sobre el mapa original. Se puede observar como la estimación de las normales mejora, obteniendo normales mas estables, sobre todo en las zonas planas, donde el ruido del mapa original tiene como consecuencia unas normales inestables. . . . .	246

- B.18 Condición para triangular vértices. En esta imagen se puede apreciar cómo la condición de arista válida establece que el ángulo  $\theta_{pov}$  formado por los vectores  $v_{v_p,p}$  y  $v_{p,q}$  debe ser superior a un umbral establecido  $\epsilon_{\theta_{pov}}$ . Este umbral nos asegura que los puntos no producen una oclusión entre ellos. La distancia euclídea entre  $p$  y  $q$  también debe ser inferior a un umbral  $T_d$ , calculado de forma dinámica de acuerdo a la resolución de la nube de puntos y su desviación estándar. . . . . 247
- B.19 Reconstrucción de la superficie de la escena utilizando el método propuesto. Se puede observar como la reconstrucción presenta algunos huecos debido a que el sensor no ofrece información de esas zonas, bien por encontrarse fuera del área de visión, o por ser una superficie especular. . . . . 249
- B.20 (a) Ejecución de  $Dim_x \times Dim_y \times Dim_z$  hilos en paralelo donde cada hilo de la GPU representa un voxel del grid. (b) Cada hilo con índices  $i, j, k$  computa el área de intersección entre la superficie triangulada y su voxel correspondiente utilizando el algoritmo de Sutherland Hodgman's para cálculo de intersección de polígonos. Aprovechándose de los índices de los hilos proporcionados por CUDA el area es almacenado un vector unidimensional. . . . . . . . . . . 252

Universidad de Alicante



# List of Tables

---

2.1	Input space adaptation MSE for different models. Voxel Grid versus GNG. Numbers in bold provide the best results.	54
2.2	Colour Mean error between computed colours using a post-processing interpolation step and the Colour-GNG. Various scenes and objects were tested obtaining less than four units of error even with colourful scenes.	65
3.1	RMS deviation error (meters) is obtained using different detector-descriptor combinations. Combinations are computed on the original point cloud (raw), and different filtered point clouds using Voxel Grid, Uniform Sampling and the proposed method. Keypoint detector search radius equals to 0.05 meters. Feature extractor search radius equalss to 0.02 meters.	99
3.2	Mean, median, minimum and maximum RMS errors of the estimated transformations with respect to different keypoint detectors. These values were extracted from results presented in Table 3.1	101
4.1	Percentage of executed instructions at each stage of GNG algorithm	110
4.2	Overall maximum acceleration estimated using Amdahl's law and assuming a factor 20 of speed-up regarding to a fraction $p$ of the algorithm. $s$ is the fraction of the algorithm that cannot be executed in parallel (sequential).	111

4.3	CUDA capable devices used in experiments . . . . .	119
4.4	Theoretical overall speed-up obtained for GNG algorithm using speed-up obtained in stage 3. This experiment was executed on a NVIDIA GTX 480. . . . .	122
4.5	Theoretical overall speed-up and obtained overall speed-up using the device GTX480. . . . .	123
4.6	Runtimes (top) and speed-ups (bottom) obtained using the accelerated version of the NG algorithm on different 3D models. All runtimes are measured in seconds. . . . .	130
4.7	CUDA capable devices used in experiments . . . . .	147
4.8	Runtime comparison and speed-up obtained for proposed methods using different graphics boards. The fastest run times were achieved by the graphics board NVIDIA GTX480. Runtimes are averaged over 50 runs. . . . .	149
5.1	Runtime comparison and speed-up obtained for matching process. As the size of the model library is increased the speed-up achieved is slightly larger. Runtimes are averaged over 50 runs. . . . .	168
5.2	Accumulated mean error for the whole shoe last (millimeters): Voxel Grid versus GNG. Different number of input patterns for the GNG method. . . . .	183
5.3	Maximum error for the whole shoe last (millimeters): Voxel Grid versus GNG. Different number of input patterns for the GNG method. . . . .	183
A.1	RMS deviation error (meters) is obtained using different detector-descriptor combinations. Combinations are computed on the original point cloud (raw), and different filtered point clouds using Voxel Grid, Uniform Sampling and the proposed method. Keypoint detector search radius equals to 0.02. Feature extractor search radius equalss to 0.02. (metres) . . . . .	210

A.2	Same experiment that presented in Table A.1 but changing the keypoint detector search radius to 0.05 and the feature extractor search radius equalss to 0.02. (metres) . . . . .	211
A.3	Same experiment that presented in Table A.1 but changing the keypoint detector search radius to 0.1 and the feature extractor search radius equalss to 0.02. (metres) . . . . .	212
A.4	Same experiment that presented in Table A.1 but changing the keypoint detector search radius to 0.15 and the feature extractor search radius equalss to 0.02. (metres) . . . . .	213
B.1	MSE de adaptación al espacio de entrada para distintos modelos. VG vs GNG. Los valores en negrita destan los mejores resultados obtenidos. . . . .	225
B.2	Error medio entre el color calculado utilizando la técnica de post-procesamiento (interpolación ponderada) y el método Color GNG. . . . .	233
B.3	Dispositivos compatibles con CUDA utilizados en los experimentos. . . . .	239
B.4	Modelo GPU integrado en un ordenador portátil utilizado en los experimentos . . . . .	253
B.5	Comparación tiempos de ejecución y factor de aceleración obtenidos en los distintos métodos implementados en la GPU. Los mejores tiempos de ejecución se obtuvieron para el modelo GTX480, el cual posee el mayor número de núcleos de procesamiento. . . . .	253



# List of Algorithms

---

1	Pseudo-code of the extended CHL stage. . . . .	71
2	Pseudo-code of the GPU-based 3D tensor extraction algorithm. $d_{\_}$ prefix means that variable is allocated in the GPU memory. . . . .	134
3	Pseudo-code of the GPU-based point cloud projection algorithm. . . . .	136
4	Pseudo-code of the GPU-based normal estimation algorithm	139
5	Pseudo-code of the GPU-based surface triangulation algorithm.	143
6	Pseudo-code of the GPU-based 3D tensor computation algorithm . . . . .	146

Universitat d'Alacant  
Universidad de Alicante



---

# Chapter 1

# Introduction

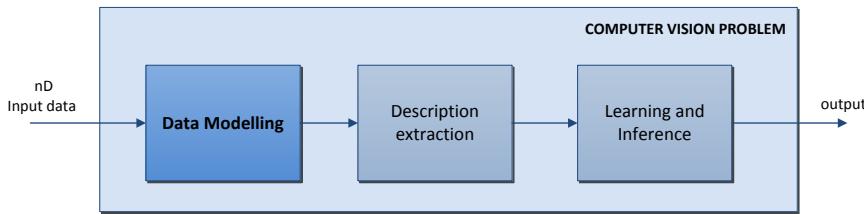
---

This first chapter introduces the main topic of this thesis. The chapter is organized in six different sections: Section 1.1 establishes the framework for the research activity proposed in this thesis. Section 1.2 introduces the motivation of this work. Next, Section 1.3 elaborates a state-of-the-art of existing representation models traditionally applied on 3D data. Section 1.4 introduces the proposal developed in this thesis. Section 1.5 presents the main goals of this work. Finally, Section 1.6 details the structure of the dissertation.

## 1.1 Introduction

In this doctoral thesis a theoretical and practical research were conducted, exploring the problem of finding a robust representation model for 3D data distributions. The research is oriented to the study of representation models for 3D data distributions usually captured from the real world, which implicitly are comprised of complex structures and non-linear models. Moreover, with the advent of low cost 3D sensors, robust representation models should deal with noisy data and outliers.

Figure 1.1 shows an overview of common steps in most computer vision applications. Most 3D vision problems require the use of an effective way of representation (data modelling), which will be a key step for later processing stages. This makes the problem harder to solve but often helps to overcome later constraints usually demanded as real-time processing rates



**Figure 1.1:** Common steps that are necessary to solve Computer Vision (CV) problems. 3D data modelling is one of the key problems within research works in 3D machine vision applications.

and management of data sequences.

This thesis has been conducted under the frame of the following projects:

- *Visual surveillance systems for the identification and characterization of anomalous behaviour* (GV/2011/034). Project financed by the *Valencian Government* in Spain.
- *Visual surveillance systems for the identification and characterization of anomalous behaviour in restricted environments under temporal constraints.* (GRE09-16). Project financed by the *University of Alicante* in Spain.

Moreover, part of the work presented in this thesis was done during my stay in the computer vision group at the University of Edinburgh. This 5 month stay was funded by the Valencian government BEFPI/2012/056, and by the European network of excellence on High Performance and Embedded Architecture and Compilation (HiPEAC). The work carried out during my stay was partially led by the professor Robert B. Fisher, which is head of the computer vision group.

This chapter is structured as follows: in Section 1.2 the motivation and appropriateness of the present work. Section 1.3 introduces an extensive review of the literature on 3D representation methods. In Section 1.4 the proposed 3D representation method is presented. Next, in Section 1.5, the goals to be achieved are described in detail. Finally, Section 1.6 describes the structure of the dissertation.

## 1.2 Motivation

This document is the result of the doctoral studies carried out within the doctoral program *Tecnologías de la sociedad de la información* taken between 2011 and 2014, at the *Computer Technology department - DTIC*, of the *University of Alicante* (Spain). This thesis was derived from a three-year PhD fellowship awarded to me by the Valencian Government.

The motivation for this thesis project arises from the participation and collaboration in many different projects in related computer vision problems.

In the visual surveillance project, a flexible, fast and accurate representation model was required in order to represent and track scene observations in real-time. The overall objective of the proposed research was the specification and design of vision-based services aimed to facilitate the monitoring of an area with poor visibility. It is proposed the specification and testing of a representation model able to run in real time on graphics processing units (GPU). The representation model will serve as a basis to identify and characterize abnormal behaviour of one or more mobile agents in restricted environments.

During my collaboration with the Robotics and 3D Vision research group I worked in the design of a representation model capable of dealing with noise produced by typical 3D sensors as Time of Flight (ToF) and RGB-D cameras. The main aim was to use this model for cooperative Simultaneous Localization and Mapping (SLAM) in large scale environments. Moreover, as the SLAM problem involves solving other problems as the registration of multiple scene views, downsampling and filtering steps become crucial for finding a valid matching during this process. Therefore, it was required a 3D representation model able to cope with all these mentioned constraints and able to keep the geometry of real-word scenes.

In addition, due to my participation in the development of a rapid prototyping method for shoe last manufacturing, the necessity of an overly accurate representation model is presented. This project is tackled in conjunction with the Footwear Technological Institute (INESCOP) and a CAD-CAM research group (UNICAD) from the University of Alicante.

Rapid prototyping techniques are based on the reuse of manufactured footwear lasts to be modified with CAD systems leading rapidly to new shoes models. In this project, it was presented the problem of shoe last reconstruction. A method able to work with different sections and establishing a number of fixed landmarks onto those sections was demanded. Those free-noise landmarks are later used to perform 3D surface reconstruction. The process of finding those landmarks is not trivial as the input space presents outliers and noise generated by the sensor. Furthermore, selected landmarks must preserve the topology of the original input space.

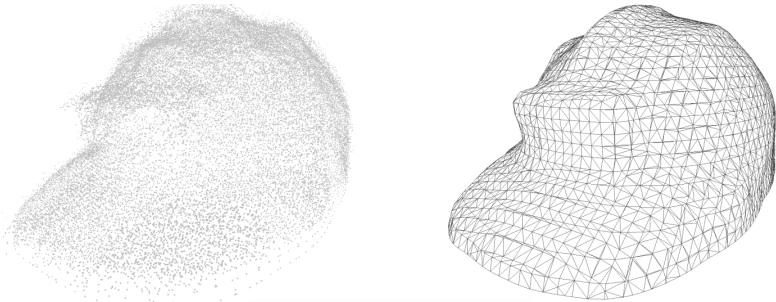
The presented problems have in common the necessity of finding a model with different features: rapid adaptation, good quality of representation and a structure that facilitates the feature extraction process that will define the underlying observation. All these constraints have to be considered. The problem is described in mathematical terms as the problem of finding a representation model  $m^*$  that approximates the physical surface  $P'$ .  $f$  is a surface fitting function that determines how to fit the representation model over the surface. The main differences between existing solutions are the different constraints required for  $f$  and the different representations of  $m_i$  that are chosen from  $M$ .  $M$  is defined as the set of different representation models.

$$m^* = \operatorname{argmax}_{m_i \in M} f(m_i, P') \quad (1.1)$$

### 1.3 Related works

3D models of objects and scenes have been extensively used both in computer vision and in computer graphics. In graphics, the object need to be represented with a structure suitable for rendering and display. The most common structure is the 3D mesh, a collection of polygons consisting of 3D points and the edges that inter-connect them. Graphics hardware usually supports this representation. For smoother and/or simpler surfaces, other graphics representations include quadric surfaces [Gotardo et al., 2004], B-spline surfaces [Gregorski et al., 2000], and subdivision sur-

faces. In addition to 3D shape information, graphics representations can contain color/textured information. Figure 1.2 shows a rough point cloud and a 3D mesh of a builder helmet from the same viewpoint.



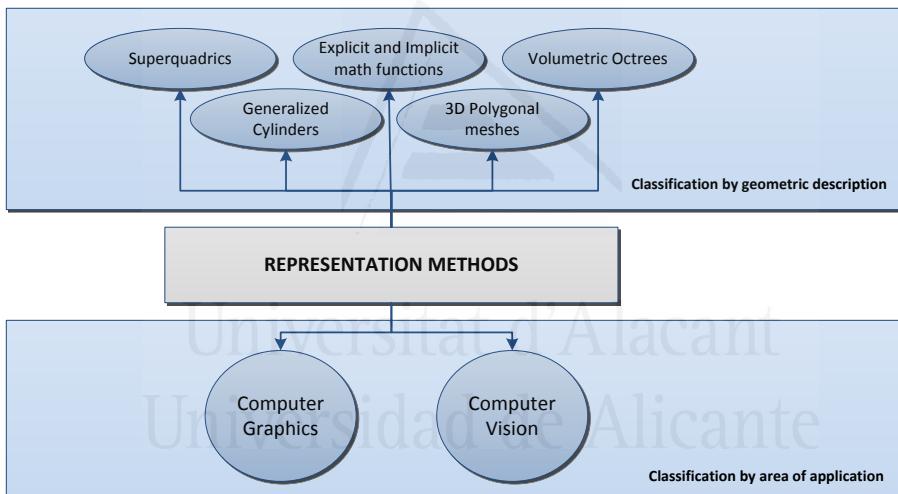
**Figure 1.2:** (Left) Point cloud captured from a manufactured object (builder helmet). (Right) 3D mesh generated from the captured point cloud

### 1.3.1 Representation models

A key issue in 3D vision problems is the election of an effective way of representation. Moreover, in computer vision the representation model must be suitable to be used in many applications: object recognition and tracking, outdoor scene recognition, 3D registration, etc. which means that there must be some potential correspondence between the representation and the features that can be extracted from the data. Most 3D representation algorithms are not general enough to handle such a variety of features, but instead were designed for a particular representation. An ideal representation model should be automatically acquired via a learning process using only topological and temporal observations. Moreover, this model should vary over the time representing topological shape deformations and allowing tracking of desired observations. Therefore, it is important to find a common representation method suitable to different computer vision problems.

One of the earlier works in computer vision [Roberts, 1965] was based on the recognition of polyhedral objects using simple wire-frame mod-

els and matching to straight line segments extracted from images. Line-segment-based models have remained popular even today, but there are also a number of alternatives that attempt to more closely represent the data from objects with high curvature and even free-form surfaces. In general categories, there are geometric representations in terms of points, lines, and surfaces; symbolic representations in terms of primitive components and their spatial relationships; and functional representations in terms of functional parts and their functional relationships. Figure 1.3 shows various categories where 3D representation models can be placed according to their application area and geometric description. Most common methods for representing 3D data are presented in next section.



**Figure 1.3:** Scheme that shows how representation models can be classified: from the point of view of the application area and the point of view of their geometric description.

### 1.3.1.1 Explicit forms

Explicit mathematical forms are the most simple way to fit surfaces. A explicit mathematical form is given by a function  $z = f(x, y)$ . The problem is that for certain values of variables  $x$  and  $y$  may exist different values for variable  $z$ . This technique is used in computer vision [Jain et al., 1995]

for range images but it is not widely used as the parametric and implicit forms that we will discuss below. Explicit form is used for range images where  $(x, y)$  are image plane coordinates and  $z$  is the distance parallel to the  $z$  axis in camera coordinates. Unfortunately, using this simple method only few shapes can be modelled and therefore the model is not suitable for computer vision problem like scene understanding where dealing with natural forms is mandatory.

### 1.3.1.2 Parametric forms

Parametric forms belong to the group of complete mathematical forms. This means that the geometric description of the model is explicit and therefore the surface can be entirely described from the representation. Furthermore, an arbitrary pose of the represented object or scene can be generated using this model. Parametric forms have been widely used in Computer Aided Design and Manufacturing (CAD/CAM) since they can be easily sampled, their representation power is meaningful and also the representation can be extracted from existing point clouds [Kitson, 1989].

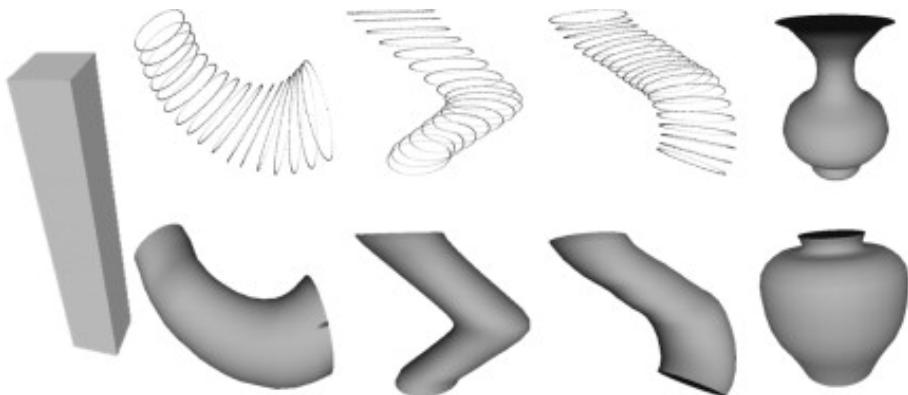
A parametric form of a surface can be defined as:

$$x = f_1(u, v), y = f_2(u, v), z = f_3(u, v) \quad (1.2)$$

where  $f_1, f_2$  and  $f_3$  have as arguments the parametric variables  $u$  and  $v$ . The Non-Uniform Rational B-spline (NURBS) is one of the most extended and used parametric forms but also exist other ones. Parametric surfaces have a number of interesting properties but when they are applied to fitting problems and deal with complex observations it can be hard to find a suitable parametrisation. This is the main reason why it is difficult to apply this common method to computer vision problems, in particular this is reviewed in depth in [Besl and McKay, 1992], but we can conclude that finding trimming curves that fit the surface is not an easy problem and is not unique. Parametric forms are usually used to estimate an initial model specification from which a wire or polygonal mesh is employed in machine vision systems.

### 1.3.1.3 Generalized cylinders

Another representation model often used in computer graphics is the Generalized Cylinder (GC) model. A GC is a volumetric primitive defined by a space curve axis and a cross section function at each point of the axis. The cross section is extended along the axis creating a solid model. A GC model of an object or a more complicated observation like an outdoor scene is comprised of combined GC descriptions and information about how they are interconnected. We find different works where this model has been successfully applied to different problems. In [Bloomenthal, 1985] it is presented a method to represent botanical trees, given three-dimensional points and connections, limbs are modelled as GC whose axes are space curves that interpolate the points. In a more recent work, [Vinayak et al., 2013], it is presented a novel interaction system for creative expression of 3D shapes through the naturalistic integration of human hand gestures. GC models are used as the basic representation model for the creation of these 3D shapes. These GC models do not have enough detail level due to their symmetric shape. For that reason, they are usually only applied to represent human-made objects or simple observations. Moreover, they are passive as they do not react to externally applied forces, sacrificing the possibility of representing detailed shape information. Therefore, generalized cylinders do not provide the most useful representation in terms of design and discrimination.



**Figure 1.4:** Different modelled forms using fully, partial and free-form constrained Generalized Cylinders

Figure 1.4 shows shapes for a variety of different constraints for the creation of Generalized Cylinders [Vinayak et al., 2013].

#### 1.3.1.4 Superquadrics

Superquadrics can be classified as implicit functions for surface representation. A superquadric function is defined by a function  $S$  whose  $x, y$  and  $z$  components are specified as functions of the angles  $\alpha$  and  $\omega$ :

$$S(\alpha, \omega) = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a_1 \cos^{\epsilon_1}(\alpha) \cos^{\epsilon_2}(\omega) \\ a_2 \cos^{\epsilon_1}(\alpha) \sin^{\epsilon_2}(\omega) \\ a_3 \sin^{\epsilon_1}(\alpha) \end{bmatrix} \quad (1.3)$$

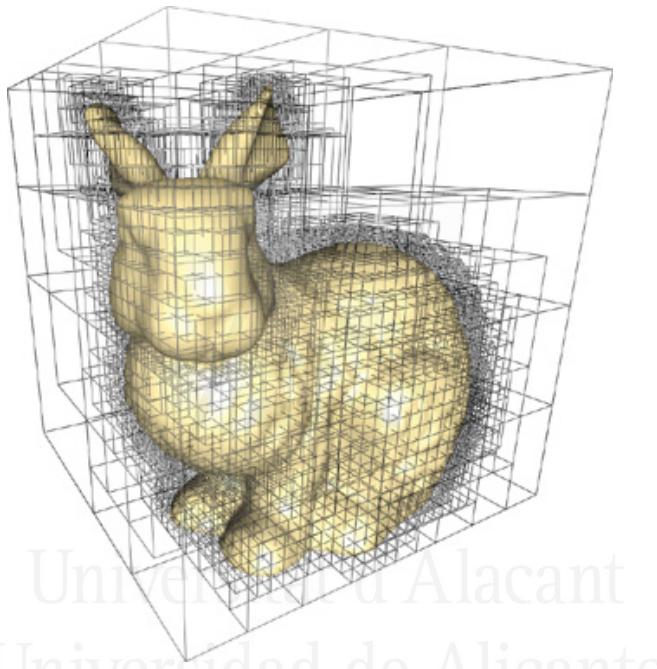
where  $-\frac{\pi}{2} \leq \alpha \leq \frac{\pi}{2}$  and  $-\pi \leq \omega \leq \pi$ .  $a_1, a_2$  and  $a_3$  define the size of the superquadric in the  $x, y$  and  $z$  axes. Parameters  $\epsilon_1$  and  $\epsilon_2$  represent the squareness of the shape. Superquadrics can be used as lumps of clay that can be tapered, twisted and bended in order to obtain the desired model.

Although superquadrics are models originally developed for computer graphics, they were proposed and used in computer vision originally by [Pentland, 1986] and later by other authors [Leonardis et al., 1997]. Despite a variety of free-forms can be represented using superquadrics, this model is not able to capture fine details that allow to differentiate natural observations like faces or other body parts. However, the ability of superquadrics to model coarse shapes has been used to determine geometric classes of objects [Raja and Jain, 1992].

#### 1.3.1.5 Octree-based representation

From computer graphics we find another representation model based on the tree data structure. The Octree is the most relevant within this kind of models. An octree is a hierarchical structure that subdivides the space in eight nodes of the same length. Each node in the tree corresponds to a cubic region of the universe. Nodes are classified as empty, full or partial if the cube partly intersects an object. A node with label partial has eight children representing the partition of the cube into octants. This step is performed recursively until all nodes are empty or full. In this

way a three dimensional scene can be represented by a  $2^n \times 2^n \times 2^n$  three dimensional array. This structure has the ability to considerably compress the information of the scene. For example, in [Schnabel and Klein, 2006] it is presented a progressive compression method that is specifically useful to deal with densely sampled surface geometry. The proposed method is suitable for streaming applications, e.g. on the Internet.



**Figure 1.5:** Stanford Bunny model representation using the Octree model.

Figure 1.5 shows the Stanford Bunny model represented using the octree method. This figure is courtesy from NVIDIA, it was originally presented in [Pharr and Fernando, 2005]. This compact and compressed form has not been considered to be used in computer vision because it is not a complete form and it does not present morphing capabilities to represent dynamic models.

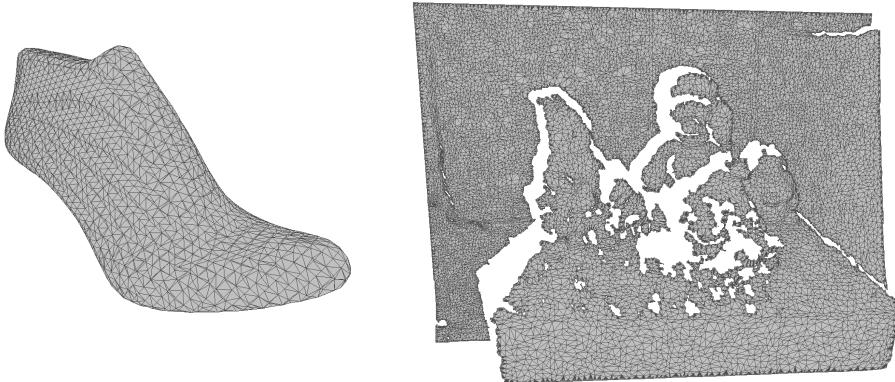
#### 1.3.1.6 Polygon mesh

The 3D mesh structure is one of the most popular approaches to represent objects and scenes. This is a simple geometric-based model which is

comprised of a set of vertices, edges and faces that together form a set of polygons. Meshes can be comprised of different kind of polygons. If polygons are of the same type meshes are referred as regular and one of the most common is the triangular mesh. 3D meshes represent the input space at different levels of detail and they are used both for graphics rendering or for computer vision applications. Furthermore, this model will be extended and used as a basis model in other approaches. Another important fact about representation models, that will be discussed later in this document, is the way how the representation model is constructed and the ability of the algorithm to adapt it to changes and variations over the time in the observation. Although polygonal meshes have been traditionally used in computer graphics, last years it has also become really active as object representation model in computer vision [Mian et al., 2006a, Johnson and Hebert, 1999, Chen and Bhanu, 2007, Wang et al., 2000]. This increase of popularity is caused by different factors as advances in computing power and storage, and also due to the advent and increase of popularity of dense range sensors, which provide depth maps that can be easily triangulated using this model. Polygonal meshes have the ability to easily represent free-form observations with different accuracy. The only constraint is the amount of computing time and the storage capacity needed when the accuracy of the final model is increased.

Derived from the 3D mesh model previously introduced there is a simplified model which is only comprised of vertices and edges. This is also widely used and commonly known in graphics as wire-frame representation. This model does not contain information about the surface of the input space and therefore only relationship information between vertices is stored.

Figure 1.6 shows two different three-dimensional meshes used in computer vision problems. On the left side, the polygonal mesh has been generated performing registration of multiple scans with different views of a shoe last. Finally, a complete model is constructed and the polygonal mesh is used as a representation method for future tasks. These tasks range from modifying original shape applying transformations and generating a new shoe last or using this polygonal mesh as an input for CAM



**Figure 1.6:** (Left) 3D mesh representing a complete shoe last model, it contains 3,500 vertices and 7,000 faces. (Right) 3D mesh obtained from a single view of a scene. It is comprised of 21,941 vertices and 33,470 faces.)

purposes in order to create a physical shoe last from this virtual model. On the right side a 3D mesh of a scene is shown. This polygonal mesh is employed as an input for many scene understanding and object recognition applications. Several descriptors are calculated over these polygons enabling the matching against a library of models previously computed.

In this section we have reviewed the state-of-the-art in representation methods but we can also find other relevant reviews in [Campbell and Flynn, 2001, Stockman and Shapiro, 2001, Söderkvist, 1999].

Finally, from the study performed above we can extract that the polygon mesh may be a suitable representation model for our problem. It meets most of the constraints that we presented in Section 1.2 and therefore it can be used in most of the presented computer vision problems. Besides the representation model, another problem arises: how this polygon mesh is constructed fitting the physical surface of the observation. In this thesis, we will focus on three-dimensional observations due to most of the tackled problems use 3D cameras or sensors as input devices. Despite different representation models can be fitted by different methods, in this work we will focus on fitting algorithms for triangular meshes.

### 1.3.2 Surface fitting methods

This section reviews the most used methods and techniques for extracting three dimensional surfaces from 3D input data acquired using different sensors. This section will focus on methods that generate three-dimensional polygons or wire meshes. Before analysing different methods it is necessary to consider some attributes of these methods: accuracy, conciseness, robustness and the level of automation. In order to fit an accurate and concise surface representation we need a measure of these properties. The conciseness can be simply described as the ratio between the number of bytes that is needed to store the representation of the fitted surface  $S$  by the accuracy of the fitted surface. For a triangular mesh this is directly proportional to the number of triangles that are used. This attribute does not only affect to the memory space that is saved, but it also affects to post-processing steps. It permits faster post-processing steps over the fitted surface: keypoint detection, feature extraction, rendering, computation of intersections, etc.

The second attribute that should be considered is the accuracy. The accuracy of the fitted surface determines how close is the fitted surface  $S$  from the physical surface  $P'$ . The physical surface is defined only by a set of noisy samples  $P' = \{p'_i\}$ ,  $p'_i = (p'_x, p'_y, p'_z)$ , therefore the accuracy is measured by using some assumptions about the noise present in the captured observation. This indicates that the method should not strive for getting an accuracy level that is better than the noise level present in the samples. This is also known as over-fitting and it occurs when the surface  $S$  approximates the sampled point including the present noise.

Robustness is one of the most important properties of reconstruction methods. This attribute means that the algorithm should be aware of missing samples and erroneous samples, borders, etc.

Finally, the level of automation is also quite important as it is more convenient that no additional parameters for the method are required, just the input points.

Moreover, the wide range of applications from which the data may emerge implies that data can have quite different properties and information. In most works no a priori information is given and there is no

structural knowledge available about the sampled points relationship.

Surface reconstruction algorithms can be split into different classes or categories according to some attributes, these classes are not necessarily disjunct.

They can belong to the category of *approximated* or *interpolated* algorithms, considering if the reconstructed surface preserves original data points (*interpolation*) or otherwise the reconstructed surface *approximates* them.

Moreover, methods can be classified as global or local considering if the final reconstructed surface is influenced only by some local points (neighbours) or otherwise all points are used to define the entire surface.

Finally, in the survey on reconstruction surface methods presented in [Mencl and Müller, 1997] four basic classes or categories are established: surface reconstruction methods based on spatial subdivision, based on distance function, on warping and on incremental surface growing. We considered these four common categories to present state-of-the-art methods in 3D surface reconstruction techniques as they cover most approaches that have been developed in this research topic during the years.

### 1.3.2.1 Spatial subdivision methods

The goal of surface reconstruction algorithms based on spatial subdivision is to find cells related to the shape described by the sampled points. Typical examples are regular grids, adaptive schemes like octrees, or irregular schemes like tetrahedral meshes. These methods have in common the next steps:

1. Space decomposition in cells
2. Finding those cells traversed by the surface
3. Computing the surface from selected cells

In [Algorri and Schmitt, 1996] an example of spatial subdivision based surface reconstruction algorithm is presented. In the first step, the space is subdivided in voxels using a regular voxel grid [Connolly, 1984, Kobbelt

and Botsch, 2004] and in the second step, the algorithm extracts those voxels that are at least occupied by one point. Finally, the outer voxels are taken as a first approximation of the surface. The obtained mesh can be refined and presented as a triangular mesh splitting each quadrilateral into two triangles.

Another common step in many surface reconstruction algorithms is the tetrahedrization of the input space using Delaunay methods. The reconstruction with Delaunay methods in three dimensions consists of the extraction of tetrahedron surfaces from the initial point cloud. This step is performed in many surface reconstruction methods to obtain a first approximation. Moreover, this step is used in different categories as spatial subdivision and distance function based methods.

[Edelsbrunner and Mücke, 1994] used an irregular spatial decomposition instead of a regular one. The chosen decomposition is the mentioned Delaunay tetrahedrization mentioned above. After this decomposition, it is introduced the concept of  $\alpha$ -shape, which is used after the decomposition for removing tetrahedra, triangles and edges from the first approach. The concept of alpha-shape formalizes the intuitive notion of "shape" for a set of points in the space. One of the earliest approaches is based on  $\alpha$ -shapes. Given a finite point set  $S$ , and the real parameter  $\alpha$ , the alpha-shape of  $S$  is a polytope (the generalization to any dimension of a two dimensional polygon and a three-dimensional polyhedron) which is neither convex nor necessarily connected. For a large  $\alpha$  value, the  $\alpha$ -shape is identical to the convex-hull [O'Rourke, 1998] of  $S$ . If the alpha value decreases progressively non-convex shapes with cavities are obtained. The algorithm proposed by [Edelsbrunner and Mücke, 1994] eliminates all tetrahedrons which are delimited by a surrounding sphere smaller than  $\alpha$ . The surface is then obtained with the external triangles from the resulting tetrahedron. Another approach is based on labelling the initial tetrahedrons as interior and exterior. The resulting surface is generated from the triangles found in and out. This idea first appeared in [Boissonnat, 1984] and was later performed by Powercrust in [Amenta et al., 2001] and the algorithm called Tight Cocone [Dey and Goswami, 2003]. Both methods have been recently extended to reconstruct point clouds with noise [Dey and Goswami,

2003, Mederos et al., 2005].

The main advantage of most Delaunay based methods is that they fit accurately the surface defined by the original point cloud. However, these methods are very sensitive to noise producing undesirable results with noisy data. Therefore, the quality of the points obtained from used 3D sensors determines the feasibility of these methods. Due to the use of the whole point cloud set to obtain the most accurate triangulation, considering the Delaunay rule, captured points with an error considered above the limit, will be explicitly represented on the reconstructed surface geometry.

### 1.3.2.2 Distance functions methods

The distance function of a surface gives the shortest distance of any point in the input space to the approximated surface. This distance can be positive or negative for closed surfaces, considering if the point lies inside or outside the volume bounded by the surface.

For example, implicit reconstruction methods (or zero-set methods) reconstruct the surface using a distance function which assigns to each sampled point a signed distance to the surface  $S$ . The polygonal representation of the object is obtained by extracting a zero-set using a contour algorithm. Thus, the problem of reconstructing a surface from a disorganized point cloud is reduced to the definition of the appropriate function  $f$  with a zero value for the sampled points and different to zero value for the rest. [Lorensen and Cline, 1987] established the use of such methods with the algorithm called Marching-Cubes. This algorithm has evolved in different variants: [Hoppe et al., 1992] used a discrete function  $f$ , and in [Carr et al., 2001] a polyharmonic radial basis function is used to adjust the initial point set. Other approaches include the Moving Least Squares adjustment function [Shen et al., 2004, Fleishman et al., 2005] and basic functions with local support [Walder et al., 2006], based on the Poisson equation [Kazhdan et al., 2006]. Moreover, in [Roth and Wibowoo, 1997] distance values are calculated at the vertices of a given voxel grid surrounding the data points. The points are set to the cell they belong and an outer normal vector is calculated for each point by using its neighbouring points

in the voxel grid. The value of the signed distance function is computed by the weighted average of the signed distances for every point in the eight neighbourhood of the voxel.

All these methods have the problem of loss of the geometry precision in areas with extreme curvature, i.e., corners, edges. Furthermore, data preprocessing by applying some kind of filtering technique also affects the definition of the corners, softening them. There are several studies related to post-processing techniques used in the reconstruction for the detection and refinement of corners [Fleishman et al., 2005, Wang, 2006] but these methods increase the complexity of the solution.

### 1.3.2.3 Warping methods

Warping-based methods deform an initial surface approximating the given point set. For example, given a initial shape, vertices are moved towards the point set in the warping process. When moving the vertices to their new location, the rest of the mesh is warped achieving a surface approximation of the physical surface  $P'$ .

[Muraki, 1991] suggested a blobby model that approximates 2.5D range images. [Miller et al., 1991] extracts a topologically closed geometric model from a volume data set. The algorithm starts with a simple model that is already topologically closed and it is deformed until a set of constraints are fulfilled, so the model grows or shrinks to fit the input space. Two different approaches within warping methods can be found: geometric and computational intelligence based.

Purely geometric approaches are based on the idea of the space deformation by a finite set of displacement vectors consisting on a pair of initial and target points. Spatial displacement vectors are interpolated using scattered data interpolation methods. In particular, if the displacement vector field is applied to all vertices of the initial mesh, or in a refined one, the mesh is warped toward the sampled points [Ruprecht et al., 1995].

Some relevant works directly related to Computational Intelligence (CI) approaches are based on Self-Organizing Maps (SOM) proposed originally by [Kohonen, 1995]. In [Baader and Hirzinger, 1994, Baader and Hirzinger, 1993] this CI algorithm is used as a surface reconstruction method. In this

case, the map is deformed to fit the input space. During the reconstruction or training process the neurons or nodes of the map are fed with the input data which affects their position (weight vectors). This approach presents some drawbacks as the original SOM algorithm does not allow to remove edges or connections between neurons, so the adaptation to the input space may not fit accurately. In addition, this approach is not able to deal with gaps and the presence of multiple shapes in the input space.

#### 1.3.2.4 Incremental reconstruction methods

The main goal of incremental surface reconstruction methods is to interpolate or approximate the surface directly using properties of the given data points. For example, surface reconstruction starts with an initial surface edge at two neighbouring points. This edge is successively extended to a larger surface area by iteratively creating new edges and triangles at boundary edges of the emerging surface.

[Boissonnat, 1984] surface algorithm starts at the shortest connection between two points of the given point set and in order to attach a new triangle at this edge, a local estimated tangent plane is computed based on the neighbour points of the boundary edge. The points in the neighbourhood of the boundary edge are projected onto that plane. The new triangle is obtained by connecting one of these points to the boundary edge. The point that maximizes the angle between its edges is selected. The presented method is quite simple and effective for some data distributions but it does not consider gaps and holes in the input data and also low density point sets.

Furthermore, based on Self-Organizing Maps (SOMs) several growing approaches have been successfully used to reduce the dimensionality of 3D input data maintaining a good topology preservation [do Rego et al., 2007], [Fanany and Kumazawa, 2004] and [Holdstein and Fischer, 2008]. As 3D reconstruction can be considered as a cluster-seeking problem in which the goal is finding a finite number of points that describe the surface precisely. The structure of the nodes (locations) should be found automatically by minimizing the error of the Growing Self-Organizing Map (GSOM) adaptation. Identifying the points of the sampled data that belong to objects

allows the map to adapt its structure to this input subspace, obtaining an induced Delaunay triangulation of the input space. This approach allows a more compact representation and straightforward geometric morphing between successive time representations of the input space. An incremental fitting is performed over the input space, inserting new neurons in the input space area that is needed (largest error).

## 1.4 Proposal

After describing the motivation of this work and analysing the state-of-the-art in representation models that nowadays exist or have been proposed, we have identified an almost unexplored gap in the topic of applying representation models and surface fitting techniques for computer vision problems. As we have described in Section 1.3.1, most representation models and surface reconstruction techniques are focused on computer graphics (rendering, computer games, visualization applications, etc.). Moreover, the few works that have been published in this novel research topic are focused on computer vision problems that take as an input two-dimensional data distributions, leaving unexplored the use of these techniques in three-dimensional data and in particular in captured data using low cost sensors which have proliferated in the last years. These new sensors present advantages and disadvantages, as we will study along this work: an important disadvantage is the presence of noise and an interesting advantage is the frame rate that they offer, reaching up to 25 acquisitions per second in range and time-of-flight cameras.

Consequently, based on our experience using neural gases [García-Rodríguez, 2009, Angelopoulou et al., 2005], which have been successfully applied to the representation of 2D shapes in many computer vision problems [Stergiopoulou and Papamarkos, 2006, Doucette et al., 2001, García-Rodríguez et al., 2010, Baena et al., 2013], we propose to extend these models to deal with 3D noisy acquisitions, to operate under time constraints and to manage the processing of sequences of 3D data. Neural Gases also known as growing self-organising maps are inherently parallel, due to this attribute we propose a GPU implementation of the algorithm

using GPGPU techniques. We expect a considerable speed-up from this GPU implementation, regarding to naive and optimized CPU implementations. Parallel computation and neural networks are new computing paradigms with increasing attention among computing applications.

Furthermore, thanks to the good topological properties of the map generated by this algorithm, we propose the use of this method to acquire object models in CAD problems and also for scene reconstruction, being able to deal with the presence of noise caused by common 3D sensors. In addition, as the representation model is based on polygonal meshes, we will use this structure for a posteriori keypoint detection and feature extraction steps. Finally, we propose to use our technique to solve challenging computer vision problems.

## 1.5 Goals

The main goal of this research is the proposal and validation of a representation model which will be able to help solving prevailing 3D computer vision problems as those already presented in Section 1.2. The proposed representation model will operate under time constraints and will be flexible to represent not only static observations but also able to deal with dynamic changes in the observation. In this way, the selected representation model will be used as the input for 3D keypoint detection and feature extraction algorithms. Furthermore, with the advent of many different new low cost 3D sensors, the proposed model will be able to operate directly over these streams of data that usually contain noise and outliers.

An accelerated hardware implementation of the proposed technique will be developed in order to achieve a considerable speed-up regard CPU implementations and real-time processing rates.

As a secondary goal but directly related to the first one, the proposed model will be tested over different real computer vision problems, helping to solve them in a successful way. Cases of study will range from the use of computer vision in robotics to CAD and visual surveillance.

## 1.6 Structure of the dissertation

The Phd dissertation has the following structure:

Chapter 2 contains a detailed study of self-organising maps applied to 3D representation. Different self-organising maps are deeply studied and compared, showing advantages and disadvantages for several kind of data. Moreover, noise removal properties and other effects as filtering and downsampling are further studied. Growing Neural Gas (GNG) [Fritzke, 1995] is selected as one of the most convenient techniques to process 3D real-world data, performing experiments on 3D scene reconstruction and 3D object acquisition. Moreover, we extended the GNG method to deal with sequences of data captured from real-time 3D sensors, redesigning the original algorithm and improving its learning process.

Chapter 3 revises the state-of-the art in 3D keypoint detectors and 3D feature descriptors focusing on those that have low runtime and are also accelerated using specific hardware as GPUs. In particular, we study and analyse one descriptor based on surface patches for its later acceleration onto GPUs. Described techniques are tested using multiple view registration problem, performing 3D feature matching between different views of a scene. The proposed representation model described in Chapter 2 will be used as an input data in the detection of high interest points and in the feature extraction step.

Chapter 4 presents a general overview of how GPUs, pointing out how they have democratized High Performance Computing (HPC) in last years. Furthermore, it presents an overview about the new architecture and the programming model of General Purpose computing on Graphics Processing Units (GPGPU).

Second part of this chapter presents GPGPU implementations of Growing Neural Gas and Neural Gas algorithms. It discusses strategies to fit these algorithms on the GPU architecture. In addition, it compares them with single thread and multiple thread implementations on the CPU. Finally, this chapter presents an accelerated GPU-based implementation of a surface patch descriptor. In particular, presented GPGPU implementations are focused on the CUDA architecture from NVIDIA.

Chapter 5 presents different cases of study. Cases of study are grouped within various categories where the proposed method has been applied: Robotics, Computer Vision and Computer Aided Design (CAD).

Finally, Chapter 6 details the conclusions extracted from the present work. Moreover, it presents the contributions to the topic and briefly describes derived publications from this work. To conclude the chapter, future directions of the research carried out are drawn.



Universitat d'Alacant  
Universidad de Alicante

---

## Chapter 2

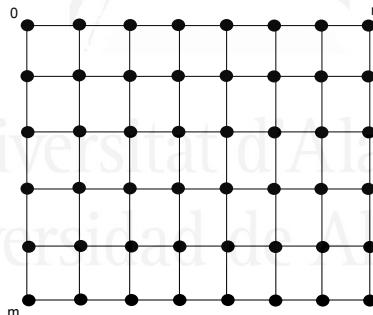
# 3D Representation with Growing Self-organizing Networks

---

This chapter demonstrates the capacity of Growing Self-Organizing Maps (GSOM) to represent 3D data distributions. In Section 2.2 different GSOM approaches have been deeply studied and compared, showing advantages and disadvantages for 3D data captured from noisy sensors. Section 2.3 shows how Growing Self-organizing Maps are capable to perform filtering and down-sampling processing steps over 3D data, obtaining a reduced representation with better adaptation to the physical surface  $P$ . Growing Neural Gas and Neural Gas are selected as robust techniques for 3D real-world data representation, performing experiments on 3D scene reconstruction and 3D object acquisition. Furthermore, in Section 2.3.5 and 2.3.7 some improvements are made on the original algorithm considering colour information during the learning step and producing complete polygon models instead of basic wire-frame representations. Finally, in Section 2.3.6 the GNG method is extended to deal with sequences of data captured from real-time 3D sensors, modifying the original algorithm and improving its learning process to manage point cloud sequences.

## 2.1 Introduction

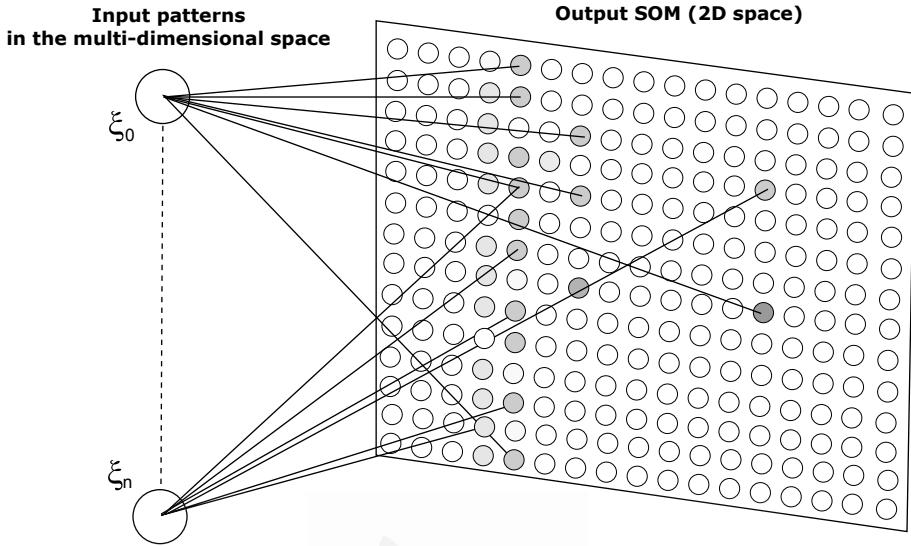
Self-organizing Maps (SOMs) are a special class of Artificial Neural Network (ANN) also known as Self-organizing Feature Maps. The model was originally presented by Teuvo Kohonen, and is sometimes called a Kohonen map or network [Kohonen, 1982]. These networks are based on competitive learning, which means that the output neurons of the network compete among themselves to be activated, with the result that only one output neuron is activated at one time. In this way, neurons become selectively tuned to various input patterns, also known as *stimuli*, performing the learning process. Moreover, in SOMs, the neurons are usually placed at the nodes of a  $n \times m$  lattice (Figure 2.1), which will be arranged during the learning process in order to represent intrinsic statistical features of the input space. Since SOMs are inherently non-linear, they can be viewed as a non-linear generalization of Principal Component Analysis (PCA). SOM structure can be seen in Figure 2.2.



**Figure 2.1:** Rectangular lattice commonly used by the Kohonen's map as initial topology.

SOMs were originally proposed for data clustering and pattern recognition purposes [Vesanto and Alhoniemi, 2000, Flexer, 2001, Dittenbach et al., 2001]. They also were used for topology learning and visualization of multi-dimensional data distributions [Delgado et al., 2009].

As we have mentioned, SOMs are trained to produce a low-dimensional, discretized representation of the input space. The map is trained using Unsupervised Learning (UL), which is also known as data clustering and defined as the problem of finding homogeneous groups of data points in a



**Figure 2.2:** The Self-Organizing Map structure. Selection of a node and adaptation of neighbouring nodes of the neural network to the input data.

given multidimensional data set. Each of these groups is called a cluster and defined as a region where the density of data points is locally higher than in others regions. Another objective of unsupervised learning can be described as topology learning: given a high-dimensional data distribution finds a topological structure that closely reflects the topology of the data distribution [Furao and Hasegawa, 2006]. In this work we focused on this last capability.

Self-organizing models [Kohonen, 1982] place their neurons so that their positions within the network and connectivity between different neurons are optimized to match the spatial distribution of activations. As a result of this optimization, existing relationships within the input space will be reproduced as spatial relationships among activated neurons. The final result of the self-organizing or competitive learning process is closely related to the concept of Delaunay triangulation of the input space corresponding to the reference vectors of neurons in the network. Traditionally, it has been suggested that this triangulation, result of competitive learning, preserves the topology of the input space. However, [Martinetz and Schulten, 1994] introduces a new condition which restricts this quality. Thus, self-

organizing maps or Kohonen maps are not Topology Preserving Networks (TPN) as they have traditionally been considered, since this condition only would happen in the case that the topology or dimension of the map and the input space match. Other approaches based on Kohonen maps as the Growing Cell Structures [Fritzke, 1993] are also not TPN since the topology of the network is established a priori.

As the original model presented by [Kohonen, 1982] had some drawbacks due to the pre-established topology of the network, growing approaches were proposed in order to deal with this problem. Growing models have been widely used in recent years in many applications due to their attributes of growth, flexibility, rapid adaptation, and excellent quality of representation of the input space. These features convert the neural networks in a suitable model to solve problems that deal with non-stationary input data. Many related works have been published in last years on computer vision and man-machine interaction issues like image compression [García-Rodríguez et al., 2007], segmentation and representation of objects [Florez-Revuelta et al., 2002, Rivera-Rovelo et al., 2006, Wu et al., 2000, Palomo et al., 2013], objects tracking [Angelopoulou et al., 2007, Cao and Suganthan, 2002, Frezza-Buet, 2008, Baena et al., 2013], gestures recognition [Florez et al., 2002, García-Rodríguez et al., 2006a, Angelopoulou et al., 2013], 3D object reconstruction [Cretu et al., 2008, do Rego et al., 2007, Holdstein and Fischer, 2008] and 3D localization and mapping [Viejo et al., 2012b, Viejo et al., 2012a]

In the case of the Neural Gases like Neural Gas (NG) [Martinetz et al., 1993], or Growing Neural Gas (GNG) [Fritzke, 1995], the mechanism for adjusting the network through a competitive learning generates an induced Delaunay triangulation. [Martinetz and Schulten, 1994] demonstrates that these models are TPN. In particular, the Growing Neural Gas [Fritzke, 1995] is an incremental model able to learn the important topological relations in a given set of input vectors by means of a simple Competitive Hebbian Learning (CHL) [Martinetz and Schulten, 1994] rule.

## 2.2 Self-Organizing Maps applied to 3D representation

Many well established techniques proposed solutions to the 3D representation and surface reconstruction problem from a geometric point of view. However, these algorithms required long times to process the input point cloud and do not scale properly with very large data [Hoppe et al., 1992, Amenta et al., 2001]. Moreover, these traditional geometric approaches do not manage non-stationary distributions and do not deal with the lack of a priori information about the input space, e.g. the presence of multiple shapes in the point cloud and noise induced by the sensors.

### 2.2.1 Related works

Considering the 3D representation problem from a machine learning approach and based on self-organization maps, a different perspective to obtain 3D reconstructions is proposed. As it was introduced in Chapter 1, these methods could be classified as flexible and growing models considering if the topology of the selected map has a priori topology or otherwise it grows until a condition is fulfilled. Moreover, we can find some similarities or correspondences between the trained map and the 3D representation that we want obtain. Nodes of the trained map correspond to vertices of a mesh and connection between nodes correspond to the edges. Therefore, in this work the terms node and vertex, and connection and edge are used interchangeably. From this perspective, some methods were proposed based on self-organizing maps.

[Yu, 1999] proposed the use of Kohonen's self-organizing map for surface reconstruction using as an input data unorganized point clouds. Moreover, since Kohonen's map does not produce regular faces, an edge collapse operation is introduced eliminating dangling faces. This approach presented some drawbacks as if the real object surface had concave structures, applying Kohonen's learning algorithm has some difficulties to correctly approximate those parts. In addition, as the Kohonen's algorithm has a high computational cost, the single thread CPU implementation presented

in this work took more than one hour to represent the Stanford bunny model. Presented method was only tested with synthetic data and the bunny model, which is comprised of 34,834 points. [Junior et al., 2004] extended [Yu, 1999] introducing new mesh operators that allowed it to perform improvements on the surface geometry: edge swap, edge collapse, vertex split and triangle subdivision. Moreover, the method introduced a new step to remove unstable vertices using the mean distance and the standard deviation of the 3D representation regarding the sampled input space. Although this new approach improved the surface geometry, the method did not deal with concave or non-convex regions and the initial structure of the representation had to be pre-established considering the topology of the input space. The fixed structure of the SOM does not learn the spatial relationships between the vertices and therefore does not generate a model that accurately represents the shape of the input space. To overcome this problem, some methods based on Growing SOMs were proposed. These methods are based on different learning schemes but all of them had in common a flexible topology.

One of these SOM-based methods is the Growing Cell Structures (GCS) algorithm [Fritzke, 1993], which is a model that is formed incrementally. However, it constraints the connections between the nodes, so any model produced during training is always topologically equivalent to the initial topology. [Ivrissimtzis et al., 2003] used the GCS algorithm to reconstruct object surface. This growing cell structure has the ability to generate meshes with different resolutions at different stages of the learning process. Meshes operators are used to change the connectivity of the mesh and therefore final topology is always equivalent to the initial mesh.

The Topology Representing Networks (TRN), proposed by [Martinetz and Schulten, 1994], does not have a fixed structure and also does not impose any constraint about the connection between the nodes. In contrast, this network has a pre-established number of nodes, and therefore, it is not able to generate models with different resolutions. The algorithm was also coined with the term Neural Gas (NG) due to the dynamics of the feature vectors during the adaptation process, which distributes themselves like a gas within the data space. [Barhak, 2002] proposed a NG-based surface

reconstruction algorithm since this network has the ability to accurately represent the topology of a point cloud. However, as the NG has a fixed number of nodes, it is necessary to have some a priori information about the input space to pre-establish the size of the network. This model was extended by [Fritzke, 1995] proposing the Growing Neural Gas (GNG) network, which combined the flexible structure of the NG with a growing strategy. Moreover, the learning adaptation step was slightly modified. This extension enabled the neural network to use already detected topological information while training in order to conform to the geometry. This approach has the capability to add neurons while preserving the detected topology. As the original GNG algorithm does not produce faces and the generated map is a wire-frame representation model, some works extended the original algorithm to produce faces. In [Cretu et al., 2008] the GNG network is employed to model the resulting point cloud and those regions that need further sampling in order to obtain a more accurate model. Rescanning at higher resolution is performed for each identified region of interest and a multi-resolution model is built. In this work, only nodes of the generated map are used as the work is focused on sampling capabilities of the GNG. In [Holdstein and Fischer, 2008] the MGNG is proposed. MGNG applied some postprocessing steps in order to perform surface reconstruction once the map is generated using the original GNG algorithm. Moreover, MGNG is adaptive as it allows manual selection of regions of interest from the cloud of sampled points and re-feeding those regions to the MGNG algorithm. Most of these approaches are tested against CAD models or synthetic data and only few experiments were performed on objects acquired with range cameras. In [Do Rego et al., 2010], instead of adding post-processing steps for surface reconstruction, the GNG algorithm is modified in order to produce topological faces. The extended method was called Growing Self-Reconstruction Maps (GSRM) and some learning steps as CHL and the operation of vertex insertion and removal were modified. Most experiments of this work were performed on the Stanford dataset, which had been previously filtered and therefore the surface reconstruction step did not have to face noisy input spaces produced by common 3D sensors.

Finally, in [Carr et al., 2001, Liu et al., 2006], a different approach based on Radial Basis Functions (RBF) [Buhmann, 2003], which have been often used in Neural Networks field, was used to reconstruct arbitrary topology objects. These methods required as input parameter a signed distance function extracted from the point cloud.

Although the use of the SOM-based techniques as NG, GCS or GNG for 3D input space representation and surface reconstruction has already been studied and successful results have been reported, there are some limitations that still persist. The majority of these works do not consider the high computational cost of the learning step and therefore these approaches do not guarantee response within strict time constraints. In addition, most of these works assumed perfect point clouds that were noise-free. Therefore, applying these methods on challenging real-world data obtained using noisy 3D sensors have not been yet object of study. Moreover, with the advent of low cost RGB-D cameras as the Microsoft Kinect<sup>1</sup> partial point clouds have to be considered. Besides providing 3D information, these devices also provide colour information, feature that was not considered in the revised works. Finally, a new feature is demanded, the necessity to deal with sequences of point clouds instead of single point clouds.

Based on the analysis performed, in this thesis we propose the use of the Growing Neural Gas algorithm for 3D representation of objects and scenes captured using real-time 3D sensors. This algorithm has been selected as it has been demonstrated to be the best option due to its attributes of flexibility, rapid adaptation and quality of representation. Neural Gas method will be also considered as it presents slightly better topology representation capabilities [Florez-Revuelta, 2002], but its computational complexity does not allow it to be used on time-constrained point cloud processing applications. Other aspects that will be further studied are the noise removal properties of the generated map and the input space adaptation quality. As both models, GNG and NG, will be object of study of the whole thesis document, in the next subsections, original algorithms are depicted.

---

<sup>1</sup>Kinect for XBox 360: <http://www.xbox.com/kinect> Microsoft

## 2.2.2 Neural Gases

In this section, original algorithms of the Growing Neural Gas and Neural Gas methods are detailed. Although both algorithms are capable of managing input distributions of any dimension, in this work we are focused on 3D data distributions and therefore the dimension of the input distribution is  $d = 3$  and the code-vectors of the neurons also have dimension 3.

### 2.2.2.1 Neural Gas Algorithm

Neural Gas is an unsupervised soft competitive clustering algorithm that given some input distribution in  $\mathbb{R}^d$  creates a graph, or network of nodes, where each node in the graph has a position in  $\mathbb{R}^d$ . The model is used for vector quantization by finding the code-vectors in clusters. These code-vectors are represented by the reference vectors (the position) of the nodes. It can also be used for finding topological structures that closely reflects the structure of the input distribution. Neural Gas learning is a dynamic algorithm in the sense that if the input distribution slightly changes over time, it is able to adapt, moving the nodes to the new input space.

The network is specified as:

A set  $A$  of nodes (neurons). Each neuron  $c \in A$  has its associated reference vector  $w_c \in \mathbb{R}^d$ . The reference vectors can be regarded as positions in the input space of their corresponding neurons.

A set of edges (connections) between pairs of neurons. These connections are not weighted, and its purpose is to define the topological structure. An edge aging scheme is used to remove connections that are invalid due to the motion of neurons during the adaptation process.

Neural Gas uses parameters that decay exponentially according to time and the distance to the input pattern.

The Neural Gas with Competitive Hebbian Learning (CHL) algorithm is the following:

1. Initialize the set  $A$  to contain  $N$  units (or neurons)  $c_i$

$$A = \{c_1, c_2, \dots, c_N\} \quad (2.1)$$

with reference vectors  $W_{c_i} \in \mathbb{R}^d$  chosen randomly from an input data distribution  $p(\xi)$ .

Initialize the connection set  $C \subset A \times A$ , to the empty set:

$$C = \emptyset \quad (2.2)$$

Initialize the time parameter  $t$

$$t = 0 \quad (2.3)$$

2. Generate at random an input signal  $\xi$  according to  $p(\xi)$ , where  $\xi$  is the input signal being learned.
3. Order all elements of  $A$  according to their distance to  $\xi$ , i.e., find the sequence of indices  $(i_0, i_1, \dots, i_{N-1})$  such that  $w_{i_0}$  is the reference vector closest to  $\xi$ ,  $w_{i_1}$ , is the reference vector second-closest to  $\xi$  and  $w_{i_k}$ ,  $k = 0, \dots, N - 1$  is the reference vector such that  $k$  vectors  $w_j$  exists with  $\|\xi - w_j\| \leq \|\xi - w_k\|$ . We denoted with  $k_i(\xi, A)$  the number  $k$  associated with  $w_i$ .
4. Adapt the reference vectors according to

$$\Delta w_i = \epsilon(t) \cdot h_\lambda(k_i(\xi, A)) \cdot (\xi - w_i) \quad (2.4)$$

with the following time-dependencies

$$\lambda(t) = \lambda_i(\lambda_f/\lambda_i)^{t/t_{max}} \quad (2.5)$$

$$\epsilon(t) = \epsilon_i(\epsilon_f/\epsilon_i)^{t/t_{max}} \quad (2.6)$$

$$h_\lambda(k) = e^{-k/\lambda(t)} \quad (2.7)$$

where step size  $\epsilon \in [0, 1]$  describes the overall extent of the modification, and  $h_\lambda(k_i(\xi, A))$  is equals to one for  $k_i = 0$  and decays to zero for increasing  $k_i$  with a characteristic decay constant  $\lambda$ . In this thesis we chose  $h_\lambda(k_i(\xi, A)) = e^{-k_i/\lambda(t)}$ .

5. If it does not exist already, create a connection between  $i_0$  and  $i_1$ :

$$C = C \cup \{(i_0, i_1)\} \quad (2.8)$$

Set the age of the connection between  $i_0$  and  $i_1$  to zero (“refresh” the edge):

$$\text{age}(i_0, i_1) = 0 \quad (2.9)$$

6. Increment the age of all edges emanating from  $i_0$

$$\text{age}_{(i_0, i)} = \text{age}_{(i_0, i)} + 1 \quad \forall i \in N_{i_0} \quad (2.10)$$

Thereby,  $N_c$  is the set of direct topological neighbors of  $c$ .

7. Remove edges with an age larger than the maximal age  $T(t)$  whereby

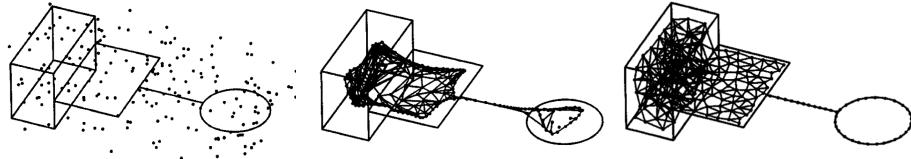
$$T(t) = T_f (T_f/T_i)^{t/t_{max}} \quad (2.11)$$

8. Increase the time parameter  $t$ :

$$t = t + 1 \quad (2.12)$$

9. If  $t \leq t_{max}$  continue with step 2

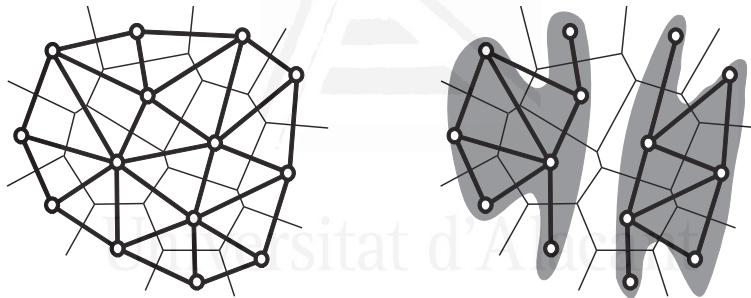
For the time-dependent parameters suitable initial values  $(\lambda_i, \epsilon_i, T_i)$  and final values  $(\lambda_f, \epsilon_f, T_f)$  have to be chosen. Following [Martinetz et al., 1993], we defined for the experiments carried out in this thesis the following parameters:  $\lambda_i = 10, \lambda_f = 0.01, \epsilon_i = 0.5, \epsilon_f = 0.005, t_{max} = 40,000, T_i = 20, T_f = 200$ .



**Figure 2.3:** Initial, intermediate and final states of the NG learning algorithm

Figure 2.3 shows the initial, intermediate and final stages of the NG learning algorithm and how the network adapts to the topology of the input space, starting from a pre-established number of neurons. Neurons are adapted over the time, fitting to the topology of the input space.

The neighborhood information is maintained throughout the execution by a variant of competitive Hebbian learning (CHL).



**Figure 2.4:** (a) Delaunay triangulation, (b) Induced Delaunay triangulation

The graph generated by CHL is called the “induced Delaunay triangulation” (Figure 2.4(a)) and is a sub-graph of the Delaunay triangulation (Figure 2.4(b)) corresponding to the set of nodes. The induced Delaunay triangulation optimally preserves topology in a very general sense [Martinetz et al., 1993]. CHL is an essential component of the NG algorithm since it is used to conduct the local adaptation of nodes.

### 2.2.2.2 Growing Neural Gas Algorithm

With Growing Neural Gas (GNG) [Fritzke, 1995] method a growth process takes place from minimal network size and new units are inserted successively using a particular type of vector quantization. To determine

where to insert new units, local error measures are gathered during the adaptation process and each new unit is inserted near the unit which has the highest accumulated error. At each adaptation step a connection between the winner and the second-nearest unit is created as dictated by the CHL algorithm. This is continued until an ending condition is fulfilled, as for example, the evaluation of the optimal network topology or reaching the insertion of a fixed number of neurons. The network is specified as:

- A set  $A$  of nodes (neurons). Each neuron  $c \in A$  has its associated reference vector  $w_c \in \mathbb{R}^d$ . The reference vectors are regarded as positions in the input space of their corresponding neurons.
- A set of edges (connections) between pairs of neurons. These connections are not weighted and its purpose is to define the topological structure. An edge aging scheme is used to remove connections that are invalid due to the motion of the neuron during the adaptation process.

The GNG learning algorithm to map the network to the input manifold is as follows:

1. Start with two neurons  $a$  and  $b$  at random positions  $w_a$  and  $w_b$  in  $\mathbb{R}^d$ .
2. Generate at random an input pattern  $\xi$  according to the data distribution  $P(\xi)$  of each input pattern.
3. Find the nearest neuron (winner neuron)  $s_1$  and the second nearest  $s_2$ .
4. Increase the age of all the edges emanating from  $s_1$ .
5. Add the squared distance between the input signal and the winner neuron to a counter error of  $s_1$  such as:

$$\Delta\text{error}(s_1) = \|w_{s_1} - \xi\|^2 \quad (2.13)$$

6. Move the winner neuron  $s_1$  and its topological neighbors (neurons connected to  $s_1$ ) towards  $\xi$  by a learning step  $\epsilon_w$  and  $\epsilon_n$ , respectively,

of the total distance:

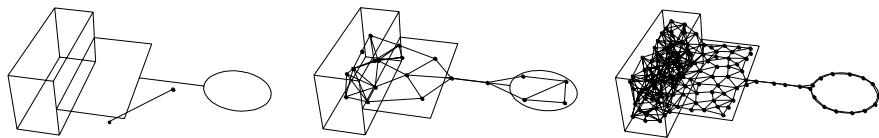
$$\Delta w_{s_1} = \epsilon_w(\xi - w_{s_1}) \quad (2.14)$$

$$\Delta w_{s_n} = \epsilon_n(\xi - w_{s_n}) \quad (2.15)$$

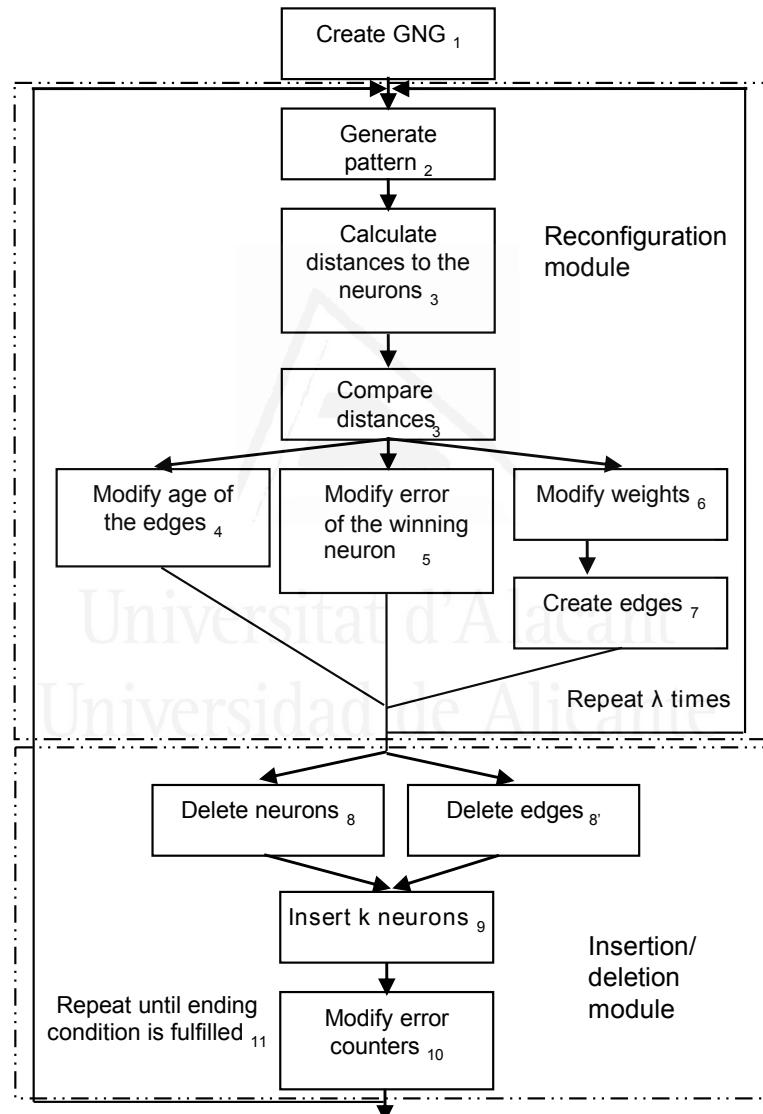
For all direct neighbors  $n$  of  $s_1$ .

7. If  $s_1$  and  $s_2$  are connected by an edge, set the age of this edge to 0. If it does not exist, create it.
  8. Remove the edges larger than  $a_{max}$ . If this results in isolated neurons (without emanating edges), remove them as well.
  9. Every certain number  $\lambda$  of input patterns generated, insert a new neuron as follows:
    - Determine the neuron  $q$  with the maximum accumulated error.
    - Insert a new neuron  $r$  between  $q$  and its further neighbor  $f$ :
- $$w_r = 0.5(w_q + w_f) \quad (2.16)$$
- Insert new edges connecting the neuron  $r$  with neurons  $q$  and  $f$ , removing the old edge between  $q$  and  $f$ .
  10. Decrease the error variables of neurons  $q$  and  $f$  multiplying them with a consistent  $\alpha$ . Initialize the error variable of  $r$  with the new value of the error variable of  $q$  and  $f$ .
  11. Decrease all error variables by multiplying them with a constant  $\gamma$ .
  12. If the stopping criterion is not yet achieved (in our case the stopping criterion is the number of neurons), go to step 2.

Figure 2.5 shows the initial, intermediate and final stages of the GNG learning algorithm. We can see how finally the network adapts to the topology of the input space, starting from two initial neurons in the map. A scheme of the GNG algorithm is presented in Figure 2.6



**Figure 2.5:** Initial, intermediate and final states of the GNG learning algorithm



**Figure 2.6:** GNG learning algorithm.

This method offers further benefits due to the incremental adaptation of the GNG, input space de-noising and filtering is performed in such a way that only concise properties of the point cloud are reflected in the output representation.

## 2.3 GNG-based method to represent real-world noisy observations

Since most works using Self-Organizing Maps and Neural Gases are focused on the representation of stationary observations which are usually post-processed and noise-free, we proposed a GNG-based method for the representation of real-world observations captured from noisy sensors. We studied the noise removal properties of the GNG with several 3D data distributions as objects and scenes. As modern 3D sensors also provide colour information, the method was modified to consider it in the map produced by the network. Moreover, as nowadays most sensors provide real-time frame rates and most computer vision applications are time-constrained, we extended the original implementation to consider the adaptation of the network to time-varying observations. Finally, some modifications were introduced to produce a complete polygon mesh representation.

### 2.3.1 Data acquisition

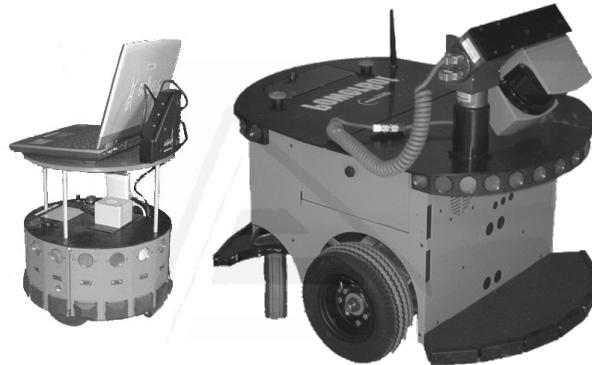
One of the goals of our work is to create an algorithm independent of the data source, i.e. to be applied to any 3D measurement device and can be used in indoor and outdoor environments. In this section we present the physical systems are used in our experiments.

We managed 3D data that can come from different sensor devices. For outdoor environments we used a laser unit, a LMS-200 Sick<sup>2</sup> mounted on a sweeping unit. It does not suffer the lack of texture and its range is 80 metres with an error of 1 millimetre per metre. The main disadvantage of this unit is the data capturing time: it takes more than one minute in one shot. This device provides us with 3D information in the form

---

<sup>2</sup>LMS-200 Sick laser: [http://robots.mobilerobots.com/wiki/SICK\\_LMS-200\\_Laser\\_Rangefinder](http://robots.mobilerobots.com/wiki/SICK_LMS-200_Laser_Rangefinder)

of point coordinates XYZ. This laser was mounted on a PowerBot from ActiveMedia. It has a battery life of 5 h, which is necessary for long outdoors experiments. Furthermore, PowerBot can carry the 3D sweeping laser unit, which is very heavy. Figure 2.7 shows two platforms that have been used for mounting 3D acquisition devices. The left one is an indoor platform, a Magellan Pro from iRobot. It is used for indoor experiments, since its dimensions are reduced (diameter: 40 centimetres, height: 24 centimetres). For outdoors we have used a PowerBot from ActiveMedia. Other experiments were performed carrying the sensors by hand.



**Figure 2.7:** Mobile robots used for experiments. Left: Magellan Pro unit used for indoors. Right: PowerBot used for outdoors.

For indoor environments we used another two sensors. The first one is a SR4000 camera from Mesa Imaging, which is a Time-of-Flight camera<sup>3</sup>, based on infrared light. ToF cameras were developed as a technology that obtains range (distance) and amplitude maps by the use of a modulated light source. The main advantages with respect to other 3D devices are the possibility to acquire data at video frame rates and to obtain 3D point clouds without scanning and from just one point of view. Its range is limited to 5 or 10 metres, providing gray level colour. ToF cameras allow the generation of point clouds during real time acquisition. The accuracy of ToF cameras varies depending on the internal components and the characteristics of the observed scene, such as objects reflectivity and ambient lighting conditions.

---

<sup>3</sup>Time-of-Flight camera SR4000 <http://www.mesa-imaging.ch/prodview4k.php>

The basic principle of ToF cameras consists of an amplitude-modulated infrared light source and a sensor field that measures the intensity of backscattered infrared light. The infrared source is constantly emitting light that varies sinusoidally. Objects located at different distances are reached by different parts of the sinusoidal wave. The reflected light is then compared to the original one, calculating the phase shift, by means of measuring the intensity of the incoming light since the phase shift is proportional to the time of flight of the light reflected by a distant object. A detailed description of the time-of-flight principle can be found in [Gokturk et al., 2004].

The SR4000 camera generates point coordinates XYZ, amplitude data of the scene and a confidence map of the distance measurements. In particular, the confidence map is obtained using a combination of distance and amplitude measurements and their temporal variations: it represents the probability that the measurement of the distance in each pixel is correct, so it can be useful to select regions containing high quality measurements or to reject low quality ones. In our experiments the amplitude data has low contrast so they have been equalized.

The second device used is the Kinect<sup>4</sup> sensor, which has been a great advance in several areas as robotics, computer vision and CAD/CAM. It is composed of two sensors: an IR (infrared) projector and IR CMOS camera, and a RGB camera. IR sensors provide depth information. The IR projector sends out a fixed pattern of light and dark speckles. Depth is calculated by triangulation against a known pattern from the projector. The pattern is memorized at a known depth and then for each pixel, a correlation between the known pattern and current pattern is done, providing the current depth at this pixel. The RGB camera has a resolution of  $640 \times 480$  (307200 pixels). The advantage of this camera against the SR4000 one is that with the Kinect, the RGB information is obtained, while in the SR4000 the information comes only from a infrared camera, which does not return good results from the current feature detectors. The Kinect depth sensor range is: minimum 800 and maximum 4000 millimetres. In order to obtain 3D information in the form of point clouds

---

<sup>4</sup>Kinect for XBox 360: <http://www.xbox.com/kinect> Microsoft

we had to project disparity information obtained from the device to the three-dimensional space using the known geometry of the sensor.

The relationship between a disparity map provided by the Kinect sensor and a normalized disparity map is given by  $d = 1/8 \cdot (d_{off} - kd)$ , where  $d$  is the normalized disparity,  $kd$  is the disparity provided by the Kinect and  $d_{off}$  is a particular offset of a Kinect sensor. Calibration values can be obtained in the calibration step [Khoshelham and Elberink, 2012]. In this way the relationship between depth and a disparity map is given by the following equation:

$$z = \frac{b \cdot f}{1/8 \cdot (d_{off} - kd)} \quad (2.17)$$

where  $b$  is the baseline between the infrared camera and the RGB camera (in meters), and  $f$  is the focal distance of the cameras (in pixels). Once the depth map is obtained calculating the depth  $z$  for all points, the projection of each point in 3D space is given by:

$$\begin{aligned} p_x &= z \cdot (x - x_c) \cdot 1/f_x \\ p_y &= z \cdot (y - y_c) \cdot 1/f_y \\ p_z &= z \end{aligned} \quad (2.18)$$

Figure 2.8 shows from left to right a depth map, a RGB map and finally the projected coloured point cloud. The depth and rgb maps were obtained from the Kinect device.



**Figure 2.8:** Left: Depth map. Center: RGB Map. Right: Projected point cloud.

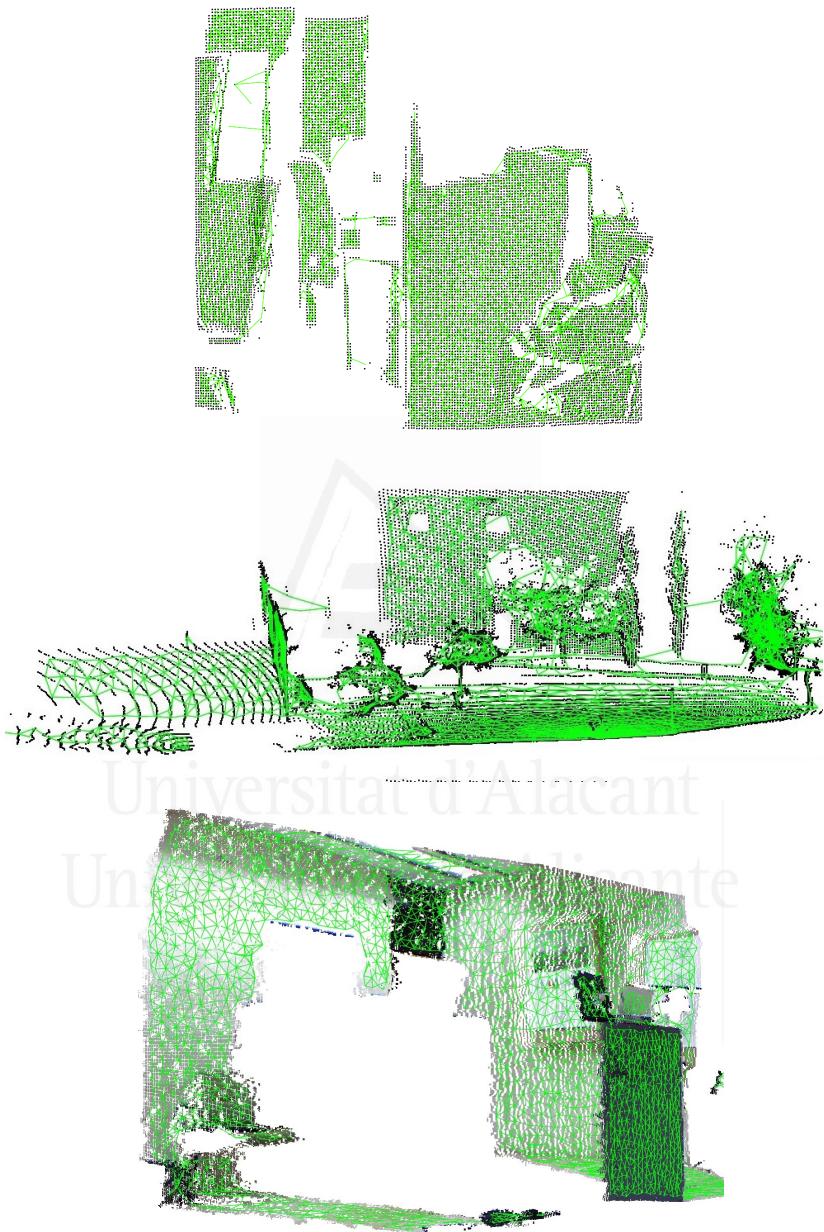
Figure 2.9 shows images of the used 3D sensors.



**Figure 2.9:** 3D sensors used for experiments. From left to right: Sick laser unit LMS-200, Time-Of-Flight SR4000 camera and Microsoft Kinect

Finally, Figure 2.10 shows outdoor and indoor scenes captured using the above presented sensors. The GNG algorithm is able to produce a reduced representation of the input space, maintaining the original topology.

In addition, some public datasets have been used to validate the proposed method. The first one, is the well known Stanford 3D scanning repository [Stanford Scanning Repository, 2013] which contains complete models that have been previously processed. We have also used a dataset captured using the Kinect sensor, it was released by the Computer Vision Laboratory of the University of Bologna [Tombari et al., 2010a]. This dataset was acquired in their lab by means of the Microsoft Kinect sensor and is composed of 7 models and 17 scenes. Finally, generated synthetic data was also used. Figure 2.11 shows real-world objects represented using the proposed method.



**Figure 2.10:** Various scenes represented using the GNG algorithm. From top to bottom: scene captured using the Time-Of-Flight SR4000 camera, the LMS-200 Sick laser and the Microsoft Kinect device



**Figure 2.11:** Different objects represented using the GNG algorithm. Top object belongs to the public dataset [Tombari et al., 2010a]. Rest of objects were captured using the Kinect device in our own laboratory.

### 2.3.2 Topology preservation

The final result of the self-organizing or competitive learning process is closely related to the concept of Delaunay triangulation. The Voronoi region of a neuron consists of all points of the input space for which that neuron is the “winner”. Therefore, as a result of CHL a 3D graph (neural network structure) is obtained whose vertices are the neurons of the network and whose edges are connections between them, which represents the Delaunay triangulation of the input space corresponding to the reference vectors of neurons in the network.

Traditionally, this triangulation resulting from competitive learning was supposed to preserve the topology of the input space. However, [Martinetz and Schulten, 1994] introduced a new condition which restricts this quality. In that work was proposed that a mapping  $\Phi_w$  from a manifold  $V$  in  $A$  preserves the neighbourhood when vectors that are close in the input space  $V$  are mapped to nearby neurons from network  $A$ . The mapping from  $V$  to  $A$  is determined by pointers  $w_i \in \mathbb{R}^D, i = 1, \dots, N$  attached to the vertices  $i$ .

$$\Phi_w : V \rightarrow A, \quad i \in V \rightarrow w_i \in A \quad (2.19)$$

It was also noted that the inverse mapping preserves the neighbourhood if nearby neurons of  $A$  have associated feature vectors close in the input space.

$$\Phi_w^{-1} : A \rightarrow V, \quad c \in A \rightarrow w_c \in V \quad (2.20)$$

Combining the two definitions, a Topology Preserving Network (TPN) is established as the network  $A$  whose mappings  $\Phi_w$  and  $\Phi_w^{-1}$  preserve the neighborhood.

Thus, self-organizing maps or Kohonen maps are not TPN as were traditionally been considered, since this condition only would happen in the event that the topology or dimension of the map and the input space coincide. Since the network topology is established a priori, possibly ignoring the topology of the input space, it is not possible to ensure that the mappings  $\Phi_w$  and  $\Phi_w^{-1}$  preserve the neighborhood.

The Growing Cell Structures [Fritzke, 1993] are not TPN since the topology of the network is established a priori (triangles, tetrahedra, ...). However, it improves the performance compared to Kohonen maps [Kohonen, 1995], due to its capacity of insertion and removal of neurons.

In the case of the Neural Gases like Growing Neural Gas and Neural Gas, the mechanism for adjusting the network through a competitive learning generates an induced Delaunay triangulation, a graph obtained from the Delaunay triangulation, which has only edges of the Delaunay triangulation of points which belong to the input space  $V$ . [Martinetz and Schulten, 1994] demonstrate that these models are TPN.

This capability can be used, for instance, in the application of these models to the representation of objects in different dimensions.

A previous comparative study [Florez-Revuelta, 2002] with Kohonen Maps, Growing Cell Structures and Neural Gas also demonstrated that NG and GNG are topology preserving networks.

### 2.3.2.1 Topology preservation measures

The adaptation of the self-organizing neural networks is often measured in terms of two parameters: the resolution and the topology preservation degree of the input space.

The most widely used measure of resolution is the quantization error [Kohonen, 1995], which is expressed as:

$$E = \frac{1}{N} \sum_{i=1}^N \|w_{s_\xi} - \xi\| \cdot p(\xi) \quad (2.21)$$

where  $s_\xi$  is the closest neuron to the input pattern  $\xi$ .

However, as the GNG is used to generate a 3D representation from an unorganized noisy point cloud sampled with noisy sensors from physical surfaces, it is needed to know the real distance from the generated network to the ground truth of the representation. This measure specifies how close our representation is from the original model, removing noisy data generated by 3D sensors. In order to have a quantitative measure of the input space adaptation of the generated map, we computed the Mean Square Error (MSE) of the map against sampled points (input space).

$$MSE = \frac{1}{|V|} \sum_{\forall p \in V} \min_{i \in A} \|p - w_i\| \quad (2.22)$$

where  $V$  is the input space,  $p$  is a sample that belongs to the input space,  $i$  is the neuron with the minimum distance to the input space sample and  $w_i$  its weight. Euclidean distances to closest neurons are averaged over the entire input space. This measure is computed once network topology learning step is completed.

### 2.3.3 Noise removal

Recent 3D sensors provide valuable information for mobile robotic tasks like scene classification or object recognition, but these sensors produce noisy data that makes impossible applying classical point cloud post-processing techniques. Thus, noise removal and downsampling have become essential steps in 3D machine vision applications. We proposed the use of a 3D filtering and downsampling technique based on a Growing Neural Gas (GNG) network. By means of a competitive learning, it makes an adaptation of the reference vectors of the neurons as well as the interconnection network among them, obtaining a mapping that tries to preserve the topology of an input space. Besides, GNG method is able to deal with outliers in the input data and these features allow to represent 3D spaces, obtaining an induced Delaunay Triangulation of the input space very useful to easily obtain features like corners, edges, curvature and so on.

The proposed method is compared with one state-of-the-art filtering techniques, the Voxel Grid (VG) method. Results presented in next sections will show how the proposed method over-performs VG method on input space adaptation and noise removal capabilities.

The VG filtering technique is based on the input space sampling using a grid of 3D voxels to reduce the number of points. This technique has been used traditionally in the area of computer graphics to subdivide the input space and reduce the number of points [Connolly, 1984, Kobbelt and Botsch, 2004].

VG algorithm defines a voxel grid in the 3D space and for each voxel a

centroid is chosen as the representative of all points that lie on that voxel. It is necessary to define the size of the voxels as this size establish the resolution of the filtered point cloud and therefore the number of points that form the new point cloud. There exist two approaches to the selection of the representative point, one is the selection of the voxel centroid and the other one is based on the selection of the mean of the points lying within the voxel. The second strategy for obtaining internal points has a higher computational cost, but offers better results. Thus, a subset of the input space is obtained that roughly represents the underlying surface. The VG method presents the same problems than other filtering techniques: it is not possible to define the final number of points that represent the surface, geometric information loss due to the reduction of the points inside a voxel and sensitivity to noisy input spaces. The second approach of the VG method will be compared with our GNG-based 3D reconstruction method, as it offers similar features for 3D filtering.

In addition, we performed different experiments on indoor scenes to evaluate the effectiveness and robustness of the proposed method. First, a normal estimation method is computed in order to show how simple features like estimated normals are considerably affected by noisy data. Then, accurate input space adaptation capabilities of the GNG method are demonstrated calculating the Mean Square Error (MSE) of filtered point clouds regard their ground truth. Due to the impossibility of obtaining ground truth data from the Kinect Device, the experiment is performed using synthetic CAD models and data obtained from a simulated Kinect sensor. Experiments, 3D data management (data structures) and their visualization have been developed using the PCL<sup>5</sup> library. In this thesis we used this library to implement, test and visualize most experiments. Some code contributions have been done to the library and can be checked in the public repository of the library. More contributions will be made after the publication of the thesis document.

---

<sup>5</sup>The Point Cloud Library (or PCL) is a large scale, open project [Rusu and Cousins, 2011] for 2D/3D image and point cloud processing.

### 2.3.3.1 Improving normal estimation

Estimation of normal vectors on a geometric surface has been widely used in many application areas such as computer graphics: generating realistic illumination of the surfaces and in computer vision as geometric features of the observed environment: keypoints with high curvature (corner or edge points).

Given a geometric surface it is possible to estimate the direction of the normal vector at a point obtaining the outward facing vector of the surface. However, we are focused on point clouds without information about the surfaces that compose it.

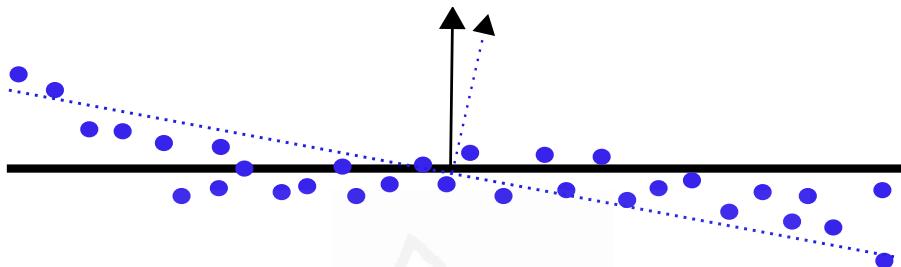
We approached the normal vector estimation of a point efficiently by using its neighbourhood to calculate the normal vector. Here we focus on the estimation of the plane that best fits the neighbourhood of points using least squares. In this way the point normal vector is calculated as the plane normal vector. The search for this plane is reduced to the calculation of eigenvalues and eigenvectors, Principal Component Analysis (PCA) [Jolliffe, 1986] of the covariance matrix created using the neighbourhood of the point on which we want to know its normal vector. The orientation of the normal vector is easily calculated because we know the point of view of the scene, in this case the sensor position, so that all normal vectors must be facing consistently toward the point of view satisfying the following equation:

$$\vec{n}_i \cdot (v_p - p_i) > 0 \quad (2.23)$$

where  $\vec{n}_i$  is the calculated normal vector,  $v_p$  is the point of view, and  $p_i$  is the target point. In cases where this constraint is not satisfied, it is necessary to reverse the sign of the calculated normal vector. Figure 2.12 shows the effect caused by normal estimation on noisy data. Normal estimation was computed on the original and filtered point cloud using the same search radius to define the point neighbourhood:  $r_s = 0.1$  (meters). Radius was empirically selected based on size and features of objects placed in the scene.

Normal estimation methods based on the analysis of the eigenvectors

and eigenvalues of a covariance matrix created from the nearest neighbours are very sensitive to noisy data. Therefore, in this experiment, we computed normals on raw and filtered point clouds in order to demonstrate how a simple 3D processing step like normal or curvature estimation are affected by the presence of noise and how the proposed method improves normal estimation producing more stable normals.

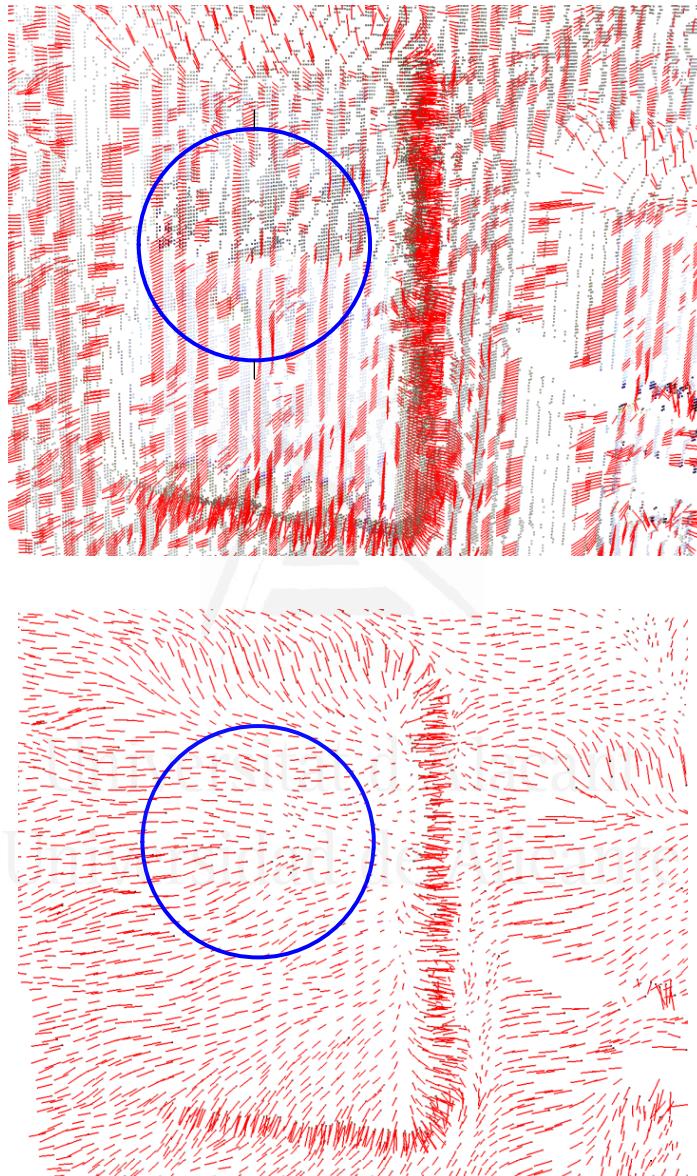


**Figure 2.12:** Noise causes error in the estimated normal

Figure 2.13 shows how more stable normals are estimated using filtered point cloud produced by the GNG method. Normals are considered more stable as their distribution is smooth and also they have less abrupt changes in their directions as it is shown in Figure 2.13. Moreover, a flat wall with some small changes in its surface was selected to appreciate changes in normal directions. 20,000 neurons and 1,000 patterns  $\lambda$  are used as configuration parameters for the GNG method in the normal estimation experiment (bottom).

### 2.3.4 Filtering quality: input space adaptation

The Growing Neural Gas (GNG) and Neural Gas (NG) algorithms both have good skills to tightly represent input data distributions maintaining the original topology. Various experiments were performed using both methods to demonstrate input space adaptation capabilities. They were performed using synthetic data as it is not possible to obtain ground truth measurements for real-world scenes and objects captured with above presented sensors.



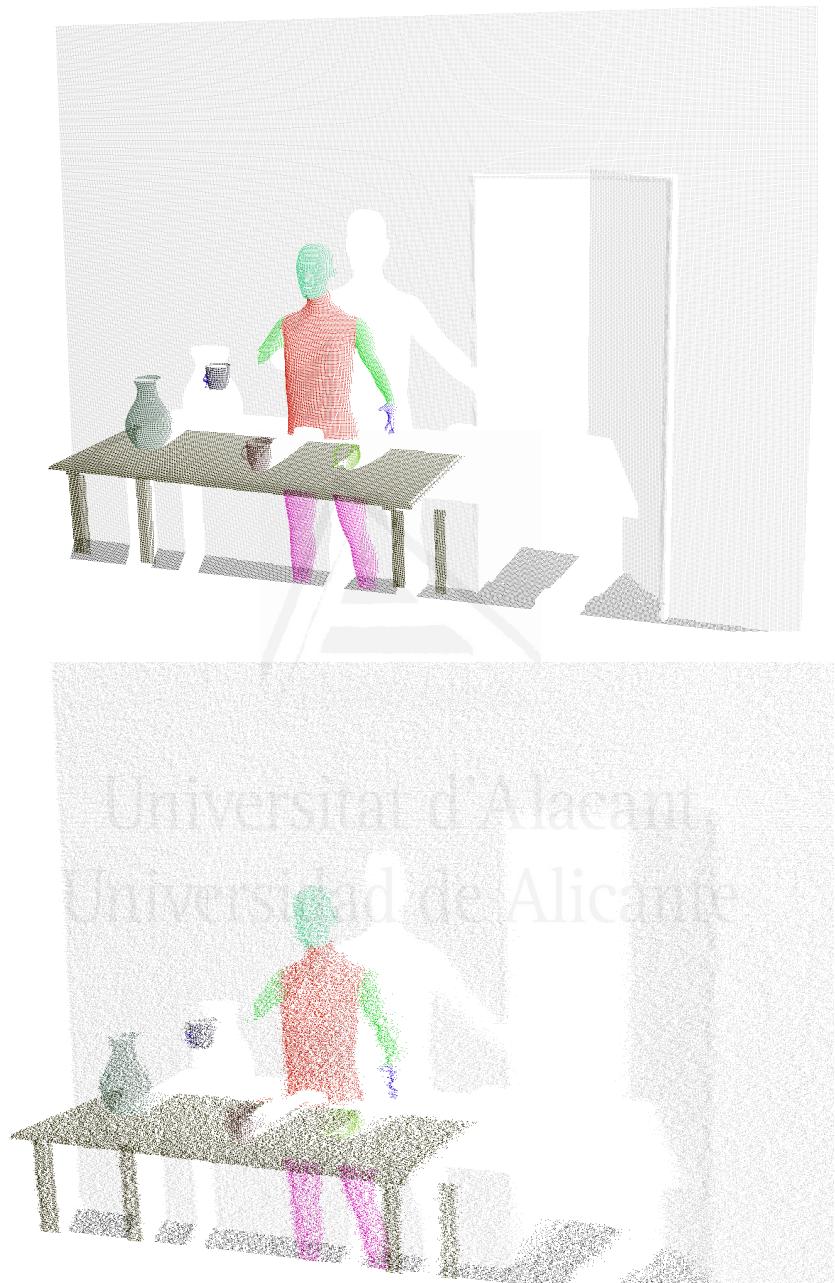
**Figure 2.13:** Normal estimation comparison. Top: Normal estimation on raw point cloud. Bottom: Normal estimation on filtered point cloud produced by the GNG method

### 2.3.4.1 Growing Neural Gas input space adaptation

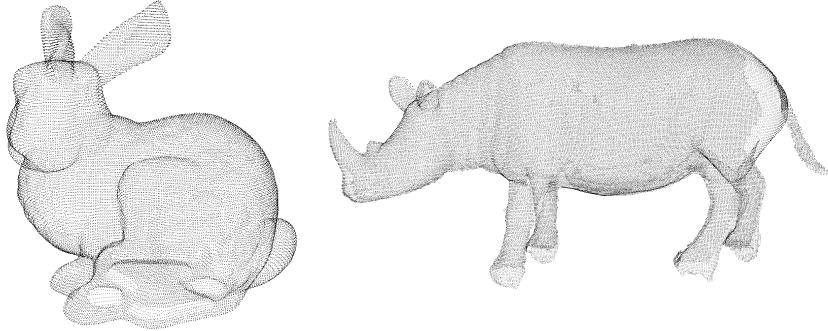
In this experiment we demonstrated how the GNG method yields better input space adaptation to noisy data than other filtering and downsampling methods like Voxel Grid. In order to perform the experiment, ground truth data was required to calculate the MSE error regarding original data. As the Kinect device does not provide ground truth information, synthetic CAD models and data obtained from a simulated Kinect sensor were used as ground truth. For that purpose the Blensor software [Gschwandtner et al., 2011] was used. It allowed us to generate synthetic scenes and to obtain partial views of the generated scene as if a Kinect device was used. The main advantage of this software is that it provides ground truth (absence of noise). Figure 2.14 shows a synthetic scene generated using Blensor. The ground truth scene without noise is shown on the top side while the partial view of the same scene adding simulated noise is shown on the bottom side. Noise was simulated using a Gaussian distribution with zero mean and different deviation factors. Figure 2.15 shows CAD models used in this experiment. Bunny and rhino models are respectively model 1 and 2. These models are considered CAD models as they present almost no error or a level of precision below 0.1 millimetres, which is far from the precision obtained using low cost 3D sensors as it was introduced in Section 2.3.1.

To perform this experiment, we computed the MSE of the filtered point cloud with Voxel Grid and with the GNG method respect to the ground truth. MSE was used to measure the filtered point cloud error relative to the ground truth and therefore it is a quantitative measure of the accuracy of the filtered point cloud. MSE is expressed in metres. Voxel Grid method presents some drawbacks as it does not allow specifying a fixed number of points, as the number of points is given by the voxel size used for building the grid. We forced the convergence to an approximate number in our experiments, making their comparison fairer. By contrast, the GNG neural network allows us to specify the exact number of end points that represent the input space. The experiment demonstrates the higher accuracy of the GNG network compared with Voxel Grid method.

Table 2.1 shows the adaptation MSE for different models and scenes



**Figure 2.14:** Synthetic scene. Top: Ground truth without noise. Bottom: simulated Kinect view with added noise:  $\sigma = 0.5$



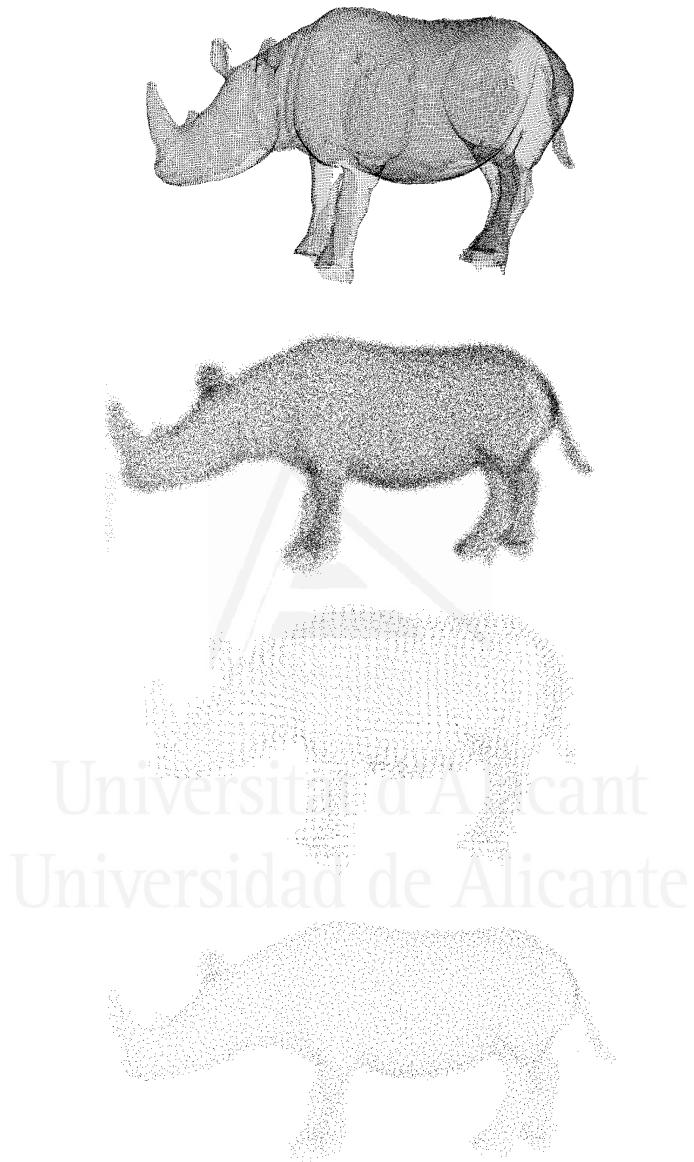
**Figure 2.15:** Top: bunny model (34,834 points). Bottom: rhino model (79,934 points).

simulated Kinect	VG 5000	VG 10000	GNG 5000 250λ	GNG 10000 500λ
<b>scene 1</b> $\sigma = 0.15$	0.0067	0.0064	<b>0.0017</b>	0.0022
<b>scene 1</b> $\sigma = 0.25$	0.0197	0.0181	<b>0.0053</b>	0.0065
<b>scene 1</b> $\sigma = 0.40$	0.0475	0.0430	<b>0.0156</b>	0.0185
<b>scene 2</b> $\sigma = 0.15$	0.0053	0.0051	<b>0.0013</b>	0.0017
<b>scene 2</b> $\sigma = 0.25$	0.0148	0.0135	<b>0.0041</b>	0.0051
<b>scene 2</b> $\sigma = 0.40$	0.0372	0.0336	<b>0.0122</b>	0.0143

CAD model	VG 5000	VG 10000	GNG 5000 250λ	GNG 10000 500λ
<b>model 1</b> $\sigma = 0.15$	0.0643	0.0641	0.0684	<b>0.0559</b>
<b>model 1</b> $\sigma = 0.25$	0.0981	0.0994	0.0768	<b>0.0642</b>
<b>model 1</b> $\sigma = 0.40$	0.2037	0.2276	<b>0.0903</b>	0.0924
<b>model 2</b> $\sigma = 0.15$	0.1540	0.1504	0.1756	<b>0.1209</b>
<b>model 2</b> $\sigma = 0.25$	0.3055	0.3227	0.1938	<b>0.1430</b>
<b>model 2</b> $\sigma = 0.40$	0.8259	0.8895	0.2346	<b>0.2122</b>

**Table 2.1:** Input space adaptation MSE for different models. Voxel Grid versus GNG. Numbers in bold provide the best results.

using the GNG and the Voxel Grid method. On the top of the Table 2.1, adaptation MSE is calculated for partial views obtained from a simulated Kinect sensor (meters) while on the bottom is calculated for CAD models (millimetres). Different levels of noise  $\sigma$  are applied to the ground truth data (in the three dimensions XYZ). Results presented in Table 2.1 shows how the GNG method provides a lower mean error and therefore better adaptation to the original input space, maintaining a better quality of representation in areas with a high degree of curvature and removing the noise generated by the sensor. The Voxel Grid method filters all the



**Figure 2.16:** Filtering quality using 10,000 points. GNG vs Voxel Grid comparison. From top to bottom: Noisy model  $\sigma = 0.4$ , Original CAD model, filtered model using GNG method and filtered model using Voxel Grid.

points present approximating (i.e., downsampling) them with their centroid, it does not represent the underlying surface accurately causing a worse adaptation. Experiments were performed with a fixed number of points, and in the case of the GNG it was tested with different number of input signals  $\lambda$  generated by iteration, and different number of neurons, obtaining better results with a high number of adjustments. Figure 2.16 visually shows adaptation MSE presented in Table 2.1 for model 2. Voxel Grid output representations does not accurately fit input space, smoothing information in the edges and corners.

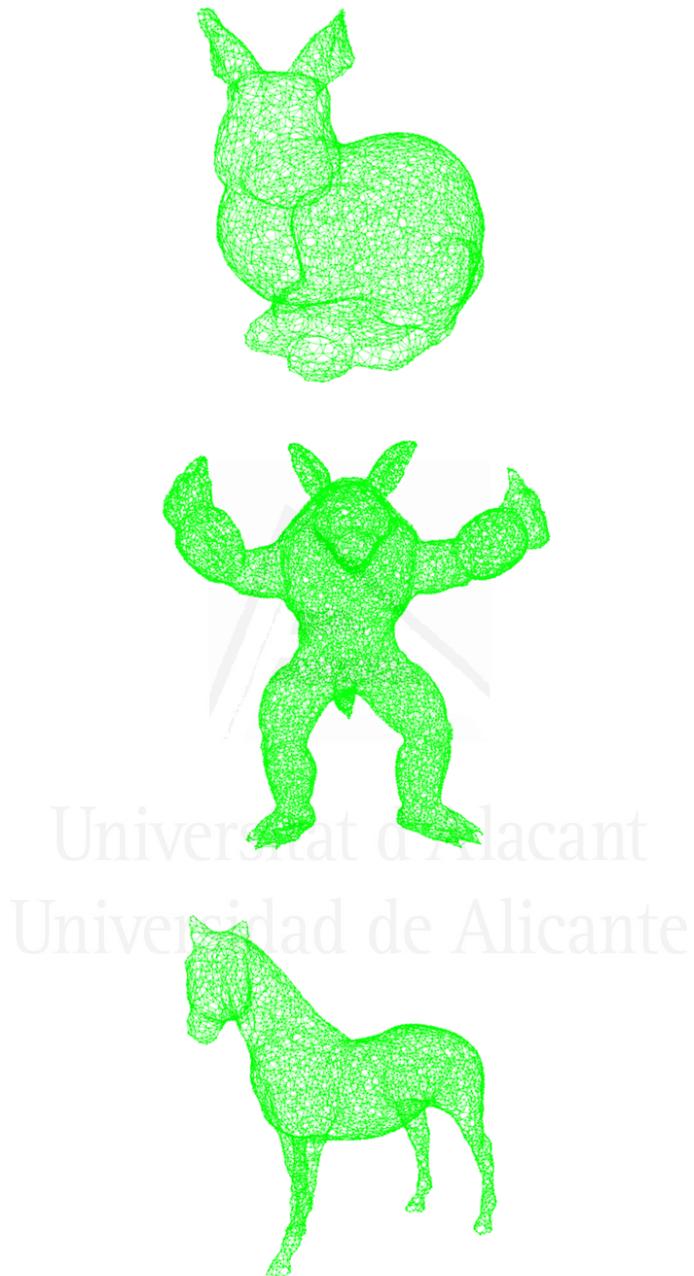
The proposed method created a GNG network over the raw point cloud, providing a 3D structure which has less information than the original 3D data, but keeping the 3D topology. We demonstrated how the proposed method obtains better adaptation to the input space than other filtering methods like Voxel Grid, obtaining lower adaptation MSE on simulated scenes and CAD models.

#### 2.3.4.2 Neural Gas input space adaptation

This section presents some experiments related with the input space adaptation and 3D reconstruction capabilities of the NG method. First, quality of representation is studied depending on the parameters established for the NG method, obtaining different adaptation mean square errors (MSEs) for 3D model reconstructions. Furthermore, a comparison between state-of-the-art Voxel Grid filtering method is presented demonstrating the accuracy of the representation generated by the NG network. Experiments were performed using noise-free models with added Gaussian noise to simulate noisy data similar to which is present in low-cost sensors. Similar experiments to those shown in the previous section were performed to validate the NG algorithm.

Figure 2.17 shows different 3D models reconstructed using NG method. Different number of neurons have been used since original models are represented with different number of points.

In the first experiment, using the 3D Stanford bunny model as ground truth, a study of NG parameters was carried out. Stanford bunny model has 34,834 points. Different levels of Gaussian noise, number of neurons



**Figure 2.17:** Reconstructed 3D models using NG method. From top to bottom: Bunny, armadillo and horse. The number of neurons of the reconstructed models is 5k, 8k and 15k.

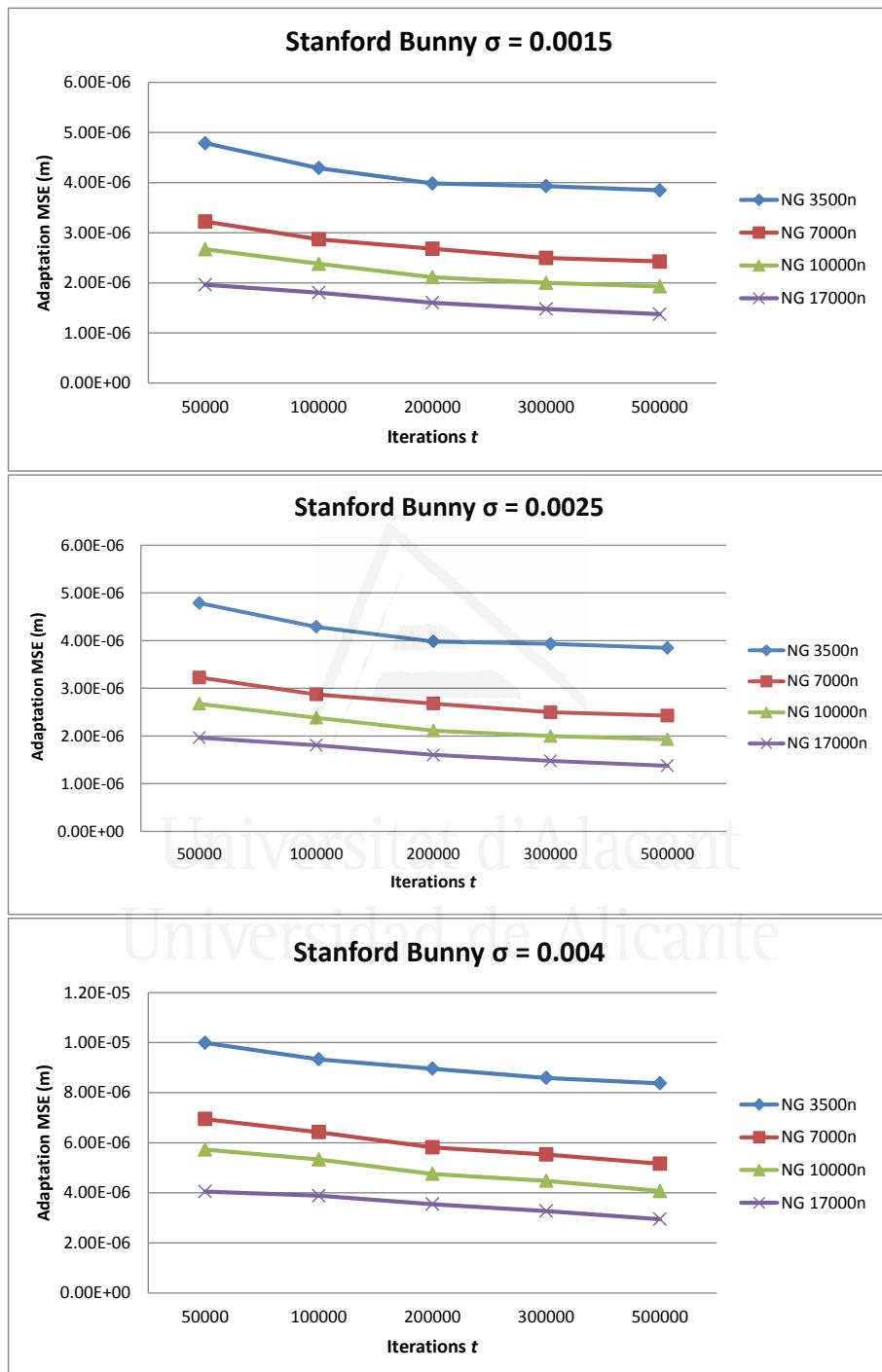
and number of iterations were exhaustively tested obtaining the adaptation MSE compared to the original model and surface reconstruction.

Figure 2.18 shows results from the NG parameters study. From presented results it can be concluded that the number of iterations parameter converge to a similar spatial approximation error after a defined number of iterations. Besides, it was demonstrated using different number of neurons and levels of Gaussian noise. Chosen number of neurons represents 10%, 20%, 30% and 50% of the original number of points in the input space. Finally, it is also demonstrated how as the number of neurons is increased, the MSE decreased converging to a similar error for latest configurations, 10,000 and 17,000 number of neurons. However, using a large number of neurons caused that present error in the model was also present in the reconstructed model. Therefore, it is convenient to use a reduced number of points to represent noisy models.

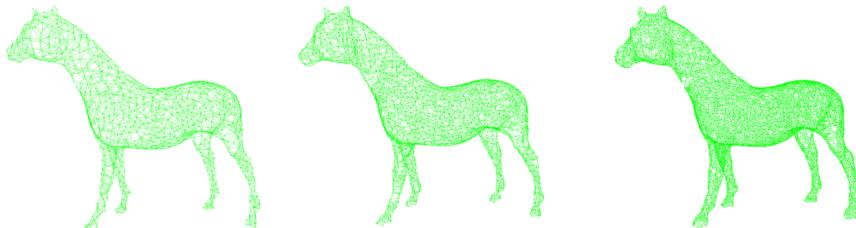
Figure 2.19 shows surface reconstruction capabilities of the NG method. This method allows establishing the number of vertex that represents the reconstructed model, obtaining models with different resolution depending on the number of neurons that comprises the network. Moreover, the input data is not completely processed, but it is only sampled one point at each time so we assumed that the time performance of the algorithm is practically independent of the size of the input data set, enabling the use of large datasets containing hundreds of millions points.

Figure 2.20 shows how the NG method provides a lower MSE and therefore better adaptation to the original input space, maintaining a better quality of representation in areas with a high degree of curvature and eliminating the noise generated by the sensor.

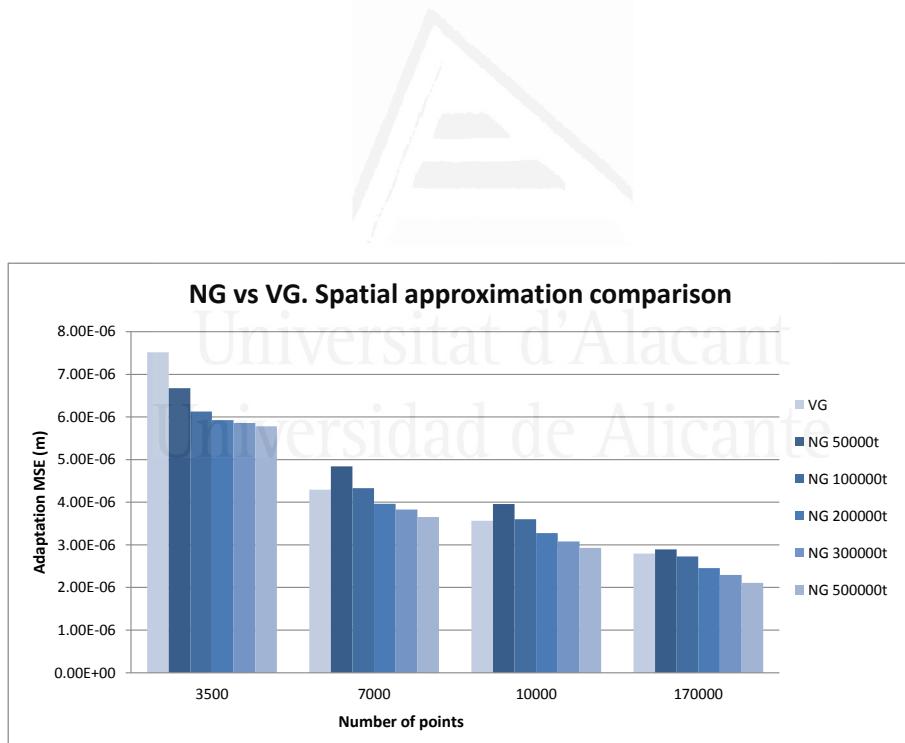
Figure 2.21 visually shows the results discussed in Figure 2.20. It can be observed how the adaption of the filtered points obtained with the Voxel Grid method produces less accurate results than the ones obtained using the NG method. This experiments showed similar results in terms of input space adaptation compared to the results presented above using the GNG method (Figure 2.16).



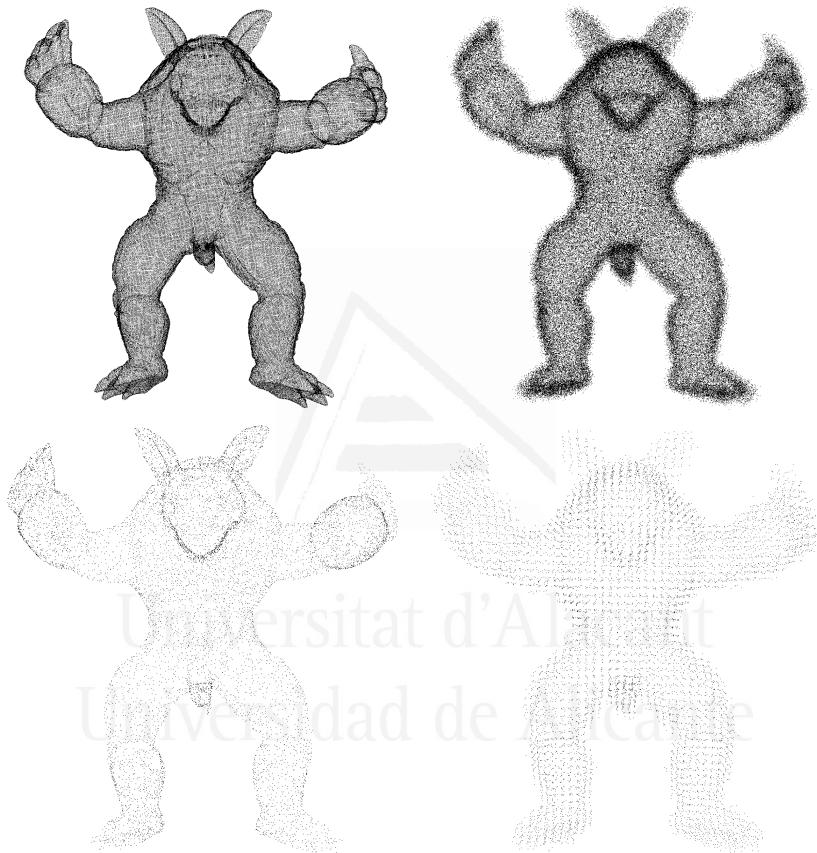
**Figure 2.18:** NG parameters study using Stanford bunny model. Three different levels of noise  $\sigma$  are applied



**Figure 2.19:** Horse model reconstructed incrementally from an unorganized point cloud of 148k points. From left to right, the number of vertex of the reconstructed mesh is 1k, 2.5k and 8k.



**Figure 2.20:** NG parameters study using Stanford bunny model. Three different levels of noise  $\alpha$  are applied.



**Figure 2.21:** Filtering quality using 10,000 points. NG vs Voxel Grid comparison. Top left: original noise-free model. Top right: noisy model  $\alpha=0.6$  mm. Bottom left: filtered model using NG method. Bottom right: filtered model using Voxel Grid.

### 2.3.5 Colour interpolation

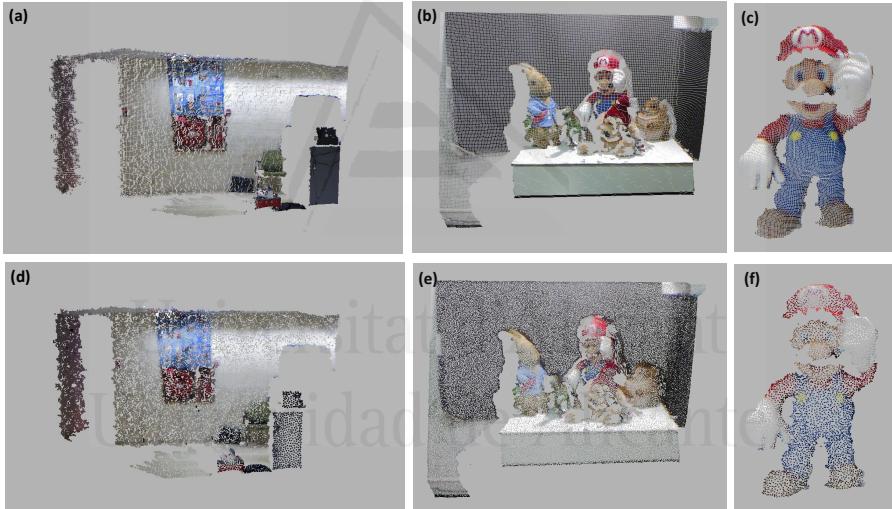
As modern 3D sensors provide us with colour information, the proposed methods were modified regarding the original versions considering also point cloud colour. Input space dimension is increased from 3 to 6 adding red, green and blue colour components. Now the input distribution is defined in  $\mathbb{R}^d$  where  $d = 6$ . Most SOM-based approaches already presented only considered spatial information as neuron's weight vector  $w_c$ , so we modified the learning step adding colour to the neuron's weight vector  $w_c$  and considering it during the learning process, now the dimension of the neuron's weight vector is 6 including spatial and colour information. Color values were normalized ranging from 0.0 and 1.0. Colour information is considered during the weight adaptation step but it was not included in the CHL (winning neurons) step as we still are focused on the input space topology learning. Therefore, winning neuron step only compute euclidean distance using  $x,y,z$  components. Figure 2.22 shows how the GNG method generated a down-sampled version of captured coloured point clouds, interpolating the colour of original observations and achieving a good topological fitting for different objects and scenes. We called this version Colour-GNG.

In order to validate and compare the colour version of the GNG, we implemented a different strategy to consider point cloud colour. Instead of adding colour information to the learning process, a post-processing step to compute colour information is added to the process. Once the network has been adapted to the input space (original GNG) and it has completed the learning step, each neuron of the network computes colour information from closest input patterns. Colour information of each neuron is calculated as the average of weighted values of the K-nearest input patterns, obtaining an interpolated value of the surrounding point. Colour values are weighted using Euclidean distance from input pattern to its closest neuron reference vector. K-nearest neighbours are obtained using a radius search process, using as a radius the resolution of the generated map by the GNG algorithm. Therefore, RGB colour for each neuron is calculated using the following equation:

$$RGB_i = \psi \sum_{\forall j \in N_i} (RGB_j \cdot w(j - i)) \quad (2.24)$$

where  $N_i$  represents the nearest input patterns of the neuron  $i$ ,  $i$  is the neuron being processed and  $w(j - i)$  is the distance weighted function between the neighbouring pattern  $j$  and the neuron itself  $i$ . Using that distance weighted function the weight of a color input pattern decays exponentially as the distance to the neuron increases.  $\psi$  is a normalization factor that makes RGB components range between 0.0 and 1.0.

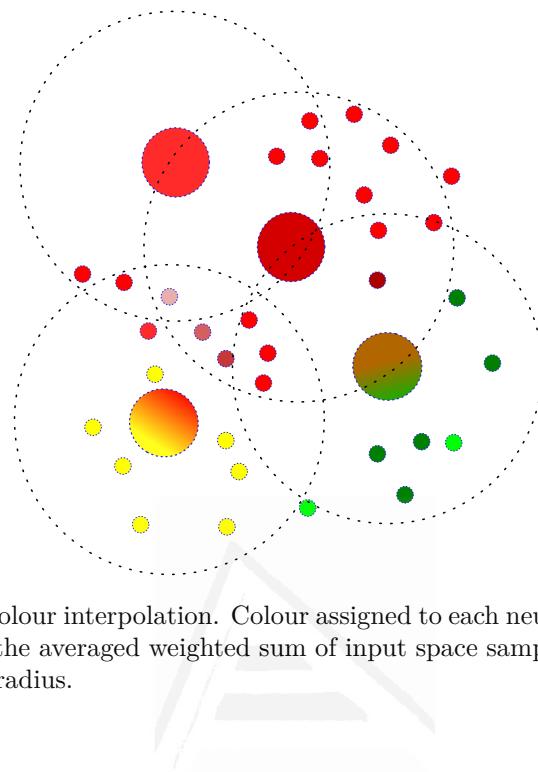
$$w(j - i) = e^{-\|j-i\|} \quad (2.25)$$



**Figure 2.22:** Several objects and scenes are down-sampled using the Colour-GNG representation. (a),(b),(c) show original pointclouds. (d),(e),(f) show down-sampled point clouds using the proposed method.

Figure 2.23 visually shows this process. Although this search is accelerated using a Kd-tree structure it is considerably slower than the colour version of the GNG. Colour-GNG in the same learning process is able to adapt its neurons' weights fitting accurately the input space.

Figures 2.24 and 2.25 show various observations that have been created using both approaches. Colour-GNG produces a map that successfully

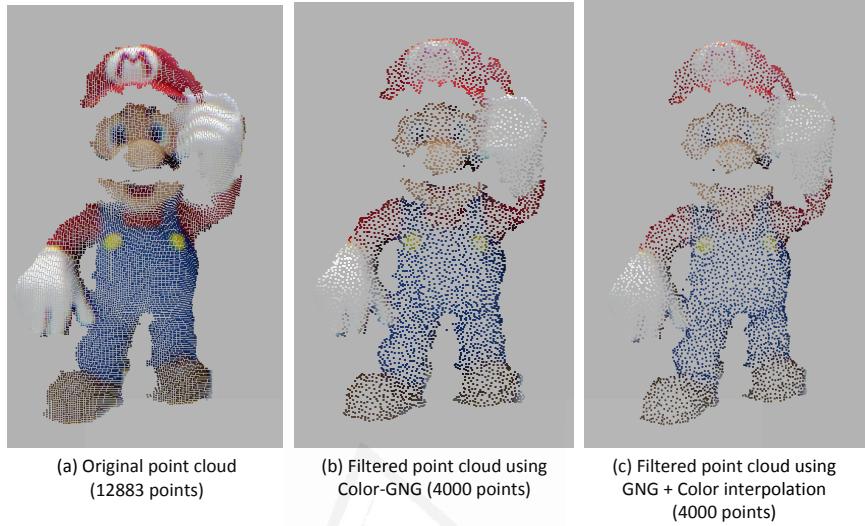


**Figure 2.23:** Colour interpolation. Colour assigned to each neuron (large circles) is calculated as the averaged weighted sum of input space samples (small circles) within a search radius.

interpolates input colour information producing an useful down-sampled map. Moreover, results were similar to those obtained with the colour interpolation post-processing step.

Finally, some quantitative results are presented in Table 2.2 showing the mean error between estimated colours using the post-processing interpolation step and the proposed Colour-GNG method. The error is computed over the three components of the RGB model. The maximum error obtained in a component between estimated colours using the post-processing step and the Colour-GNG is less than four units, considering that each component is represented using unsigned char format.

With these results we can conclude that Colour-GNG method is able to obtain similar results compared to complex post-processing steps and reducing the processing time.



**Figure 2.24:** Mario figure is down-sampled using the Colour-GNG method. Results are similar to those obtained with the colour interpolation post-processing step.

	Mean error		
	Red	Green	Blue
<b>Scene 1 - <math>20,000n</math> <math>50\lambda</math></b>	1	1	2
<b>Scene 1 - <math>50,000n</math> <math>100\lambda</math></b>	1	1	1
<b>Scene 2 - <math>20,000n</math> <math>50\lambda</math></b>	4	4	4
<b>Scene 2 - <math>50,000n</math> <math>100\lambda</math></b>	2	2	2
<b>Object 1 - <math>3,000n</math> <math>50\lambda</math></b>	2	3	3
<b>Object 1 - <math>5,000n</math> <math>100\lambda</math></b>	1	2	2
<b>Object 2 - <math>3,000m</math> <math>50\lambda</math></b>	3	3	3
<b>Object 2 - <math>5,000n</math> <math>100\lambda</math></b>	2	2	2

**Table 2.2:** Colour Mean error between computed colours using a post-processing interpolation step and the Colour-GNG. Various scenes and objects were tested obtaining less than four units of error even with colourful scenes.

### 2.3. GNG-based method to represent real-world noisy observations

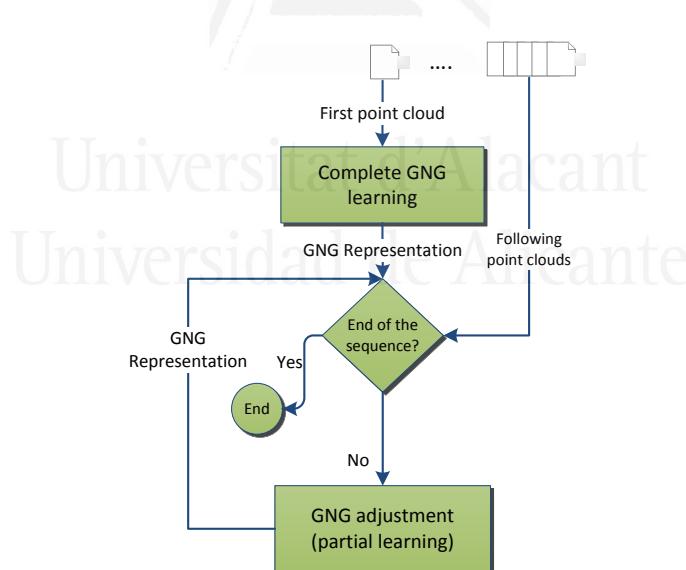


**Figure 2.25:** Two different scenes captured using the Kinect sensor are represented using the Colour-GNG method. Results are similar to those obtained with the colour interpolation post-processing step.

### 2.3.6 Point cloud sequences management

GNG capabilities for representing single 3D observations have been already shown in this chapter but we propose the extension of this method to allow representing continuous observations. In this way GNG representation can be used in complex computer vision problems as scene understanding by long-term observation or Simultaneous Localization and Mapping (SLAM).

To obtain the representation of the first point cloud, the entire algorithm presented in Figure 2.6 is computed, performing the complete learning process of the network. However, for the following point clouds, only internal loop of the algorithm is processed. In this way, neurons already placed in the map are adjusted and moved towards changes in the observation (partial learning). Only weight modification and spatial relationship are changed maintaining the number of neurons. Figure 2.26 shows the improved workflow for processing point cloud sequences.



**Figure 2.26:** An improved workflow to manage point cloud sequences using the GNG algorithm.

One of the most advantageous features of the extension of the GNG

for processing sequences of point clouds is that is not required to restart the learning of the network for each new point cloud that is captured from the scene. Once the initial map is generated, this map can be used as the initial topology for the next point cloud. In order to take advantage of this feature a good sample rate is required, so the changes between different captured point clouds are small. With this approach, and using a record of processed point clouds and a small network adjustment, it is possible to obtain a new representation of the changing observation. This extension also provides a speed-up in the runtime of the learning step as neurons are kept between frames. This adaptive method is also able to face real-time constraints, because the number  $\lambda$  of times that the internal loop is performed can be chosen according to the available time between two successive frames that depends on the acquisition rate. The mean time to adjust the GNG network on a frame is less than 50 ms, using the adaptive method. More results about runtimes will be presented later in this thesis in Chapter 4 and 5, where a GPU implementation of the algorithm is proposed and also this implementation is integrated into a 6DoF egomotion application.

This extended version of the algorithm will be further detailed and validated in the case of study presented in Chapter 5. In that case of study a six degrees of freedom (6DoF) mapping application (Section 5.2.1) is proposed. The extension of the GNG method is useful for managing point cloud sequences reducing processing times of the following frames.

### 2.3.7 3D surface reconstruction

Three-dimensional surface reconstruction is not considered in the original GNG algorithm as it only generates wire-frame models. As we presented in Section 2.1, some previous works as [Holdstein and Fischer, 2008, Do Rego et al., 2010, Barhak, 2002] have already considered the creation of 3D triangular faces modifying the original GNG algorithm. In these works, mesh operators as edge split, edge collapse, triangle subdivision, etc, were implemented to achieve that purpose. However, although most of these works produced 3D meshes they have a common drawback or disadvantage, the need of post-processing steps to finally close holes and

missing faces that were not created during the learning stage. For example, in [Holdstein and Fischer, 2008], the proposed MGNG method required some post-processing steps in order to create a complete 3D model. It cannot deal with the creation of the complete mesh during the learning stage. In [Do Rego et al., 2010, Barhak, 2002], although the Competitive Hebbian Learning was extended considering the creation of 2-manifold meshes and face reconstruction, it was also required to apply some post-processing steps to create a complete model.

Figure 2.27 shows the result of using an existing GNG-based method for surface reconstruction [Do Rego et al., 2010] without applying post-processing steps. The reconstructed model has a lot of gaps and holes that makes the model not suitable for computer vision applications. Therefore, in this thesis we proposed a GNG-based method able to perform surface reconstruction without applying post-processing steps.



**Figure 2.27:** Different views of reconstructed models using an existing GNG-based method for surface reconstruction. Post-processing steps were avoided causing gaps and holes in the final 3D reconstructed models.

In this section, we detail the proposed extension of the already described GNG method to generate full coloured 3D models without applying post-processing steps.

### 2.3.7.1 Extended CHL

Original CHL, presented in Section 2.2.2.1, only considered the creation of edges between neurons producing wire-frame 3D representations. Therefore, it is necessary to modify this process in order to create triangular faces during the learning process. Based on [Do Rego et al., 2010] and [Barhak, 2002] we extended the CHL developing a method able to produce full 3D meshes. In contrast to existing methods mentioned above, our extension does not need post-processing steps. The 3D mesh is created during the learning stage.

The edge creation stage, presented in Section 2.2.2.2, was also extended considering the creation of triangular faces during this process. Algorithm 1 describes our extended CHL to produce triangular faces.

In order to avoid non-manifold and overlapping edges, the edge creation step was modified restricting the creation of edges if the winning neurons  $s_1$  and  $s_2$  have already more than two common neighbours. This constraint helps avoiding edges with more than two incident triangles. Then, for every sampled point, a face is created whenever the already existing edges or the new ones form a triangle. Moreover, if the creation of faces would produce edges with more than two incident faces, then the face is not created avoiding overlapped triangles and non manifold meshes. During the creation of triangular faces it is checked if the face to be created already exist, in that case, the face is not created. Figure 2.28 shows common situations produced during the CHL and how our method create edges and triangular faces in those cases.

The age scheme presented in the original GNG algorithm was also considered to remove those edges that have an age higher than a given threshold  $age_{max}$ . This age scheme was extended including the removal of faces that shared this edge. Furthermore, in order to obtain regular triangular faces we included another constraint that was introduced in [Mole and Araújo, 2010]. This constraint is based on the Thales sphere

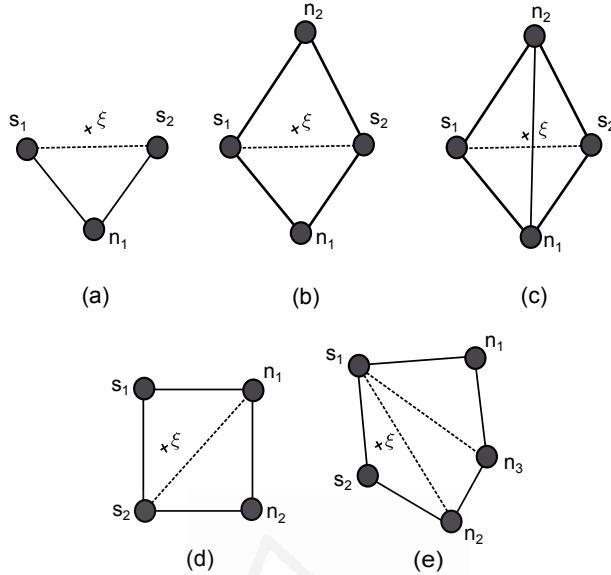
```

input : A point cloud
output: 3D mesh

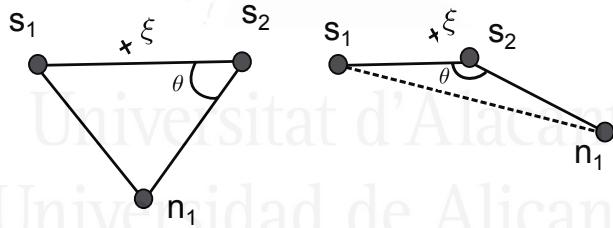
1 For each input pattern presented to the network, the two nearest
neurons to the input pattern are selected as winning neurons  $s_1$  and  $s_2$ ;
2 if  $s_1$  and  $s_2$  are already connected by an edge then
3   Set edge age to 0 in order to “reinforce” it;
4   Check edge removal mechanism based on the Tales Sphere;
5   if  $s_1$  and  $s_2$  have one or two common neighbours then
6     foreach common neighbour  $n_i$  do
7       | Create a face  $f$  using  $s_1$ ,  $s_2$  and  $n_i$ ;
8     end
9   end
10  if  $s_1$  and  $s_2$  have one or less common neighbours then
11    if There exist two neighbours  $n_1$  and  $n_2$  of  $s_1$  and  $s_2$  respectively
that are connected and are not common to  $s_1$  and  $s_2$  then
12      | Triangulate rectangular hole (Figure 2.28d): Create two
faces using  $s_1$ ,  $s_2$ ,  $n_1$  and  $s_2$ ,  $n_1$ ,  $n_2$ ;
13    end
14    else if There exist two neighbours  $n_1$  and  $n_2$  of  $s_1$  and  $s_2$ 
respectively that are not connected between them and are not
common to  $s_1$  and  $s_2$  then
15      | Triangulate pentagonal hole (Figure 2.28e): Create three
faces using  $s_1$ ,  $s_2$ ,  $n_2$ ;  $s_1$ ,  $n_2$ ,  $n_3$  and  $s_1$ ,  $n_1$ ,  $n_3$ ;
16    end
17  end
18 else
19  if  $s_1$  and  $s_2$  have two common neighbours  $n_1$  and  $n_2$  then
20    if  $n_1$  and  $n_2$  are already connected (Figure 2.28c) then
21      | Edge between  $n_1$  and  $n_2$  is removed;
22      | Faces coincident to  $n_1$  and  $n_2$  are removed;
23      | Create two faces using  $n_1$ ,  $s_1$ ,  $s_2$  and  $n_2$ ,  $s_1$ ,  $s_2$ ;
24    else
25      | Create an edge between  $s_1$  and  $s_2$ ;
26      | Create two faces using  $n_1$ ,  $s_1$ ,  $s_2$  and  $n_2$ ,  $s_1$ ,  $s_2$ ;
27      | (Figure 2.28b);
28    end
29  else
30    | Create edge between  $s_1$  and  $s_2$ ;
31    if  $s_1$  and  $s_2$  have one common neighbour  $n_1$  (Figure 2.28a)
then
32      | Create a face  $f$  using  $s_1$ ,  $s_2$  and  $n_1$ ;
33    end
34  end
35 end

```

**Algorithm 1:** Pseudo-code of the extended CHL stage.



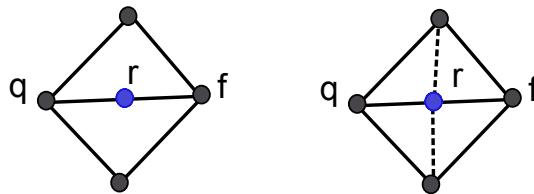
**Figure 2.28:** Considered situations for edge and face creation during the extended CHL.



**Figure 2.29:** Edge removal constraint based on the Tales sphere. Left: The triangle formed by these 3 neurons is close to a right triangle, therefore it is not removed. Right: The edge connecting  $s_1$  and  $n_i$  is removed as the angle formed by vectors  $s_2 - s_1$  and  $n_i - s_1$  is larger than  $3/4\pi$ . Moreover, the triangle formed by these edges is also removed.

concept. For every edge that already existed in the CHL process, this mechanism computes the angle between the vectors formed by  $s_1 - s_2$  and  $s_1 - n_i$  where  $n_i$  is a common neighbour of  $s_1$  and  $s_2$ . If this angle  $\theta > \theta_{max}$  the edge between  $s_1$  and  $n_i$  is removed. Faces incident to this edge are also removed. Different values for  $\theta_{max}$  were tested, obtaining regular triangles for  $\theta_{max}$  values between  $2/3\pi$  and  $3/4\pi$ .

### 2.3.7.2 Inserting and deleting neurons



**Figure 2.30:** Face creation process during the insertion of new neurons. Left: neuron insertion between the neuron  $q$  with highest error and its neighbour  $f$  with highest error. Right: four new triangles and two edges are created considering  $r$ ,  $q$  and  $r$ .

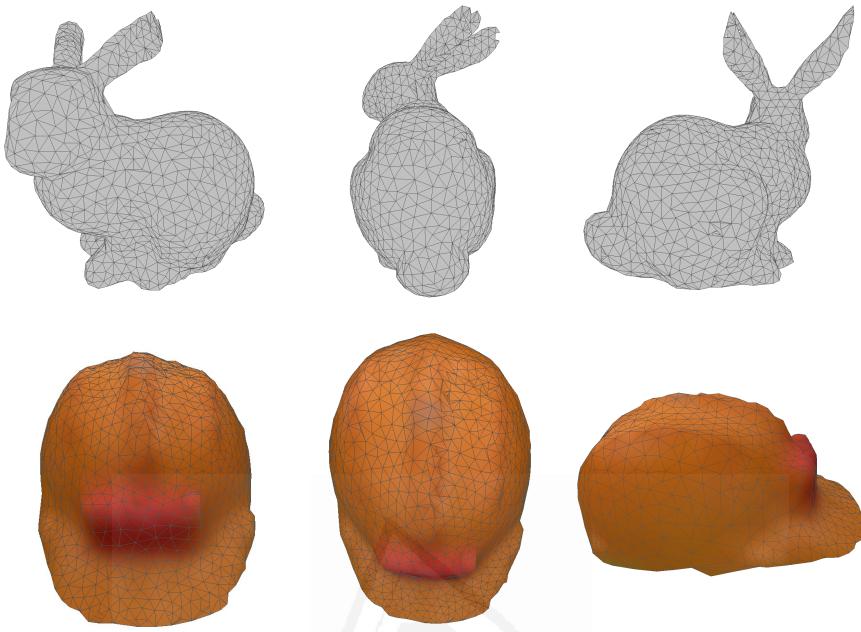
The neuron insertion process was also modified. Every time a neuron is inserted in the network, an edge between the neurons with highest errors is removed and therefore, triangles incident to this edge are also removed. If it is possible new faces are created along with the new neuron. Figure 2.30 shows this process.

Finally, if the given number of neurons is reached, all input patterns are presented to the network in order to close possible gaps and holes that were generated during the learning process.

### 2.3.7.3 Experiments

In this section, different experiments are shown validating the capabilities of our extended GNG method to create 3D meshes. The proposed method is also able to learn colour information and create coloured 3D meshes. 3D models were rendered using colour information stored in the neurons and a triangle smooth-shading technique. The method was tested using different models: a complete model of a builder helmet, a model of a person and some scenes. These were captured using the Kinect sensor. The model of the person and the scene are not complete models, they are partial 3D views. Moreover, the method was tested using a model of a foot and a high precision model like the Stanford bunny.

Figure 2.31 and 2.32 show the ability of the proposed method to create colour meshes of different types of models. It can be seen how most holes



**Figure 2.31:** Reconstructed models using our extended GNG method for face reconstruction and without applying post-processing steps. Top: Stanford bunny. Bottom: builder helmet.

showed in 2.27, generated by existing extensions of the GNG method for surface reconstruction, were not generated using the proposed method. Furthermore, we applied this extension to reconstruct partial views of scenes captured using the Kinect sensor (Figure 2.33).

The proposed extension is also able to generate 3D meshes with different resolutions and therefore, detail level. Figure 2.34 shows the builder helmet model reconstructed using different number of neurons, creating meshes with different level of detail.

However, the proposed method still produced some small gaps in the generated 3D reconstructions. Figure 2.35 shows small gaps and holes created in some of the experiments carried out. These are caused by the randomness of the network learning stage. Moreover, in some cases triangles are removed caused by the edge removal ageing scheme, which also is responsible for the good level of adaptation and relationship between neu-

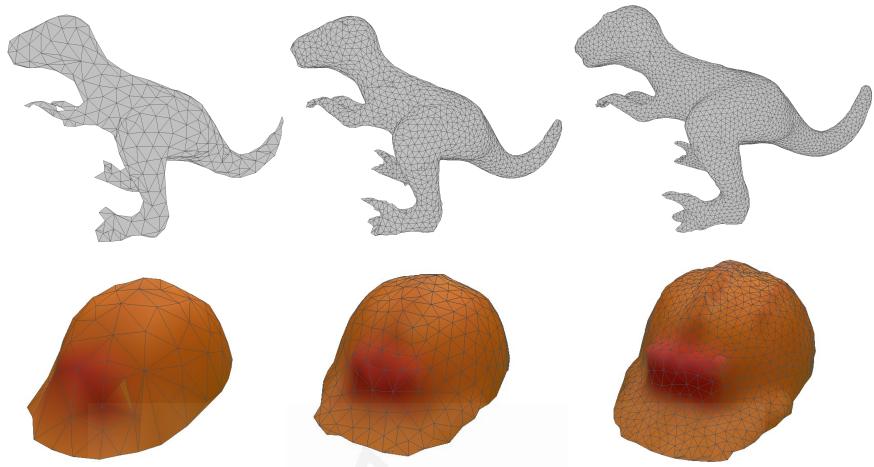


**Figure 2.32:** The proposed method for 3D reconstruction was also applied to 3D models of people and a foot of a person. Top: 3D model of a person. Bottom: digitized foot.

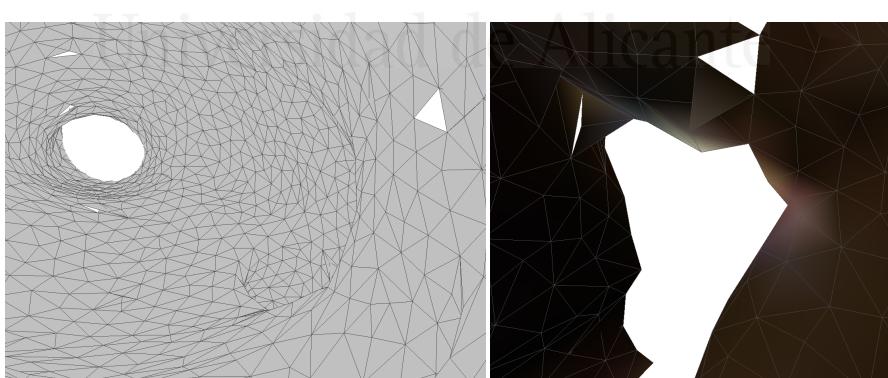
rons. Despite this fact, the proposed method is valid for many computer vision applications. 3D triangular faces are used in many 3D descriptors which are often used in object and scene recognition applications. This will be shown in Section 5.3, where we successfully used these 3D surfaces patches to recognize objects in cluttered scenes.



**Figure 2.33:** The proposed method for 3D reconstruction was also applied to partial 3D views of scenes. Left: Noisy point clouds captured using the Kinect sensor. Right: 3D reconstruction using the proposed method.



**Figure 2.34:** Different 3D reconstructions of a the builder helmet model using various network sizes. Left: 3D reconstruction using 250 neurons and 200 input patterns. Middle: 3D reconstruction using 1000 neurons and 500 input patterns. Right: 3D reconstruction using 2500 neurons and 1000 input patterns.



**Figure 2.35:** Small gaps produced by our extended GNG method for 3D surface reconstruction.

### 2.3.8 Conclusions

We have presented a novel computing method to create 3D models from unorganized raw noisy 3D data. Previous knowledge about the sensor is not necessary. It has been demonstrated how Growing Self-Organizing Maps (GSOM) are capable to represent noisy 3D data distributions. Different GSOM approaches have been deeply studied and compared, showing advantages and disadvantages for 3D data captured from different sensors.

Moreover, growing self-organizing approaches have been considered as the best methods to solve this problem as they present some beneficial attributes as: flexibility, rapid adaptation, topology preservation, noise removal, dimensionality reduction, etc. To demonstrate the validity of our proposal we tested our method with several models and performed a study of the network parameterization, calculating the quality of representation and also comparing results with other methods like Voxel Grid. We also demonstrated that the proposed method obtains better adaptation to the input space than other filtering methods like Voxel Grid, obtaining lower adaptation MSE on simulated scenes and CAD models.

We have also modified the original GNG method to represent 3D data sequences, which accelerates the learning algorithm and allows the architecture to work faster. Moreover, we have also improved original method considering colour information. Neurons' internal structure was modified adding colour components to their weights. Due to this modification the algorithm is able to adapt its structure to the input space topology during the learning step and also to learn and store colour from the observation. This eliminates the necessity to add post-processing steps to add colour information to the final representation.

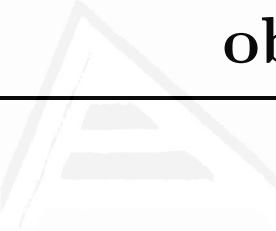
Finally, the GNG algorithm was modified considering the creation of triangular faces during the learning stage. In contrast with existing methods, our extension allowed to create complete triangular meshes with colour information during the learning stage, not requiring any post-processing steps to close gaps and holes. The method was validated with several models ranging from scanned objects to real-world scenes.

---

## Chapter 3

# Improving keypoint detection from noisy 3D observations

---



In this chapter we propose the use of the Growing Neural Gas (GNG) network, presented in Chapter 2, for filtering and down-sampling point clouds. These reduced representation were used as input data for further keypoint detector and feature extraction steps. Section 3.1 introduces the way that humans are able to identify and recognize scenes and how computers nowadays are able to simulate that behaviour using computer vision algorithms. Furthermore, classical filtering techniques are revised. Section 3.2 presents state-of-the-art keypoint detectors and feature descriptors. These techniques were used to validate the proposed method. Moreover, since our extended colour version of the GNG is able to learn color space of the input data, colour information was also used in some descriptors demonstrating the capabilities of the GNG representation. Section 3.3 shows the workflow of the proposed method and validates it performing different experiments on indoor scenes. Finally, Section 3.4 presents the main conclusions from this chapter.

### 3.1 Introduction

Historically, humans have the ability to recognize an environment they visited before based on the 3D model they unconsciously build in their heads based on the different perspectives of the scene. This 3D model is built with some extra information so that humans can extract relevant features [Treisman and Gelade, 1980] that will help in future experiences to recognize the environment and even objects in the scene. This way of learning has been transferred to mobile robotics field over the years. So, most current approaches in scene understanding and visual recognition are based on the same principle: keypoint detection (high interest areas) and feature extraction (description) on the perceived environment. Over the years most efforts in this area have been made towards feature extraction and keypoint detection on information obtained by traditional image sensors [Szummer and Picard, 1998, Tamimi et al., 2006], existing only few works in feature-based approaches that deal with 3D sensors as input devices. However, in recent years, the number of research papers concerned with 3D data processing has increased considerably due to the emergence of cheap 3D sensors capable of providing a real time data stream and therefore enabling feature-based computation of three dimensional environment properties like curvature, getting closer to human learning procedures.

Devices introduced in Chapter 2 as the Kinect device, the time-of-flight camera SR4000 or the 3D laser, are examples of these devices. Besides, providing 3D information, some of these devices like the Kinect sensor also provides color information of the observed scene. However, using 3D information in order to perform visual recognition and scene understanding is not an easy task. The data provided by these devices is often noisy and therefore classical keypoint detection approaches extended from 2D to 3D space do not work correctly. The same occurs to 3D methods traditionally applied on synthetic and noise-free data. Applying these methods to partial views that contains noisy data and outliers produced incorrect keypoint detection and hence computed features did not contain precise descriptions. Consequently, removing as much noise as possible is needed in order to perform an effective keypoint detection.

Classical filtering techniques like median or mean have been widely used to filter noisy point clouds [Nuchter et al., 2004, Kobbelt and Botsch, 2004] obtained from 3D sensors like the ones previously mentioned. The median filter is one of the simplest and wide-spread filters that has been applied. It is efficient and simple to implement but can remove noise only if the noisy pixels occupy less than one half of the neighbourhood area. Moreover, it removes noise but at the expense of smoothing corners and edges of the input data.

A filtering technique frequently used in point cloud noise removal is the Voxel Grid (VG) method, which was previously introduced in Chapter 2. Moreover, similar to Voxel Grid, another technique exists which is known as Uniform Sampling (US). As in VG, a grid of cubes is built and for each cube the centroid of the points is computed. This new point is the representative point for the cube. For the US technique, another step is added to the algorithm. The point of the original point cloud nearest to the computed centroid of the voxel is set as the final representative point for that cube. Therefore, points generated by the US method belong to the original input space while VG introduce new ones and therefore may have noisy results caused by the approximation process.

Furthermore, other noise removal techniques from image processing [Tomasi and Manduchi, 1998] have been used also on point clouds: the Bilateral filtering technique. [Wasza et al., 2011] applied it on depth maps obtained from 3D sensors, allowing noise removal and considering corners and edges. This is achieved by using Gaussian functions and range kernels to smooth data in a non-uniform way. However, Bilateral filtering is not able to deal with outliers in the input point cloud and it does not also produce a reduced representation, maintaining the same number of points in the output representation.

In this work we focus on the processing of 3D information provided by the Kinect sensor. Experimental results show that the random error of depth measurements provided by the Kinect sensors increases with larger distances to the sensor, and ranges from a few millimeters up to about 4 cm at the maximum range of the sensor. More information about the accuracy and precision of the Kinect device can be found in [Khoshelham

and Elberink, 2012].

In this chapter we propose the use of the GNG-based network, presented in Chapter 2, to filter and downsampling input point clouds that will serve as an input data for further keypoint detector and feature extraction steps. Noise removal, topology preservation, and input space adaptation properties of the GNG method help to create more robust and compact representations. The proposal was validated against the state-of-the-art 3D keypoint detectors. Moreover, since our extended color version of the GNG is able to learn color space of the input data, color information was also used in some descriptors demonstrating the capabilities of the GNG representation. Finally, the GNG method was validated in a 3D scene registration application.

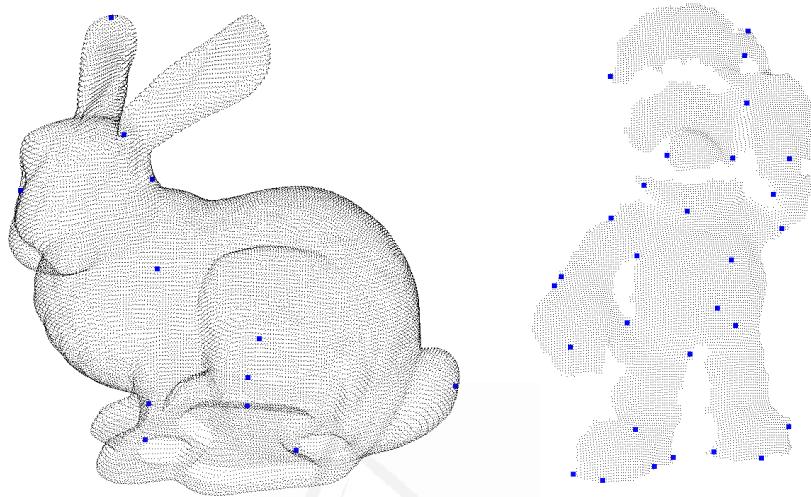
## 3.2 Keypoint detectors/descriptors

In this section, we present the state-of-the-art 3D keypoint detectors used to test and measure the improvement achieved using GNG method to filter and downsample the input data. In addition, we describe the main 3D descriptors and feature extraction methods that we used in our experiments. Finally, a method for feature matching is introduced.

### 3.2.1 Keypoint detectors

SIFT (Scale Invariant Feature Transform) [Lowe, 2004] detector performs a local pixel appearance analysis at different scales. SIFT features are designed to be invariant to image scale and rotation. It has been traditionally used in 2D image but it has been extended to 3D space. 3D implementation of SIFT differs from original in the use of depth or curvature as the intensity value. SIFT detector uses neighbourhood at each point within a fixed-radius assigning its intensity as the Gaussian weighted sum of the neighbours' intensity values in order to archive the 4-dimensional difference of Gaussians (DoG) scale space. Then it detects local maxima in these 4D DoG scale space when the point value is greater than all its neighbours. As new 3D sensors also provide RGB information, original SIFT algorithm has also been implemented on 3D data, using the colour

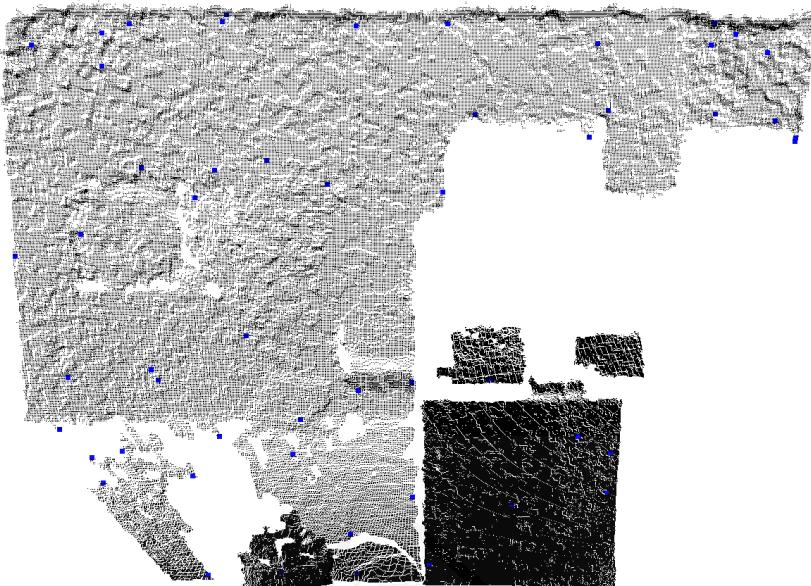
components as the intensity value.



**Figure 3.1:** Keypoints are detected based on the normal gradients i.e. curvature SIFT keypoint estimation. Detected keypoints are marked using blue dots. (Left) Complete Stanford bunny model. (Right) 3D partial view of Mario Bros model

Some examples of keypoint detection using the SIFT algorithm are presented in Figure 3.1. For the bunny model, as it is a noise-free model, detected keypoints are robust while for the partial 3D view of the mario model detected keypoints seems less stable, based on that some of them are detected in noisy areas and in the border of the view. In Figure 3.2, a similar behaviour was reported using a partial 3D scan of a scene. Some detected keypoints were located in flat areas of the wall and these should not be considered as keypoints.

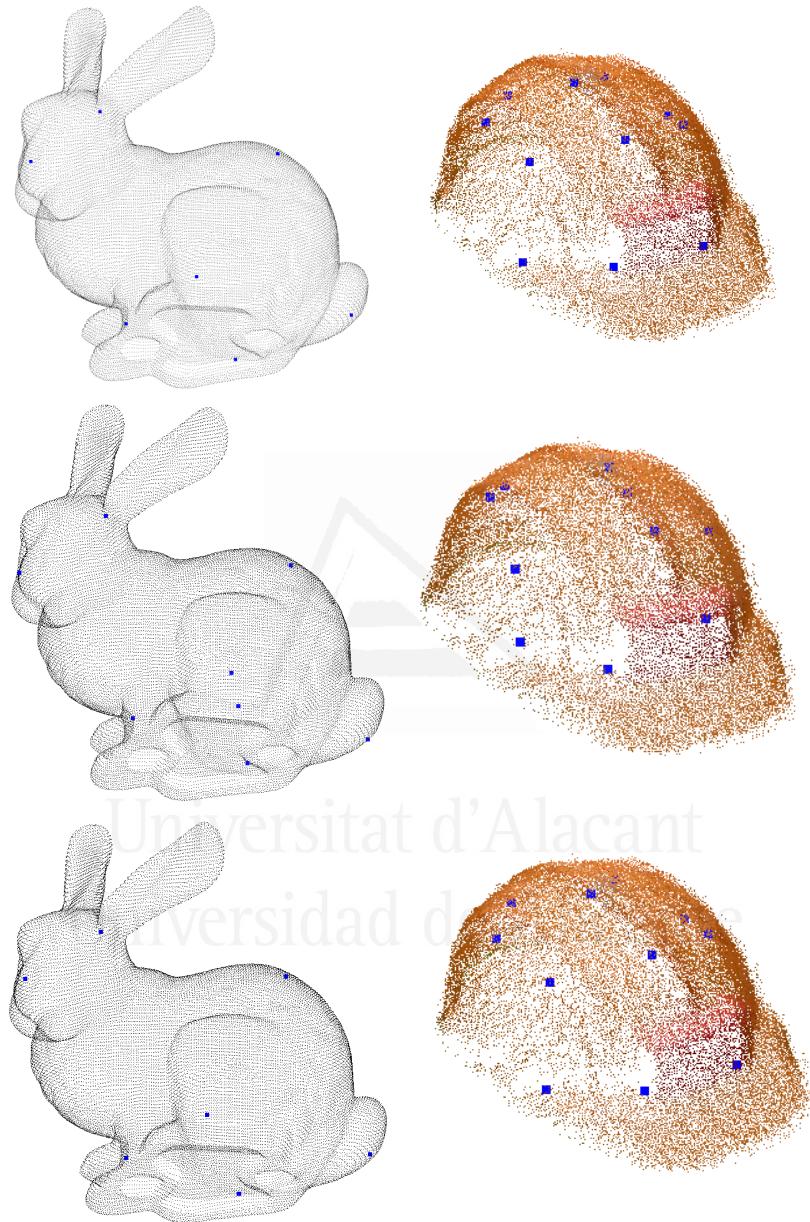
Other tested keypoint detectors are based on a classical Harris 2D keypoint detector. In [Mikolajczyk and Schmid, 2002] a refined Harris detector is presented in order to detect keypoints invariable to affine transformations. 3D implementations of these Harris detectors [Sipiran and Bustos, 2011] use surface normals of 3D points instead of 2D gradient images. Harris detector and its variants, extended from 2D keypoint detectors, have been tested using the proposed method in Section 3.3. All Harris variants have in common covariance matrix computation, but each variant makes



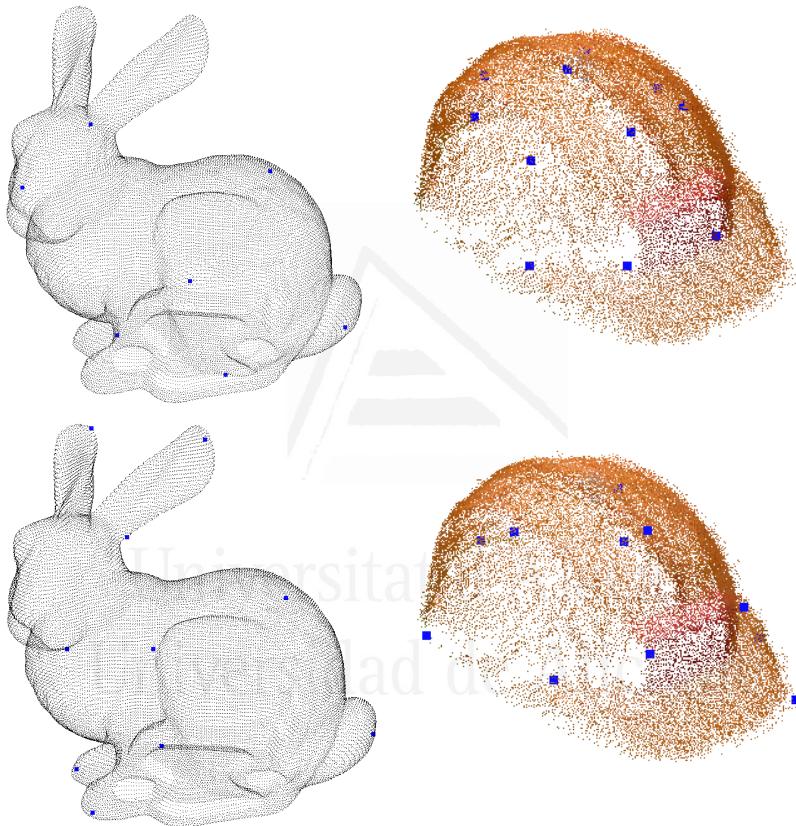
**Figure 3.2:** Scene captured using the Kinect sensor. Keypoints are detected based on the normal gradients i.e. curvature SIFT keypoint estimation. Detected keypoints are marked using blue dots.

a different evaluation of the trace and the determinant of the covariance matrix. Noble's variant corners detection algorithm [Noble, 1988] evaluates the ratio between the determinant and the trace of the covariance matrix. Tomasi's variant [Shi and Tomasi, 1994] performs eigenvalue decomposition over the covariance matrix using the smallest eigenvalue as keypoint score. Lowe's variant performs in a similar way than Noble's one but evaluating the ratio between the determinant and the squared trace of the covariance matrix. These small changes in the evaluation of the covariance matrix generate different keypoint detection.

Figures 3.3 and 3.4 show detected keypoints using above presented 3D Harris algorithm and its variants. Keypoints were detected in areas that have corners or peaks. Furthermore, different variants detected similar points but not exactly the same points, therefore producing slightly different results in further processing steps.

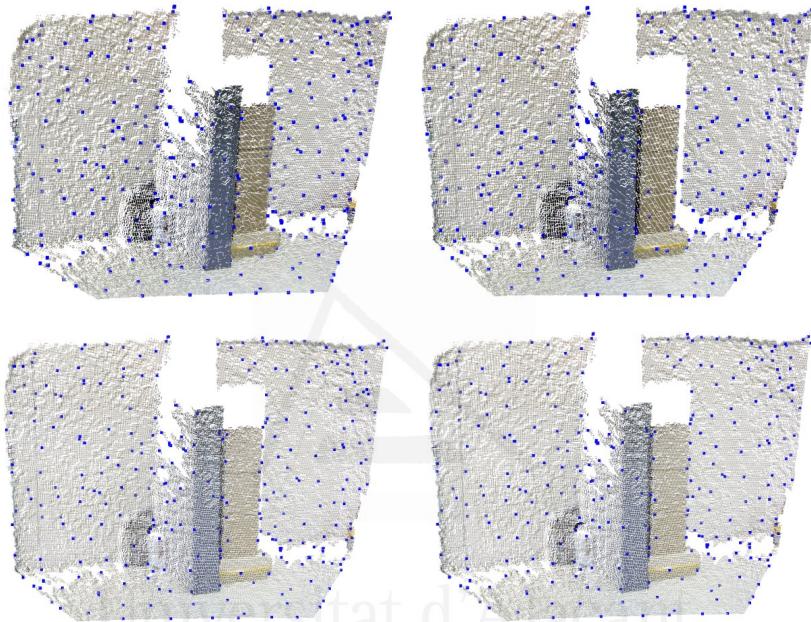


**Figure 3.3:** Keypoints are detected using the 3D Harris detector and its variants. Detected keypoints are marked using blue dots. Two different models have been tested: on the left side the Stanford Bunny model and on the right side a builder helmet captured using the Kinect sensor. From top to bottom: Harris3D, Tomasi3D and Noble3D.



**Figure 3.4:** Same models than in Figure 3.3. From top to bottom: Lowe3D and Curvature3D.

Figure 3.5 shows detected keypoints using Harris3D and its variants on a captured scene. Detected keypoints are noisy and unstable as Harris3D-based detectors do not deal with the presence of noise and captured scene has high levels of noise.



**Figure 3.5:** Keypoints are detected using the 3D Harris detector and its variants. Detected keypoints are marked using blue dots. A partial view of the complete scene has been tested. From left to right and from top to bottom we used Harris3D, Tomasi3D, Noble3D and Lowe3D.

Another 3D detector included in the study is based on the Intrinsic Shape Signature (ISS) originally presented in [Zhong, 2009]. Tested implementation is focused on the keypoint detection step of the algorithm, so signature is not computed. As the descriptor reference frame is computed using normal information and Eigenvalue decomposition, this data is also used to detect keypoints. A saliency measure value used by the ISS detector is calculated using Eigenvalue Decomposition (EVD) of the scattered matrix computed using the nearest neighbours for each point. Once eigenvalues are computed, only points which have eigenvalues differences (the

biggest vs the second biggest and the second biggest vs the third biggest) greater than given thresholds are selected as keypoints. More details about the algorithm are depicted below:

For each point  $p$  a weight is computed that is inversely related to the number of points in its spherical neighborhood of radius  $R_{density}$ . This weight is used to compensate for uneven sampling of the 3D points, so that points at sparsely sampled regions contribute more than points at densely sampled regions. Then, a weighted scatter matrix is computed for each point  $p$  using all neighbour points within a distance  $R_{frame}$ . Computed eigen values are finally used to classify a point  $p$  as keypoint or not.  $p$  is considered a valid keypoint if values differences are greater than given thresholds  $t_1$  and  $t_2$ , which are established by the user.

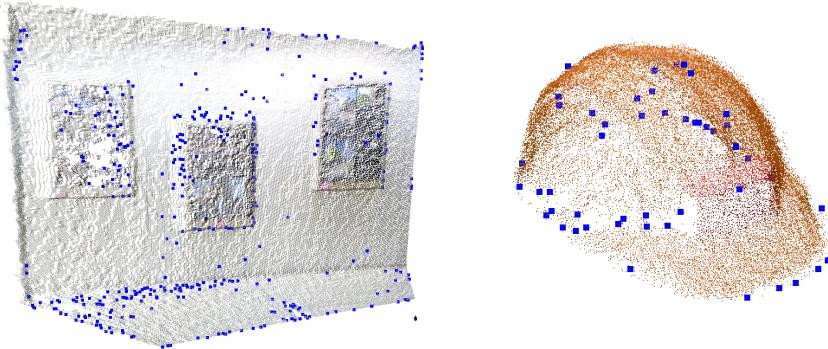
Figure 3.6 shows detected keypoints using the ISS algorithm on different 3D observations. On the left side, a partial view of a scene is shown. Most keypoints are detected at the joint between the wall and the floor since there is a high variation in terms of normal information. On the right side are shown the detected keypoints over a point cloud representing a builder helmet. In this case, most detected keypoints are located at corners of the helmet and in the boundary of the point cloud. This keypoint detector fails detecting keypoints around the boundary, producing therefore keypoints that are not high interest points. This is caused by the lack of normals around a point located at the boundary. Some preprocessing steps could be added to this method in order to improve boundary detection.

We presented how different keypoint detection algorithms computed high interest points in different areas, although sometimes different methods detected keypoints in similar positions.

### 3.2.2 Feature Descriptors

Once keypoints have been detected, the next step is to extract descriptors on these points. Feature descriptors allow to extract features that help performing matching between different observations.

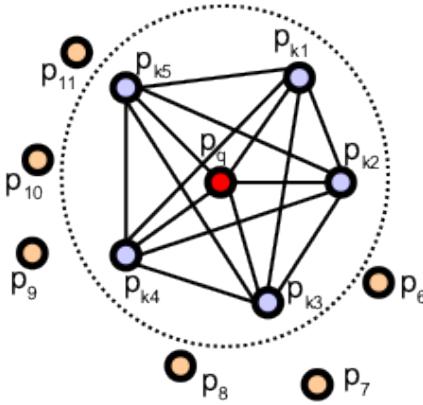
In the last few years some descriptors that take advantage of 3D information have been presented. [Rusu et al., 2008a] presents a pure 3D descriptor called Point Feature Histograms (PFH). The goal of the PFH



**Figure 3.6:** Detected keypoints using the ISS algorithm on different 3D observations. (Left) partial view of a scene. (Right) complete model of a builder helmet. Both were captured using the Kinect sensor.

formulation is to encode the point’s k-neighbourhood geometrical properties by generalizing the mean curvature around the point using a multi-dimensional hyperspace. This highly dimensional hyperspace provides an informative signature for the feature representation, which is invariant to the 6D pose of the underlying surface, and copes very well with different sampling densities or noise levels in the neighbourhood. A PFH representation is based on the relationships between the points in the k-neighbourhood and their estimated surface normals. Briefly, it attempts to capture as best as possible the sampled surface variations by considering all the interactions between the directions of the estimated normals. The final PFH descriptor is computed as a histogram of relationships between all pairs of points in the neighborhood, and thus it has a computational complexity of  $O(nk^2)$  where  $n$  is the number of points and  $k$  is the number of neighbours for each point  $p$ . Figure 3.7 shows an influence region diagram of the PFH computation for a query point  $p_q$ .

The relative difference between two points  $p_i$  and  $p_j$  and their associated normals  $n_i$  and  $n_j$  is used to define a fixed coordinate frame at one of the points. The coordinate frame  $uvw$  is defined as:



**Figure 3.7:** Diagram of the PFH computation for a query point  $p_q$ . The query point is marked red and it is in the middle of the diagram. Neighbour points  $k$  are selected within a radius  $r$ , which is marked using a dotted line.

$$\begin{aligned}
 u &= n_i \\
 v &= u \times \frac{p_i - p_j}{\|p_i - p_j\|_2} \\
 w &= u \times v
 \end{aligned}
 \tag{3.1}$$

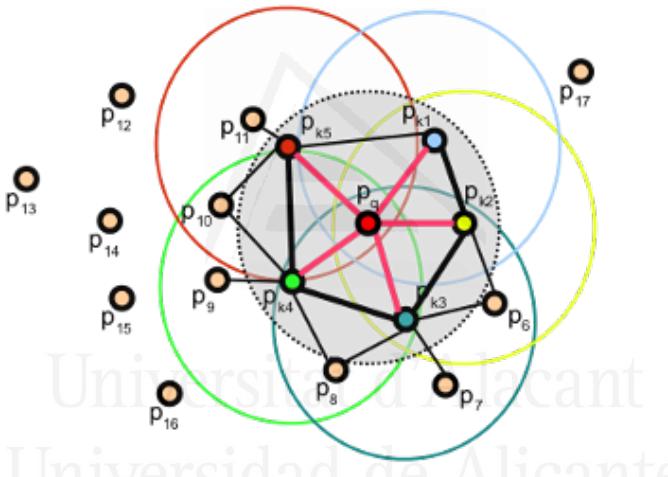
A signature comprised of 4 values is computed using the above  $uvw$  frame and the difference between the two normals  $n_i$  and  $n_j$  :

$$\begin{aligned}
 \alpha &= v \cdot n_j \\
 \phi &= u \cdot \frac{p_i - p_j}{d} \\
 \theta &= \arctan(w \cdot n_j, u \cdot n_j) \\
 d &= \|p_i - p_j\|
 \end{aligned}
 \tag{3.2}$$

where  $d$  is the Euclidean distance between the two points  $p_i$  and  $p_j$ . To create the final PFH representation for the query point, the set of all quadruplets is binned into a histogram. The binning process divides each

features's value range into subdivisions, and counts the number of occurrences in each subinterval.

A simplification of the descriptor described above was also presented. It was coined as Fast Point Feature Histograms (FPFH) [Rusu et al., 2009] and is based on a histogram of the differences of angles between the normals of the neighbour points and the query point. This method is a fast refinement of the Point Feature Histogram (PFH) that computes its own normal directions and a subset of pair point normal differences using only keypoint neighbourhood. It reduces the computational complexity of the PFH algorithm from  $O(nk^2)$  to  $O(nk)$  where  $k$  is the number of neighbours.

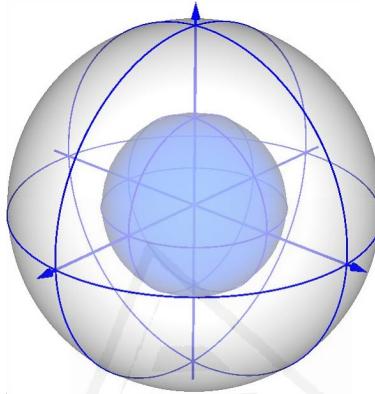


**Figure 3.8:** Diagram of the FPFH computation for a query point  $p_q$ . The query point is marked in red and it is in the middle of the diagram. Neighbour points  $k$  are selected within a radius  $r$ , which is marked using a dotted line.

So, in the first step the FPFH algorithm performs for each query point  $p_q$  the computation of the signature presented above. In a second step, for each  $k$  neighbour the signature is computed using the same approach that in the PFH algorithm. This is visually depicted in Figure 3.7. Figures 3.7 and 3.8 were extracted from the original works [Rusu et al., 2008a, Rusu et al., 2009].

In addition, we used another descriptor called Signature of Histograms of Orientations (SHOT) [Tombari et al., 2010b], which is based on the computation of a repeatable local reference frame using the eigenvalue

decomposition around an input point. Given this reference frame, a spherical grid centered at the point divides the neighbourhood so that in each grid bin a weighted histogram of normals is obtained. The descriptor concatenates all such histograms into the final signature. The structure of the signature uses an isotropic spherical grid that encompasses partitions along the radial, azimuth and elevation axes, as sketched in Figure 3.9.



**Figure 3.9:** Spherical grid used by the SHOT descriptor to calculate normals differences and therefore to build a weighted histogram of normals.

This descriptor was also extended in [Tombari and Salti, 2011] using color information (CSHOT). It basically adds a triplet containing RGB information to the original implementation. Therefore, a new histogram is created using the absolute differences between the RGB triplets instead the dot product that was used in the original method to calculate the differences between normals.

All presented 3D descriptors will be deeply tested in Section 3.3.

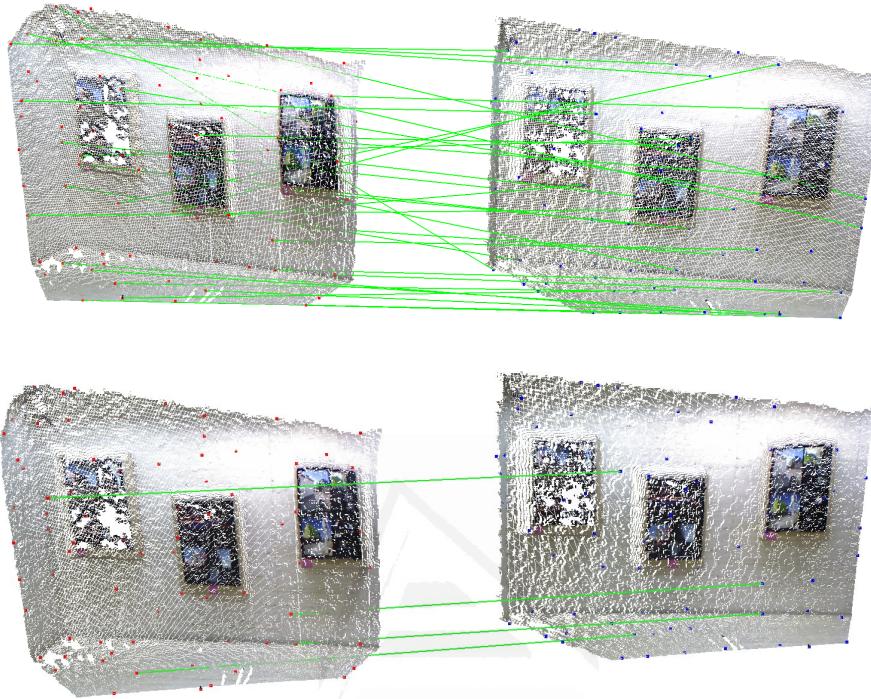
### 3.2.3 Feature matching

Since we want to test our proposal in a feature matching process, once features have been computed over detected keypoints, it is necessary to establish correspondences between features. Correspondences between features or feature matching methods are commonly based on the euclidean distances between feature descriptors. Moreover, commonly a rejection step is included in order to remove false positives correspondences. One of

the most used method to validate the estimated transformation between pairs of matched correspondences is based on the RANSAC (RANdom SAmple Consensus) algorithm [Fischler and Bolles, 1981]. It is an iterative method that estimates the parameters of a mathematical model from a set of observed data which contains outliers. In our case, we used this method to search a 3D transformation (our model) which best explain the data (matches between 3D features). At each iteration of the algorithm, a subset of data elements (matches) is randomly selected. These elements are considered as inliers and a model (3D transformation) is fitted to those elements. The rest of the data is then tested against the fitted model and included as inliers if its error is below a given threshold. If the estimated model is reasonably good (its error is low enough and it has enough matches), it is considered as a good solution. This process is repeated a number of iterations and the best solution found at that moment is returned. Other more simple rejection methods based on euclidean or mean distance and surface normal have not been considered as they produce poor results. Figure 3.10 shows found correspondences between two scenes without performing rejection step (Top) and performing rejection based on RANSAC method (Bottom). Thanks to the RANSAC based rejection step, most false correspondences are removed and then a better 6DoF transformation is obtained (less error-prone). In this work the number of iterations for the RANSAC algorithm was empirically established to 100 based on the work presented in [Civera et al., 2010]. This work presented a probabilistic model to estimate the number of iterations. All the experiments were performed using the same number of iteration so as the comparison between different keypoint detectors and feature descriptors was fair.

### 3.3 Improving Keypoint detection

We proposed the use of the GNG-based network presented in Chapter 2 for filtering and downsampling input point clouds that served as input for further keypoint detector and feature extraction steps. Filtered point cloud produced by the GNG method is used as an input of many state-

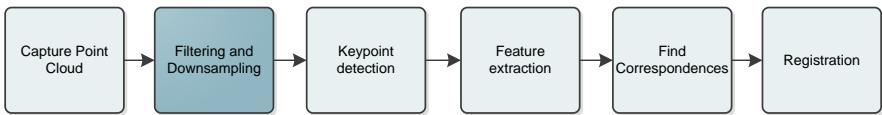


**Figure 3.10:** Correspondences matching between two different scenes are computed. Top: correspondences without performing rejection step. Bottom: After applying RANSAC based rejection step false positives are removed.

of-the-art 3D keypoint detectors in order to demonstrate how the filtered and reduced point cloud improves keypoint detection and hence feature extraction in 3D registration methods.

Moreover, GNG representation improves the precision of the matching process. Precision defined as the percentage of detected matches that are correct (i.e. inliers) is an important performance measure of a point matching algorithm as it directly influences the efficiency of the transformation. For instance, if RANSAC is used, the number of required iterations in the final alignment process is reduced as the precision increases. That is important for applications where high recognition rates are needed. It is highly desirable to have a correspondence matching process that provides high levels of precision

The proposed method is compared with various state-of-the-art filtering techniques, the Voxel Grid and the Uniform Sampling methods. Results presented in Section 3.3.1 show how the proposed method overperforms both techniques, producing more accurate transformations between different views. Figure 3.11 shows a general system overview of the steps involved in most 3D computer vision problems and how our proposal is integrated in that processing pipeline.



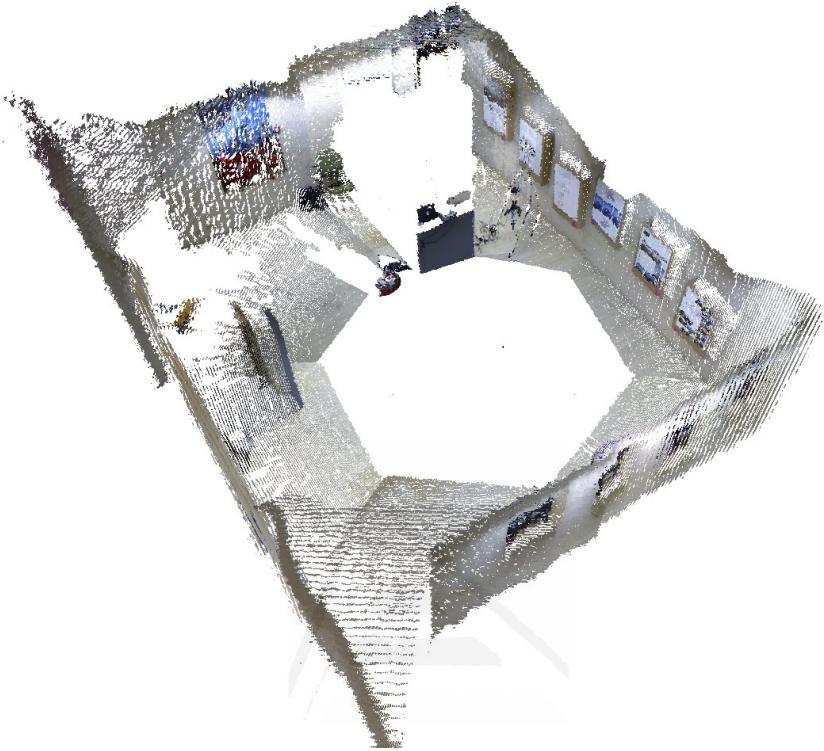
**Figure 3.11:** General scheme of most 3D computer vision applications. Proposed GNG-based algorithm presented in Chapter 2 is used to obtain a robust representation of the input space (highlighted step). This representation will improve following steps as the keypoint detection.

### 3.3.1 Experiments

We performed different experiments on indoor scenes to evaluate the effectiveness and robustness of the proposed method. The proposed method is applied to 3D scene registration to show how keypoint detection methods are improved obtaining more accurate transformations. 3D scene registration is performed on a dataset comprised of 90 overlapped partial views of a room. Partial views are rotated 4 degrees in order to cover 360 degrees of the scene. Partial views were captured using the Kinect device mounted in a robotic arm with the aim of knowing the ground truth transformation. Experiments implementation, 3D data management (data structures) and their visualization have been developed using the PointCloud (PCL) library.

A complete model of the indoor scene is presented in Figure 3.12. The registration of partial views in a completed 3D model is performed using the proposed method.

We used keypoint detectors introduced in Section 3.2.1 in order to test noise reduction capabilities of the GNG method for the 3D scene



**Figure 3.12:** Registered views of the indoor scene.

registration problem. We used a different number of neurons and patterns to obtain a downsampled and filtered representation of the input space. Various configurations have been tested, ranging from 5,000 to 20,000 neurons and 250 to 2,000 patterns per iteration. In addition, the same configurations were tested with the Voxel Grid and Uniform Sampling methods to establish a comparison with the proposed method.

Root Mean Square (RMS) deviation measure [Jenkinson, 2003] was used to measure error between the estimated transformation and the ground truth. It is a frequently used measure of the differences between values predicted by a model or an estimator and the values actually observed (ground truth). It calculates the average error between two affine transformations. RMS is computed as follows:

$$RMS = \sqrt{\frac{\sum_{i=1}^N (p'_x - q'_x)^2 + (p'_y - q'_y)^2 + (p'_z - q'_z)^2}{N}} \quad (3.3)$$

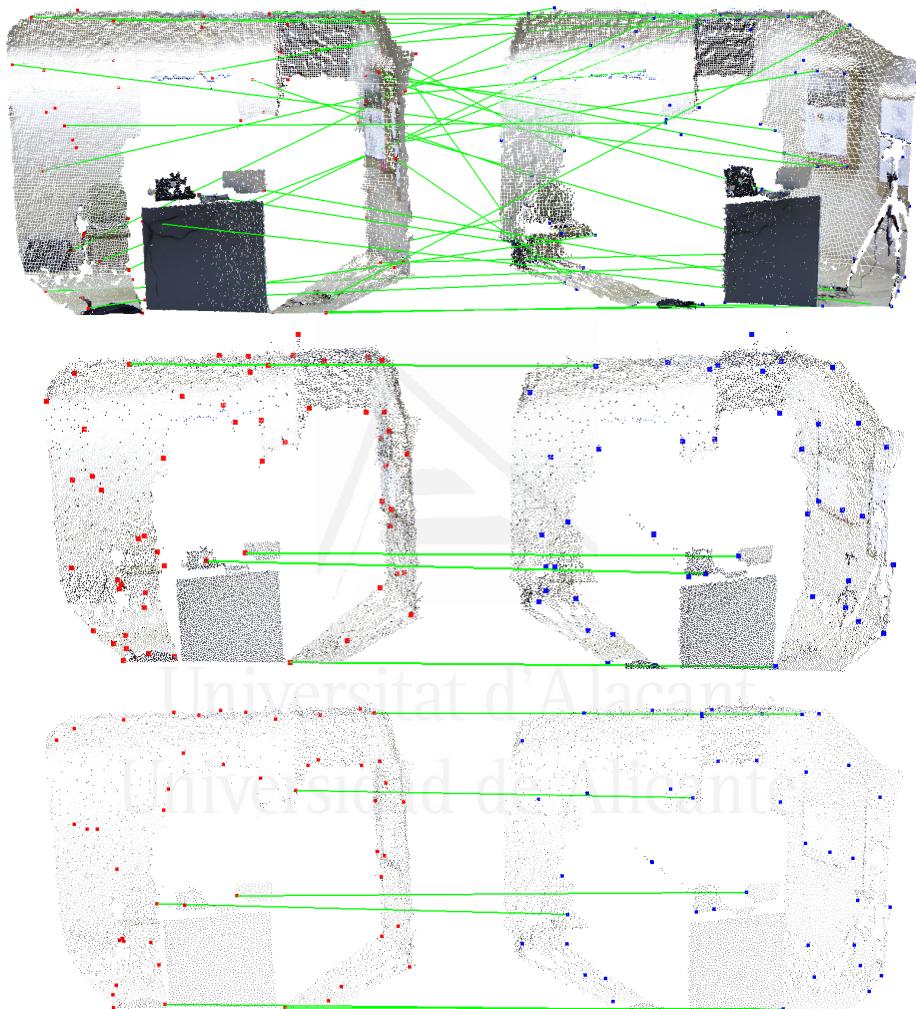
where  $N$  is the number of points,  $p'_x, p'_y$  and  $p'_z$  are the transformed point coordinates using the estimated transformation and  $q'_x, q'_y$  and  $q'_z$  are the transformed point coordinates using the ground truth transformation.

Figure 3.13 shows correspondence matching between two different views of the scene using different input point clouds. On the top, the process has as an input a raw point cloud captured from the sensor. While in the middle and the bottom of the figure correspondence matching is performed using as an input reduced representations produced by the GNG method. Two different configurations are shown: 5,000 and 10,000 neurons with different number of input patterns: 250 and 500 respectively. Correspondences matching applied directly to raw point clouds produces a large number of false positives even with the RANSAC-based rejection step while reduced representations produce more stable keypoint detection and finally better correspondences between two different views. This fact was quantitatively demonstrated computing RMS transformation errors.

Figure 3.14 shows how correspondence matching is calculated over filtered point clouds using the proposed method. Red and blue points represent keypoints detected on filtered point clouds using the GNG method. Green lines are correct matchings between computed descriptors of two different views of the same scene.

Experiments were performed using different search radius for keypoint detection and feature extraction methods. Search radius influences directly on the size of the extracted features, making methods more robust against occlusions. A balance between large and small values must be found depending on the size of the objects in the scene and the size of the features we want to extract. In experiments, we empirically established that the best estimated transformations were found using keypoint detector search radius equals to 0.1 and 0.05 meters, and feature extractor search radius equals to 0.2 meters.

Table 3.1 shows how using GNG output representation as input cloud for the registration step, lower RMS errors are obtained in most detector-



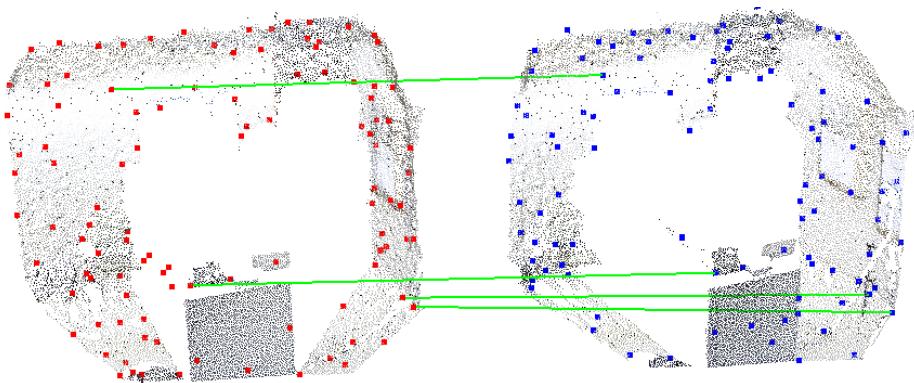
**Figure 3.13:** Correspondences matching computed on different input data. Top: raw point clouds. Middle: reduced representation using the GNG (20,000 neurons). Bottom: reduced representation using the GNG (10,000 neurons)

	SIFT3D				Harris3D			
	FPPFH	CSHOT	PFH	PFHRGB	FPPFH	CSHOT	PFH	PFHRGB
<b>GNG 5000p</b>	0.168	0.052	0.185	0.106	0.151	0.085	0.109	0.054
<b>GNG 10000p</b>	0.092	0.037	0.367	0.029	0.114	0.046	0.177	0.047
<b>GNG 20000p</b>	0.231	0.019	0.255	0.057	0.079	0.052	0.153	0.042
<b>VG 5000p</b>	0.239	0.063	0.540	0.080	0.404	0.038	0.305	0.033
<b>VG 10000p</b>	0.073	0.070	0.171	0.050	0.088	0.033	0.097	0.058
<b>VG 20000p</b>	0.139	0.037	0.375	0.027	0.180	0.069	0.469	0.043
<b>US 5000p</b>	0.558	0.089	0.202	0.093	0.156	0.090	0.127	0.072
<b>US 10000p</b>	0.065	0.056	0.113	0.062	0.074	0.036	0.215	0.053
<b>US 20000p</b>	0.146	0.036	0.239	0.019	0.151	0.038	0.152	0.082
<b>Raw PCL</b>	0.103	0.039	0.082	0.041	0.128	0.066	0.144	0.093

	Tomasi3D				Noble3D			
	FPPFH	CSHOT	PFH	PFHRGB	FPPFH	CSHOT	PFH	PFHRGB
<b>GNG 5000p</b>	0.049	0.043	0.189	0.112	0.143	0.098	0.273	0.188
<b>GNG 10000p</b>	0.054	0.023	0.308	0.121	0.186	0.052	0.127	0.117
<b>GNG 20000p</b>	0.270	0.063	0.319	0.066	0.199	0.063	0.239	0.064
<b>VG 5000p</b>	0.383	0.022	0.067	0.066	0.387	0.085	0.073	0.096
<b>VG 10000p</b>	0.127	0.047	0.258	0.049	0.188	0.059	0.244	0.082
<b>VG 20000p</b>	0.148	0.046	0.123	0.078	0.289	0.048	0.188	0.079
<b>US 5000p</b>	0.076	0.078	0.262	0.055	0.137	0.039	0.278	0.062
<b>US 10000p</b>	0.393	0.063	0.106	0.039	0.202	0.068	0.122	0.051
<b>US 20000p</b>	0.326	0.054	0.142	0.079	0.045	0.049	0.210	0.065
<b>Raw PCL</b>	0.659	0.049	0.765	0.062	0.114	0.060	0.099	0.077

	Lowe3D				Curvature3D			
	FPPFH	CSHOT	PFH	PFHRGB	FPPFH	CSHOT	PFH	PFHRGB
<b>GNG 5000p</b>	0.143	0.065	0.273	0.188	0.099	0.048	0.262	0.139
<b>GNG 10000p</b>	0.117	0.052	0.120	0.135	0.228	0.032	0.151	0.071
<b>GNG 20000p</b>	0.076	0.062	0.239	0.016	0.113	0.022	0.123	0.053
<b>VG 5000p</b>	0.387	0.040	0.073	0.077	0.228	0.033	0.244	0.068
<b>VG 10000p</b>	0.188	0.059	0.244	0.082	0.103	0.042	0.101	0.083
<b>VG 20000p</b>	0.093	0.107	0.167	0.076	0.093	0.050	0.057	0.023
<b>US 5000p</b>	0.137	0.096	0.278	0.062	0.151	0.055	0.369	0.053
<b>US 10000p</b>	0.202	0.093	0.169	0.066	0.200	0.076	0.307	0.050
<b>US 20000p</b>	0.365	0.036	0.097	0.031	0.151	0.052	0.123	0.037
<b>Raw PCL</b>	0.203	0.067	0.240	0.027	-	-	-	-

**Table 3.1:** RMS deviation error (meters) is obtained using different detector-descriptor combinations. Combinations are computed on the original point cloud (raw), and different filtered point clouds using Voxel Grid, Uniform Sampling and the proposed method. Keypoint detector search radius equals to 0.05 meters. Feature extractor search radius equals to 0.02 meters.



**Figure 3.14:** Registration example performed with the Lowe keypoint detector and the FPFH descriptor using as an input a GNG representation with 20,000 neurons.

descriptor combinations compared to other filtering techniques and also using raw point clouds. Lowe3D detector combined with the FPHRGB descriptor obtains lower RMS errors compared with Voxel Grid, Uniform Sampling and raw point clouds. RMS errors presented in Table 3.1 were computed using a search radius equal to 0.05 for keypoint detection as this configuration obtained most precise transformations. A more detailed study and all computed RMS errors using various search radius for key-point detectors and feature descriptors are presented in Appendix A.

Experiments showed that Lowe3D detector obtained lower RMS error for all descriptors. Tomasi3D and Noble3D detectors produced better results for the CSHOT descriptor and finally, Harris3D is also improved with FPFH and CSHOT descriptors. However, Voxel Grid filtered point clouds also obtains good transformations in a few detector-descriptor combinations as SIFT3D-CSHOT or Tomasi3D-FPFH since some keypoint detectors and descriptors have implicit capabilities to deal with noisy data. SIFT3D or Tomasi3D are examples of keypoint detectors that deal well with noisy or incomplete data. There are some blank values in Table 3.1 caused by some configurations that did not find any keypoint and therefore registration process was not able to continue with feature extraction step. ISS3D keypoint detector has not been included since it could not

find enough number of correct matchings and therefore the transformation between different point clouds could not be computed.

Finally, minimum, median, mean and maximum RMS error (with respect to different keypoint detectors) were computed (Table 3.2) for values presented in Table 3.1. These statistics results show how filtered point clouds using the GNG method generally improved the precision of the estimated transformation. Moreover, worst estimated transformations (maximum errors) were also slightly improved using the GNG compared to the other techniques.

	<b>Min</b>	<b>Median</b>	<b>Mean</b>	<b>Max</b>
<b>GNG 5000p</b>	0.126	0.126	0.134	0.273
<b>GNG 10000p</b>	0.023	0.115	0.117	0.367
<b>GNG 20000p</b>	<b>0.016</b>	0.071	0.120	0.319
<b>VG 5000p</b>	0.022	0.079	0.168	0.540
<b>VG 10000p</b>	0.033	0.083	0.108	0.258
<b>VG 20000p</b>	0.023	0.086	0.125	0.469
<b>US 5000p</b>	0.039	0.094	0.149	0.558
<b>US 10000p</b>	0.036	0.075	0.120	0.393
<b>US 20000p</b>	0.019	0.081	0.114	0.365
<b>Raw PCL</b>	0.027	0.088	0.156	0.765

**Table 3.2:** Mean, median, minimum and maximum RMS errors of the estimated transformations with respect to different keypoint detectors. These values were extracted from results presented in Table 3.1

One remarkable example showed in Table 3.1 that demonstrated capabilities of the proposed method is the combination of GNG filtered point clouds with the Lowe3D detector and PFHRGB feature descriptor. This combination obtained the most accurate transformation compared with other methods. For the same combination, the proposed method obtained less than 1.6 centimeters error whereas original point cloud produced almost 3 centimeters error in the best case.

### 3.4 Conclusions

In this chapter we have presented the capabilities of the GNG algorithm to improve keypoint detection step. The GNG, through filtering and

downsampling is able to deal with noisy 3D data captured using low cost sensors like the Kinect Device. The GNG network provides a 3D structure which has less information than the original 3D data, but keeping the 3D topology. It has been shown how state-of-the-art keypoint detection algorithms perform better on filtered point clouds using the proposed method. Moreover, the efficiency of the method is improved as less iterations are required to reject false matchings. Improving the precision of the method and the computing time.

The proposed method was validated in a 3D scene registration process, obtaining lower transformation errors in most detector-descriptor combinations that took as an input data the GNG representation. The most accurate transformations between different point clouds were obtained using the proposed method. Finally, it has also been quantitatively demonstrated how the GNG method overperforms other related filtering and downsampling techniques as VG and US. The computing time was also improved as point clouds were reduced and therefore the number of points to process.

Universitat d'Alacant  
Universidad de Alicante

---

## Chapter 4

# GPGPU parallel Implementations

---

In this chapter we describe the General-Purpose computation on Graphics Processing Units (GPGPU) paradigm and its ability to considerably accelerate algorithms traditionally executed on the CPU. In addition, we propose GPU-based implementations of the Self-Organizing Maps algorithms presented in Chapter 2 and a fast robust 3D local shape descriptor is also computed on the GPU. The parallel version of the algorithms were implemented using CUDA technology from NVIDIA. First, Section 4.1 introduces GPUs evolution in last years and revises existing computer vision implementations. Section 4.2 gives a briefly overview of modern GPU architectures. Next, Section 4.3 and 4.4 presents GPU-based implementations of the GNG and NG algorithms. These algorithms have been analysed and redesigned to be implemented on a modern GPU architecture. Several experiments are presented analysing the performance of the proposed implementations. Section 4.5 shows the GPU-based implementation of a 3D descriptor based on the calculation of 3D semi-local surface patches (local shape). Moreover, preprocessing steps for the computation of the descriptor are described as they were also implemented on the GPU pipeline. Finally, Section 4.6 presents the main conclusions from this chapter.

## 4.1 Introduction

In many cases computer applications present temporal constraints that make necessary to find mechanisms that accelerate their runtime. Traditionally, this problem has been addressed using parallel approaches as distributed computing (DC), Field Programmable Gate Arrays (FPGAs), System on Chips (SoCs), multi-core CPUs, Single Instruction Multiple Data (SIMD) instruction sets, parallel machines and other dedicated and in most cases expensive solutions.

In last years, GPUs have dramatically evolved providing high performance capabilities to existing computing solutions, ranging from desktop personal computers to large clusters. GPUs originally were used only for computer graphics, but nowadays they combine their computing power for hosting general-purpose parallel computations and for rendering computer graphics. This new paradigm was coined as General-Purpose computation on Graphics Processing Units (GPGPU) [Luebke et al., 2004].

Furthermore, GPU programming tools have also evolved providing developers easier ways to take advantage of the computer power of the GPU. Originally, GPU programming was done through shaders which had to be written in assembly language. With the constant rising of functionality provided by GPUs, different level programming languages were developed, such as High-Level Shading Language (HLSL) and NVIDIA's C for Graphics (Cg) [Mark et al., 2003] language. Another alternative was to directly use computer graphics tools, such as OpenGL or DirectX. Later, other high level languages emerged based on turning GPUs into stream processors, such as Brook [Buck et al., 2004] or PyGPU [Lejdfors and Ohlsson, 2006]. Finally, Computing Unified Device Arquitecture (CUDA) [NVIDIA, 2013] and Open Computing Language (OpenCL) [Stone et al., 2010] technologies were released by NVIDIA and Khronos Group respectively. Although they are considered the currently dominant GPGPU languages, recently new approaches have been proposed for programming GPUs based on pragma compiler directives (OpenACC<sup>1</sup>).

---

<sup>1</sup>OpenACC is a programming standard for parallel computing developed by Cray, CAPS, NVIDIA and PGI. The standard was designed to simplify parallel programming of heterogeneous CPU/GPU systems.

Current GPUs have a large number of processors used for general purpose computing. The GPU is specifically appropriated to solve computationally intensive problems expressed as data parallel computations [Hwu, 2011, Nickolls and Dally, 2010]. However, GPU implementations require the redesign of the algorithms focused and adapted to the architecture. In addition, GPU programming has different restrictions such as the necessity to maximize the occupancy on each processor in order to hide latencies produced by memory accesses, management of different threads running simultaneously, the proper use of the hierarchy of memories, and other considerations. Researchers have already successfully applied GPU computing to problems that were traditionally addressed by the CPU [Satish et al., 2009, Horn et al., 2007, Hwu, 2011].

The GPU have become really important in the High Performance Computing (HPC) field compared to other solutions due to some key factors. The first one is the emergence of new high level languages for programming GPUs which permitted most developers have access to the high computational power of the GPU even without having specific knowledge about computer graphics. Moreover, GPUs are considered massively parallel processors and provide a high computational power. Nowadays, GPUs have democratized HPC as GPUs are present in most computers.

In this thesis, different GPU implementations were developed. The first one was motivated by the learning stage of Self-Organizing Maps (SOMs). This step is the most computationally demanding part of these algorithms. The good point is that they are intrinsically parallel, and thus well-suited to be implemented on parallel architectures. Learning stages of the GNG and NG algorithms were implemented on GPUs using the CUDA technology. Moreover, in order to accomplish the acceleration of the neural network learning algorithm, the sequential algorithm was redesigned to fit the GPU architecture.

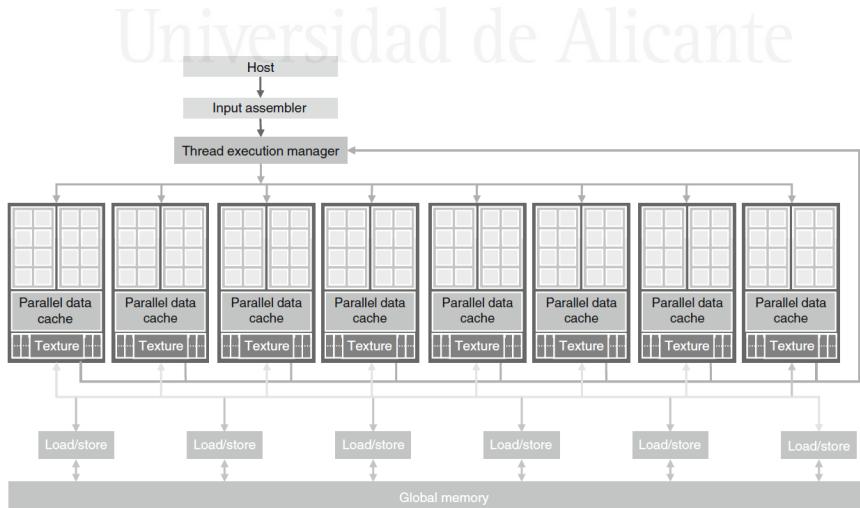
Furthermore, a feature descriptor able to represent complex 3D local shapes was implemented on the GPU in order to considerably accelerate its runtime performance.

The GPU implementations presented in this work are based on NVIDIA's CUDA architecture [NVIDIA, 2013], which is supported by most current

NVIDIA graphics chips. Supercomputers that currently lead the world top ranking combine the use of a large number of CPUs with a high number of CUDA enabled GPUs.

## 4.2 GPGPU architecture

A CUDA compatible GPU is organized in a set of multiprocessors as shown in Figure 4.1 [Kirk and Hwu, 2010]. These multiprocessors, denominated Streaming Multiprocessors (SMs), are highly parallel at thread level. However, the number of multiprocessors varies depending on the GPU generation. Each SM consists of a serie of Streaming Processors (SPs) that share the control logic and cache memory. Each of these SPs is able to launch in parallel a huge amount of threads. For instance, the GT400 chip family supports up to 1024 threads per SM, with 480 SPs distributed in 15 SMs. The GT400 chip is capable of performing a computing power of 1,5 teraflops, launching a total of 15,360 threads simultaneously. The current GPUs have up to 12 GBytes of DRAM, referenced in Figure 4.1 as global memory. This memory is used and shared by all the multiprocessors, but it has a high latency. Figure 4.1 shows a scheme of a CUDA-compatible GPU architecture.



**Figure 4.1:** CUDA compatible GPU Architecture.

CUDA architecture reflects a SIMT model: Single Instruction, Multiple Threads. These threads are executed simultaneously working in parallel onto large data. Each of them runs a copy of the kernel<sup>2</sup> on the GPU and uses local indexes to be identified.

Threads are grouped into blocks to be executed. Each of these blocks is allocated on a single multiprocessor, enabling the execution of several blocks within a multiprocessor. The number of blocks executed depends on the resources available on the multiprocessor and they are scheduled by a system of priority queues. Within each of these blocks, the threads are grouped into sets of 32 units in order to perform a fully parallel execution onto processors. Each set of 32 threads is called *warp*. In the architecture there exist certain constraints about the maximum number of blocks, *warps* and threads that are executed on each multiprocessor, but it varies depending on the generation and model of the GPU architecture. In addition, these parameters are set for each execution of a kernel to get the maximum occupancy of hardware resources and obtain the best performance.

The CUDA architecture has also a memory hierarchy with different types of memory: constant, texture, global, shared and local registries. The shared memory is useful to implement cache strategies. Texture and constant memories are used to reduce the computational cost avoiding global memory accesses which have high latencies.

In addition, GPUs are ideally suited to execute data-parallel algorithms. These algorithms execute identical work-units (programs) over large sets of data. Algorithms are efficiently parallelized when the work-units are independent and are able to run on small divisions on the data. One critical aspect of designing parallel algorithms is to identify the work-units and determine how they interact via communication and synchronization patterns. A second critical aspect is the analysis of memory access patterns. It is also necessary to consider the program execution pipeline in order to avoid unnecessary data transfers.

---

<sup>2</sup>Piece of code that is executed in parallel on the GPU.

### 4.3 GPU-based implementation of the GNG algorithm

In the last years, a large number of applications have used GPUs to speed up the processing of neural networks algorithms [Jang et al., 2008, Oh, 2004, Nageswaran et al., 2009, Juang et al., 2011, Igarashi et al., 2011]. GPUs have been also applied to different computer vision problems such as the representation and tracking of objects in scenes [Uetz and Behnke, 2009], face representation and tracking [Nasse et al., 2009] or pose estimation [Oh and Jung, 1999].

Moreover, several GPU-based SOMs implementations have been developed in the last years. In [Prabhu, 2008] it was presented a GPU parallel implementation called SOMGPU. This implementation of the SOM algorithm was focused on a pattern recognition problem using gray-scale images. That work used the Microsoft Research Accelerator technology to implement a GPU parallel version of the algorithm. This is a data-parallel library focused on GPUs, CPUs, and other processors. [Xiao et al., 2011] proposed a GPU-based implementation for SOM training using vertex shaders to implement the nearest codevector searching. They implemented the batch mode [Cheng, 1997] instead of the iterative SOM training mode which is less suitable to be parallelized. The sequential implementation presents a better adaptation to the input space as partial adaptations are considered through the entire learning process. In [Mathew and Joy, 2010, Platos and Gajdos, 2010] GPU-based implementations of the original Kohonen Map algorithm using CUDA were presented. These works were focused on analysing the performance achieved by the CUDA implementation using different network parameters in clustering applications. Another SOM-based approach as the spherical SOM [Wu and Takatsuka, 2006] has been also implemented on the GPU [Moraes et al., 2012] achieving satisfactory performances.

In addition, several non GPU-based parallel implementations of SOMs have already been proposed [Mann and Haykin, 1990, Nordström, 1991, Hämäläinen, 2002, Wu et al., 1991]. These approaches addressed the intrinsic parallelism of SOMs using different hardware solutions as: multi-core

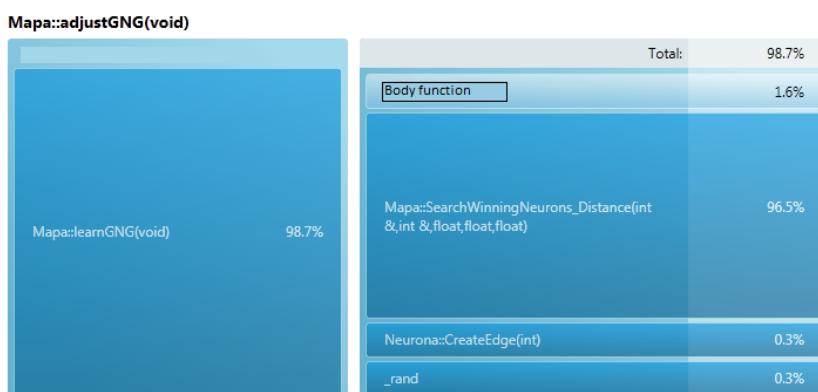
CPUs, supercomputers, VLSI, etc.

Regarding to parallel implementations of growing SOMs there are not many works that addressed this topic. Recently, [Vojacek and Dvorsky, 2013] proposed a high performance implementation of the GNG algorithm using distributed computing. This implementation was focused on high multidimensional data clustering, which is very time-consuming.

In this thesis, we proposed GPU-based implementations of the Growing Neural Gas (GNG) and Neural Gas (NG) algorithms due to nowadays there no exist GPU-based implementations of growing SOMs. First, since the GNG learning algorithm has a high computational cost, we proposed its acceleration by using GPUs and taking advantage of the many-core architecture provided by these devices, as well as their parallelism at the instruction level.

#### 4.3.1 Estimating the upper bound of the acceleration factor

Before tackling the parallel implementation of any algorithm, it is necessary to know the percentage of instructions executed. In order to achieve this, we used a profiler so that depending on the values of the parameters we used to adjust the algorithm (number of neurons and number of input patterns) we obtained the percentage of instructions executed at each stage. (Figure 4.2).



**Figure 4.2:** Percentage of instructions that are executed each stage for 20000 neurons and 1000 input patterns per iteration.

Table 4.1 shows the percentage of instructions occupied at each stage of the algorithm for different values of the number of neurons  $N$  and input patterns  $\lambda$ . It is also shown how stage 3 increases its percentage respect to the total, when  $N$  and  $\lambda$  are increased. Most time spent in the execution of the algorithm is consumed by the search of the winning neurons (stage 3), which also implies the computation of Euclidean distances.

Neurons	Patterns	Stage 2	Stage 3	Stage 4,5,6,7	Stage 8	Stage 9
1000	500	1,8	<b>73,30</b>	15	1,2	0,8
5000	500	0,7	<b>88,80</b>	5,8	0,9	1
10000	500	0,4	<b>93,20</b>	3,3	0,6	0,8
20000	500	0,3	<b>97,60</b>	1,9	0,5	0,9
1000	1000	1,8	<b>69,60</b>	21,3	0,6	0,3
5000	1000	0,7	<b>90</b>	5,6	0,5	0,5
10000	1000	0,4	<b>94,3</b>	3,2	0,3	0,4
20000	1000	0,3	<b>96,5</b>	1,9	0,2	0,4

**Table 4.1:** Percentage of executed instructions at each stage of GNG algorithm

Once this information was obtained we applied different metrics of parallel computing to estimate which would be the overall maximum acceleration that we may obtain assuming that these stages are accelerated by a factor  $S$ . The metrics used are widely known: Amdahl's Law [Amdahl, 1967] and other performance metrics of parallel computing [Gustafson, 1988, Sun and Gustafson, 1993, Hill and Marty, 2008].

In particular, we focused our study on the modern version of Amdahl's Law, which states that if a fraction  $f$  is accelerated by a factor  $S$ , the overall acceleration is:

$$Speedup(f, S) = \frac{1}{(1 - f) + \frac{f}{S}} \quad (4.1)$$

This is the equation that better estimates the theoretical maximum acceleration that can be obtained using parallel implementations on GPUs. It applies the achieved improvement on a fraction of the code instead of applying it to the total number of cores. The number of cores is used to measure the acceleration in the case of the execution onto a single GPU core, but in our case and in most of the cases, the acceleration that we get is related to the execution onto one CPU core. So  $S$  is defined as the

speed-up obtained regarding to a fraction of the CPU code.

Table 4.2 shows the estimated maximum accelerations we would achieve in the algorithm after accelerating a fraction  $p$  of the algorithm by a factor  $S$ . This was estimated by applying the Amdahl's law. Other implicit latencies exist in the architecture that will be discussed in later sections. The acceleration of the winning neurons search and Euclidean distance stages offered the highest overall acceleration.

Neurons	Patterns $\lambda$	s	p	Overall speed-up
1000	500	0,28	0,72	<b>3,29</b>
5000	500	0,11	0,89	<b>6,39</b>
10000	500	0,07	0,93	<b>8,73</b>
20000	500	0,02	0,98	<b>13,74</b>
1000	1000	0,30	0,67	<b>2,95</b>
5000	1000	0,1	0,9	<b>6,89</b>
10000	1000	0,06	0,94	<b>9,60</b>
20000	1000	0,04	0,97	<b>12,01</b>

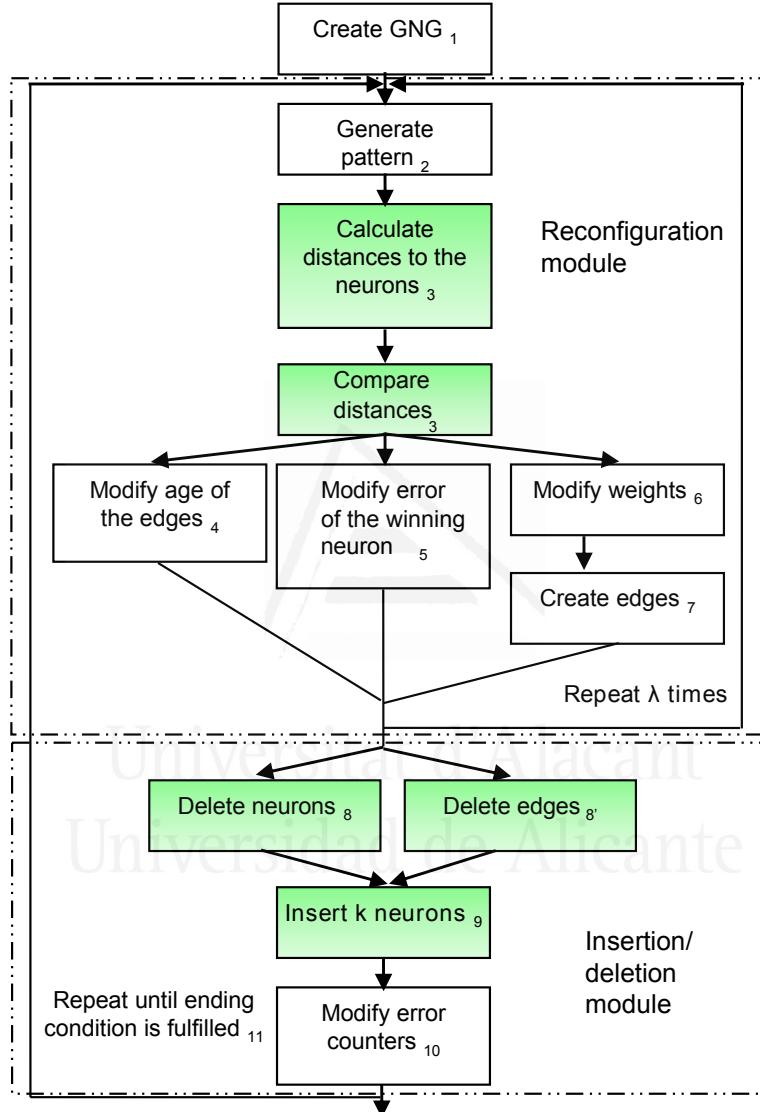
**Table 4.2:** Overall maximum acceleration estimated using Amdahl's law and assuming a factor 20 of speed-up regarding to a fraction  $p$  of the algorithm.  $s$  is the fraction of the algorithm that cannot be executed in parallel (sequential).

In the experiments section, real values for the speed-up of winning neuron search stage were obtained. Then, we applied Amdahl's law again to compare theoretical values with real overall speed-up obtained using the GNG algorithm. Thereby, we were able to measure how much time was consumed by other latencies like data transfers or the device initialization, allowing us to know the speed-up upper bound of the GNG algorithm.

### 4.3.2 GPU-based implementation

In order to accelerate the GNG algorithm on GPUs using CUDA, it was necessary to redesign it so that it fitted within the GPU architecture. Many of the operations performed in the GNG algorithm were parallelized because they act simultaneously on all the neurons of the network. That was possible because there is no direct dependence between neurons at the operational level. However, there exists a dependence in the adjustment of the network, which made necessary the synchronization of various

parallel execution operations at each iteration. Figure 4.3 describes GNG algorithm steps that were accelerated on the GPU.



**Figure 4.3:** GNG learning algorithm remarking the parallel stages.

#### 4.3.2.1 Euclidean distance calculation

The first stage of the algorithm that was accelerated was the calculation of Euclidean distances performed at each iteration. This stage calculates

the Euclidean distance between a random pattern and each of the neurons. This task takes place in parallel by running the calculation of each neuron distance using as many threads as neurons the network contains. It is possible to calculate more than one distance per thread, but this is efficient only for large vectors where the number of blocks executed on the GPU is also very high.

#### 4.3.2.2 Parallel reduction

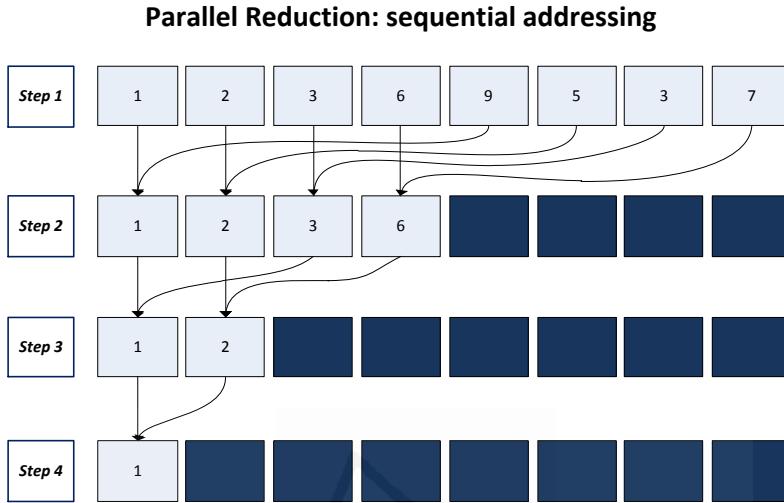
The second task parallelized was the search of the winning neuron: the one with the lowest Euclidean distance to the pattern generated and the second closest. For this search, we used a parallel reduction technique originally described in [Harris, 2008]. This technique accelerates parallel operations such as the search of the minimum value in large data sets.

For our GPU-based implementation we modified the original algorithm so that with a single parallel reduction step we not only obtain the minimum, but also the two smallest values of the entire data set. This new version was coined *2MinParallelReduction*. Figure 4.4 shows how parallel reduction is presented as a binary tree where at the end of the  $\log_2(n)$  steps we obtain the final result of the operation onto a set of  $N$  elements.

#### 4.3.2.3 Complexity

To perform the complexity calculation of this approach regarding to the sequential version, which has a computational complexity of  $O(N)$ , it should be noted that in parallel processing there exist three types of complexity: complexity in the number of execution steps, complexity of the work performed and time complexity. These complexities were identified in the parallel reduction algorithm:

- The complexity in terms of execution steps is  $O(\log_2(N))$  since it is necessary to perform  $\log_2(N)$  steps to reach the final result. Moreover, within each execution step  $s$ ,  $\frac{N}{2^s}$  operations are performed.
- The complexity of the work performed is: For  $N = 2d$  elements,  $\sum_{i=1}^S 2^{(d-i)} = N - 1$  operations are performed. The complexity of the work done is  $O(N)$ , where  $S$  is the total number of steps.



**Figure 4.4:** Example of Parallel Reduction Algorithm execution.

- Finally, the time complexity is  $O(N/P + \log_2(N))$ , where  $P$  is the number of processors.

Therefore, since each block launches  $t$  threads, each of them processing each element of the set  $N$ , the number of threads is equal to the number of elements. Considering this fact to calculate the time complexity, the time complexity is reduced to  $\log_2(N)$  compared to the complexity  $O(N)$  in the sequential version.

Despite this difference in complexity between the parallel and the sequential versions, the preparation and execution of programs on the GPU involves a time penalty, as well as the GPU memory transfers of data that causes a new penalty that begins to be compensated from a number  $X$  of processed elements. This issue also affects the cost of the operations performed onto the data.

#### 4.3.2.4 Other optimizations

To speed-up the remaining steps we followed the strategy used during the first phase. Each thread is responsible to perform different operations on a neuron: checking edges age, removing edges, updating local error of the neuron or adjusting neuron's weights. At the stage of finding the neuron with maximum error, the strategy followed is the same as the one used in the step 3 (winning neuron search), but in this case the reduction is looking only for the neuron with the highest error.

Regardless of the parallelism of the algorithm, we followed some good practices on the CUDA architecture to get more performance. First, the use of the constant memory to store the neural network parameters  $\epsilon_w$ ,  $\epsilon_n$ ,  $\alpha$ ,  $\gamma$ ,  $a_{max}$ . By storing these parameters in this memory, the access is faster than working with values stored in the global memory.

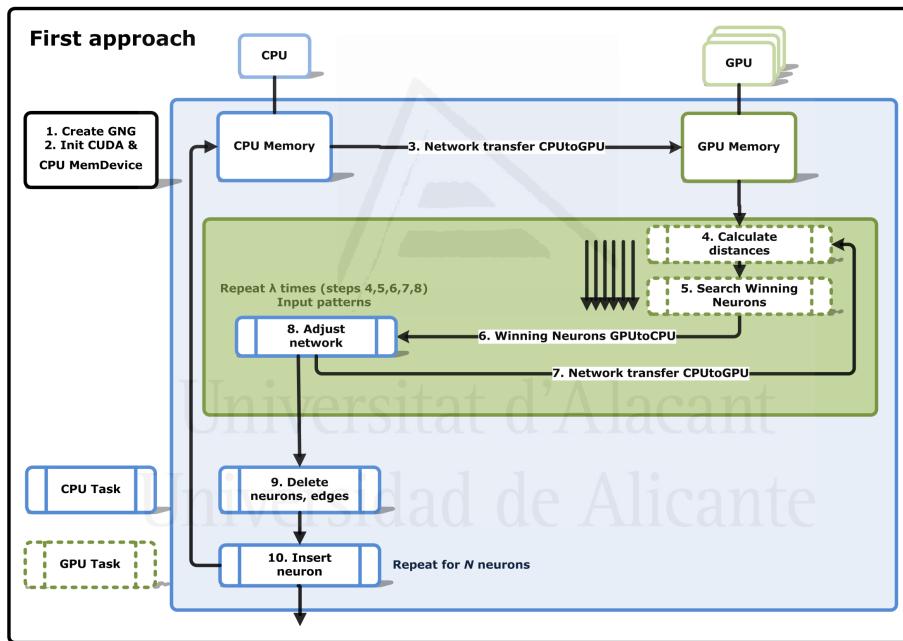
Our GPU-based GNG implementation is also limited by the memory bandwidth available. In the experiments section we show analysis reports for each CUDA capable device used and its memory bandwidth. However, this bandwidth was only achievable under highly idealized memory access patterns. However, it provided us with an upper limit of memory performance. Nevertheless, some memory access patterns, like moving data from the global memory to shared memories and registers, provided better coalesced access. The shared memory within each multiprocessor was used to get the highest performance of the memory bandwidth. So that, it acts as a cache to avoid frequent access to global memory in operations with neurons and allows threads to achieve coalesced reads when accessing neurons data.

For instance, a GNG network composed of 20,000 neurons and auxiliary structures requires only 17 MB of memory space. Therefore, GPU implementation in terms of size did not present problems as current GPU devices have enough memory to store it.

#### 4.3.2.5 Minimizing transfers approach

Memory transfers between CPU and GPU were the main bottleneck to obtain speed-up. So these transfers were removed as much as possible

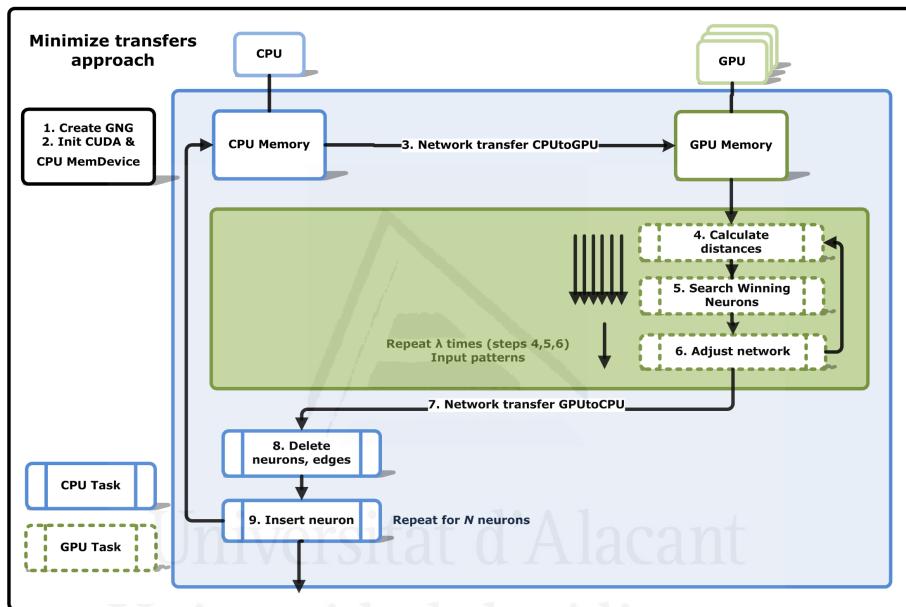
in the GPU-based implementation. Initial versions of the algorithm failed to obtain performance over the CPU version because the complete neural network was copied from GPU memory to CPU memory and vice versa for each generated input pattern. This penalty, introduced due to the bottleneck of the transfer through the PCI-Express bus, was so high that we did not improve runtime of the CPU version. After careful consideration of the flow of execution, we decided to move the inner loop of pattern generation to the GPU and removing memory transfers. It caused that some tasks were not executed in parallel and had to be run on a single GPU thread.



**Figure 4.5:** First approach, CUDA workflow.

Figure 4.5 shows the workflow of our first approach using CUDA. First, GNG network is created in the CPU and CUDA device is initialized. Then, the necessary space is allocated in the GPU memory to perform processing. Once the GNG network structure is copied to the GPU memory the learning algorithm begins: first, a random input pattern is generated and the Euclidean distance is calculated from each of the neurons. Second, these distances are calculated in parallel taking advantage of the massively par-

allel computing on the GPU and, the two neurons with the lowest distance (winning neurons) are also obtained using a parallel reduction. Then, the indexes of winning neurons are copied to the CPU memory and adjustment is performed (sequential). This step is repeated  $\lambda$  times. This first approach did not obtain improvement regarding of the CPU version because the entire neural network was copied from the CPU memory to the GPU  $\lambda$  times at each new neuron insertion, causing significant latencies.

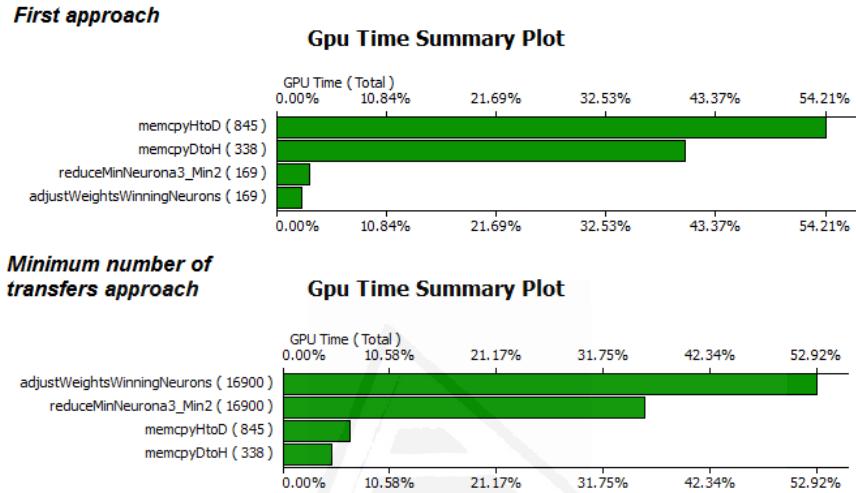


**Figure 4.6:** Approach that minimize transfers. CUDA workflow.

Figure 4.6 shows the approach used to avoid these large number of transfers between GPU and CPU memory. The inner loop was moved to the GPU, so it was not necessary to copy the network structure back to the memory of the CPU and make the adjustment. In this case, the adjustment is performed in a single thread onto the GPU because the task set is sequential and cannot be parallelized. Performing this task in a single thread in the GPU was better due to the high latency of transfers between GPU and CPU. It was also clear that reducing the number of memory transactions from the device memory results in a significant increase of the processing throughput.

Figure 4.7 shows that, for 500 patterns, the percentage of time spent

in the execution of the algorithm for memory transfers between CPU and GPU was drastically reduced. Thus, we increased the number of input patterns without increasing the number of transfers between memories.



**Figure 4.7:** Percentage of time spent in execution of the algorithm for memory transfers.

The use of CUDA in this algorithm provided better performance for a large number of neurons. This is due to the time needed to prepare some specific guidelines (GPU initialization) for the architecture implementation as kernels execution or GPU memory allocation. Performing these operations on small vectors of 50-500 neurons is almost immediate on the CPU, while the GPU cannot hide these inherent latencies in the architecture if a large number of neurons is not reached. Therefore, we considered the idea of applying hybrid techniques according to the restriction that the GNG is an incremental network that initially works with a small number of neurons, which grows progressively. This hybrid technique begins by running the GNG onto the CPU, but when it is detected that the runtime of the sequential version is higher than the runtime of the parallelized one, the network is copied to GPU memory and the remaining calculation is performed on the GPU.

### 4.3.3 Experiments

The accelerated version of the GNG algorithm was developed and tested on a machine with an Intel Core i3 540 3.07Ghz and different CUDA capable devices. Table 4.3 shows different models that we have used, including their features.

The multi-core CPU implementation of the GNG algorithm was developed using Intel Threading Building Blocks (TBB) library [Intel, 2012], taking advantage of the multi-core processor capabilities and avoiding the existing overhead [Bhattacharjee et al., 2011]. The number of threads used in the multi-core CPU implementation is the maximum defined in the specifications of Intel i3 540 processor.

Device Model	Capability	SMs	cores per SM	Global Mem	Bandwidth Mem
Quadro 2000	2.1	4	192	1 GB	41.6 GB/s
GeForce GTX 480	2.0	15	480	1.5 GB	177.4 GB/s
Tesla C2070	2.0	14	448	6 GB	144 GB/s

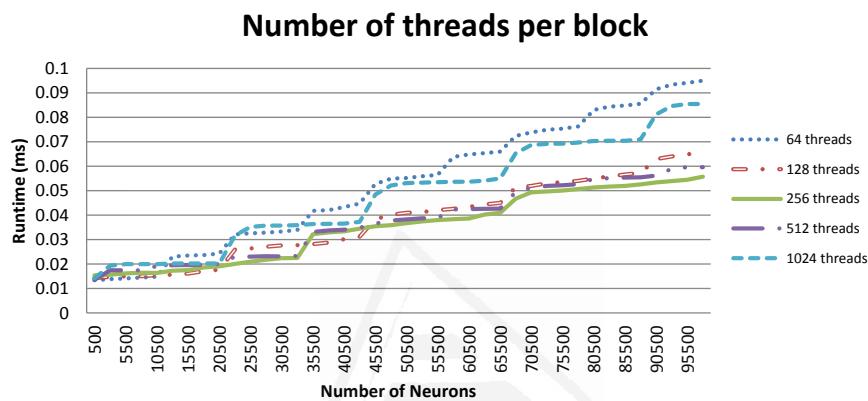
**Table 4.3:** CUDA capable devices used in experiments

First, we performed some experiments to obtain the best parameters to launch kernels in our application, obtaining the maximum occupancy of CUDA multiprocessors. Once we obtained the best parameters, we used them to test the *2minimumParallelReduction* implementation obtaining different performance depending on the graphics card used and its number of cores. Finally, we carried out some experiments to obtain the speed-up regarding to single thread and multi-thread CPU versions. Multi-thread CPU implementation using multi-core CPUs followed a similar parallelization scheme than the GPU-based implementation.

#### 4.3.3.1 Number of threads per block

As mentioned in Section 4.2, threads are organized into blocks to carry out their execution onto multiprocessors. Depending on the algorithm, a different number of threads per block should be used to obtain the best performance. We tested different kernels on the NVIDIA GTX 480 with different numbers of threads per block. The best performance was obtained when using a number of threads between 128 and 256 for a number of

neurons below 20,000. For a larger number of neurons, 256 and 512 threads per block obtained the best performance. These conclusions were extracted from Figure 4.8. Selected parameters obtained maximum occupancy of CUDA multiprocessors. These results were directly applied to the other tested graphics cards since the number of threads per block mostly depends on the type of algorithm and the way it is implemented.

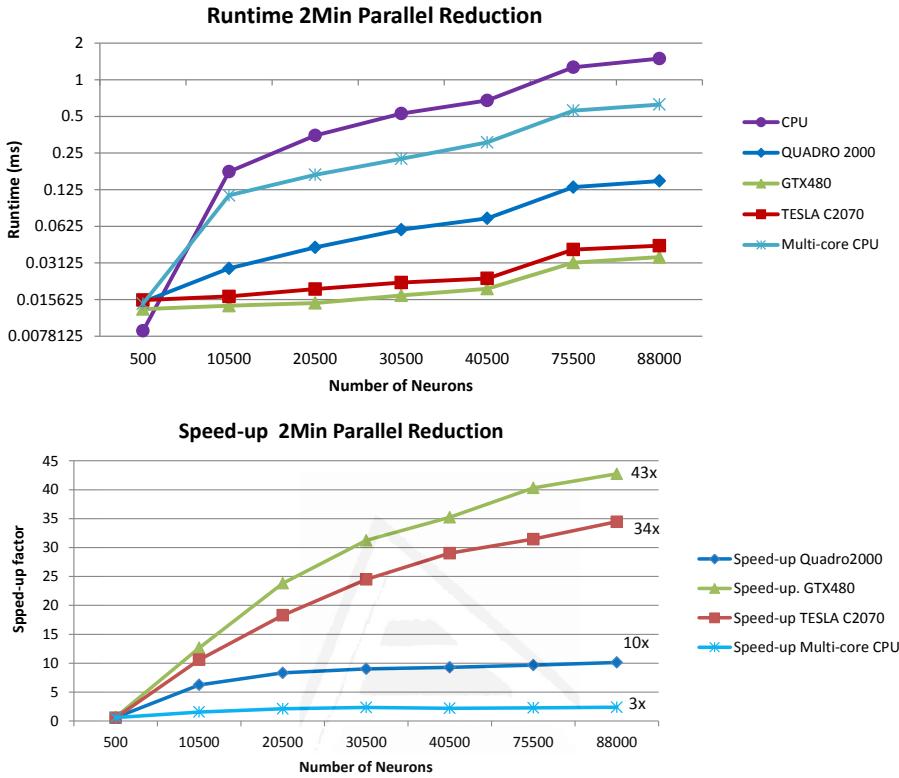


**Figure 4.8:** Execution time depending on the number of threads per block.

### 4.3.3.2 Speed-up 2 Min Parallel Reduction

We tested the *2MinParallelReduction* implementation with different graphics boards using 256 threads per block configuration for kernels launching. We obtained a speed-up factor up to 43x faster regarding a single-core CPU and 40x faster compared to the multi-core CPU, learning a network with a number of neurons equal to 100k. Figure 4.9 shows (bottom) the speed-up factor achieved with different devices also Figure 4.9 (top) shows the evolution of the execution time in the sequential reduction operation compared to the parallel version. It was also appreciated how it improved the acceleration provided by the parallel version as the number of elements grew.

Best results were obtained with the most powerful graphic card tested, in our case the NVIDIA GTX 480 with 480 cores and 1,5 GBytes of memory. Using others GPUs such as the Quadro 2000 model, which are found



**Figure 4.9:** Speed-up of *2MinParallelReduction* implementation using different graphic cards.

in desktop computers, we also obtained a good speed-up, 10x-15x faster than the CPU implementation.

Applying these values to Amdahl's law that we previously analyzed and replacing the speed-up factor of the fraction  $f$  by these values, we estimated the upper limit of speed-up we may obtain in the GNG algorithm. Thereby, the current acceleration was compared with the calculated upper limit and we extracted the percentage of time consumed by other latencies implied in the CUDA architecture.

Table 4.4 shows the estimated overall speed-up obtained for different parameters of GNG using the speed-up we empirically obtained accelerating stage 3 of the algorithm.

Neurons	Patterns	$\lambda$	s	p	Speed-up	Overall speed-up
1000	500	0,27	0,73		0,78	<b>0,83</b>
5000	500	0,11	0,89		2,95	<b>2,42</b>
10000	500	0,07	0,93		5,61	<b>4,27</b>
20000	500	0,02	0,98		10,60	<b>8,62</b>
1000	1000	0,30	0,70		0,69	<b>0,761</b>
5000	1000	0,1	0,9		2,9	<b>2,44</b>
10000	1000	0,05	0,94		5,65	<b>4,47</b>
20000	1000	0,03	0,96		10,68	<b>7,98</b>

**Table 4.4:** Theoretical overall speed-up obtained for GNG algorithm using speed-up obtained in stage 3. This experiment was executed on a NVIDIA GTX 480.

#### 4.3.3.3 GNG learning speed-up factor

To test our parallel version of the GNG algorithm, we did experiments using the GNG for 3D data representation. To solve the problem of 3D representation, the necessary number of neurons to adapt the input space was high which benefited the use of the GPU. Therefore, an increase in speed over the CPU version was achieved.

Based on a previous work [García-Rodriguez et al., 2011], it was chosen a number of neurons  $N$  of 1000 / 5000 / 10000 / 200000 and a number of input patterns  $\lambda$  of 500/1000. Other parameters were also fixed based on our previous experience:  $\epsilon_w = 0.1$ ,  $\epsilon_n = 0.001$ ,  $\alpha = 0.5$ ,  $\gamma = 0.95$ ,  $a_{max} = 250$ .

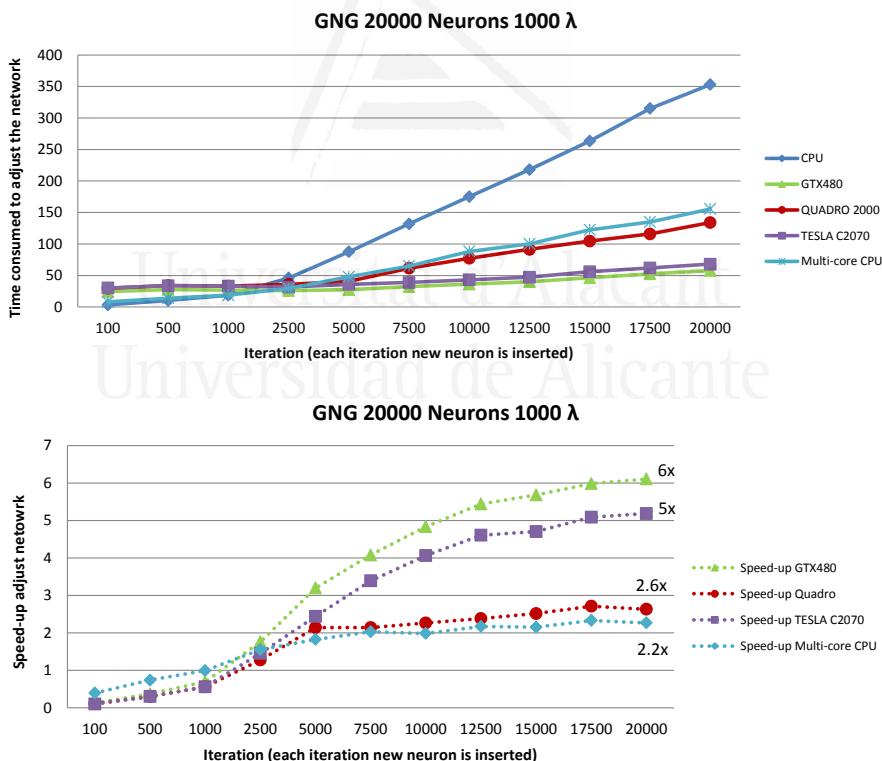
Figure 4.10 shows an experiment using GNG to represent a 3D object with 20,000 neurons and 1,000 input patterns where the CPU solution took more and more time as the number of neurons grew. However, the parallel CUDA version increased the size of the array of neurons without degrading significantly the performance. For a number of 20k neurons we obtained a 6x speed-up factor using a NVIDIA GTX 480 GPU. This speed-up is lower than the theoretical overall speed-up that we estimated in the previous section. This was caused by the implicit latencies of the GPU architecture. Table 4.5 shows differences between the theoretical overall speed-up and the obtained one.

Figure 4.10 shows that the CPU version was faster during the first iterations, so a hybrid version would be faster than separate CPU and

Neurons	Patterns	Theoretical overall speed-up	Real overall speed-up
1000	1000	0,76	0,68
5000	1000	2,44	2,36
10000	1000	4,47	4,23
20000	1000	7,98	6,11

**Table 4.5:** Theoretical overall speed-up and obtained overall speed-up using the device GTX480.

GPU versions. Multi-core CPU implementation was also slower during the first iterations compared with single-core CPU due to the existing overhead caused by the management of threads and by the subdivision of the problem.



**Figure 4.10:** Example of GPU and CPU GNG runtime, and speed-up for different devices.

#### 4.3.3.4 GNG hybrid version

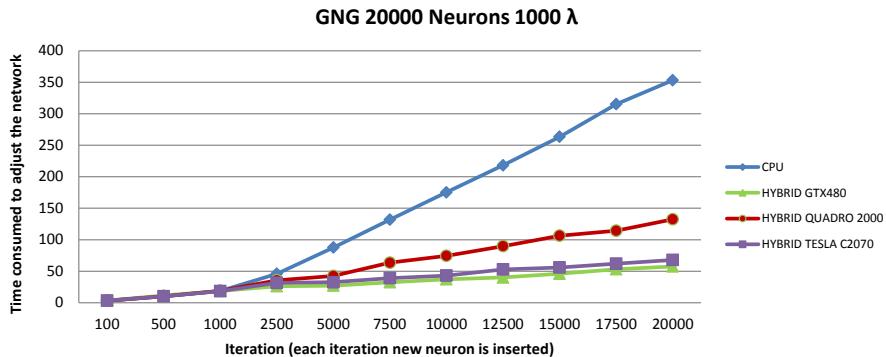
As we discussed in the previous experiments, GPU version had a low performance in the first iterations of the learning algorithm, where the GPU was not able to hide the latencies due to the small number of processing elements. To achieve even bigger acceleration of the GNG algorithm, we proposed the use of the CPU in the first iterations of the algorithm, and then start processing data in the GPU only when there was an acceleration regarding the CPU implementation, thus achieving a bigger overall acceleration of the algorithm (see Figure 4.11).

To determine the minimum number of neurons necessary to start computing on the GPU, we analyzed in detail the execution times for each new neuron insertion. We concluded that each device, depending on its computing power, started being efficient at a different number of neurons. Figure 4.10 (top) shows how we inferred threshold values for different devices looking at the number of neurons that caused that GPU implementation started being faster than the CPU one. Therefore, threshold values were set to 1,500, 1,700, 2,100 for GTX 480, Tesla C2070 and Quadro 2000 models respectively.

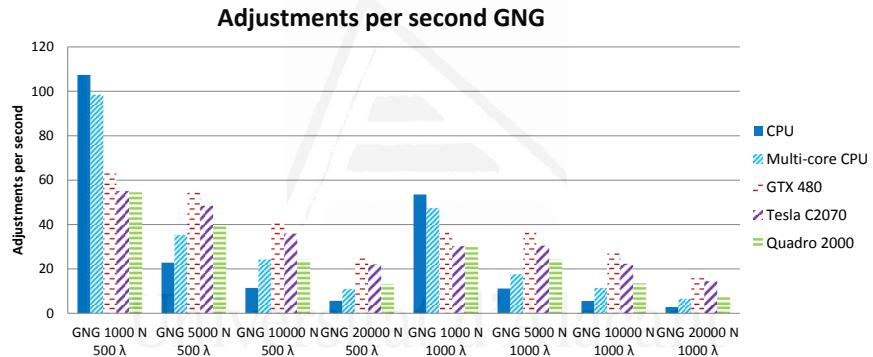
The hybrid version was proposed as some applications need to operate under time constraints obtaining a solution with a specified quality in a period of time. In cases when the goal is the disruption of learning due to the application requirements, it is important to insert the maximum number of neurons and perform the maximum number of adjustments to achieve the highest quality in a limited time. The hybrid version ensures a maximum performance in this type of applications using the computational capabilities of the CPU and the GPU. For example, our proposal was validated on 3D scene reconstruction performed by a mobile robot. This will be further studied in Section 5.2.

#### 4.3.3.5 Rate of adjustments per second

Finally, we performed several experiments that showed how the accelerated version of the GNG was not only capable of learning faster than CPU, but also obtained more adjustments per second than the single-thread and



**Figure 4.11:** Example of CPU and Hybrid GNG runtime for different devices.



**Figure 4.12:** Rate of adjustments per second performed by different GPU devices and CPU.

multi-thread CPU implementations. For instance, after learning a network of 20,000 neurons we performed 17 adjustments per second using the GPU while the single-core CPU performed 2.8 adjustments per second and the multi-core CPU performed 8 adjustments per second. This means that the GPU implementation is able to define a good topological representation even under time constraints. Figure 4.12 shows the different adjustments rates per second performed by different GPU devices compared to CPU. It shows how the increase of the number of neurons in the CPU caused a lower rate of adjustments per second.

#### 4.3.3.6 Discussion

From the experiments described above we concluded that the number of threads per block that best fitted in our implementation was 256 due to the following reasons: First, the amount of computation the algorithm performs in parallel and second, the number of resources that each device has and finally the use of shared memories and registries. It was also demonstrated that in comparison to the CPU implementation, the *2MinParallelReduction* achieved a speed-up of more than 40x to find out a neuron with the minimum distance to the generated input pattern.

Theoretical values obtained applying Amdahl's law and its comparison with real values obtained from the experiments indicated that GPGPU architecture had some implicit latencies: initialization time, data transfers time, memory access time, etc.

Experiments on the complete GNG algorithm showed that using the GPU on small networks under-utilize the device, since only one or a few multiprocessors were used. Therefore, the GPU-based implementation has a better performance for large networks than for small ones. To get better results for small networks we proposed a hybrid implementation. Experiments on the hybrid version showed that GNG learning achieved a speed-up 6 times faster than the single thread CPU implementation.

Finally, it was shown how our GPU implementation processed up to 17 adjustments per second of the network while single thread CPU implementation only managed 2.8, getting a speed-up factor of more than 6 times.

## 4.4 Neural Gas implementation using GPUs

The Neural Gas (NG) algorithm is also computationally demanding. Its computational cost is even higher than the GNG algorithm. Therefore, we proposed to accelerate it using GPUs. As we did with the GNG algorithm, a profiling analysis was performed in order to identify the algorithm steps with larger computational complexity. Complete information about the performance analysis is not shown as most information is redundant. Looking at the performance analysis we extracted that steps 3,

4 and 7 presented in Section 2.2.2.1 and corresponding to Euclidean distances calculation, sorting distances, weights update and edges removal, were identified as the steps with larger computational cost. In addition, many of these operations performed in the NG algorithm were parallelized since they operated simultaneously on all the neurons of the network. That was possible because there is no direct dependence between neurons at the operational level, as happened in the GNG algorithm. However, there exists dependence in the adjustment of the network, which makes necessary the synchronization after computing each iteration  $t$  of the algorithm.

The first accelerated stage of the algorithm was the calculation of Euclidean distances performed at each iteration. This stage calculates the Euclidean distance between a random pattern and each of the neurons. This task takes place in parallel by running the calculation of each neuron distance on as many threads as neurons the network contains. It is possible to calculate more than one distance per thread. This approach increases the level of parallelism as the number of neurons is increased, obtaining a larger speed-up for a large number of neurons.

The second parallelized task was the sorting stage, corresponding to step 3 of the NG method. For this task, we used a parallel sorting technique implemented in the Thrust library [Bell and Hoberock, 2011]. Thrust dispatches a GPU-based highly-tuned Radix Sort algorithm [Merrill and Grimshaw, 2010] which is considerably faster than alternative comparison-based sorting algorithms such as Merge Sort [Satish et al., 2009].

Once neurons are sorted, weights update step is computed in parallel, launching as many threads as neurons comprise the network. Before continuing with the next possible parallelizable step, NG learning algorithm establishes that an edge between first and second winner neurons has to be created (step 5). Moreover, all edges emanating from winner neuron increment their age (step 6). Steps 5 and 6 are not possible to implement in parallel since they affects to only two neurons and their computational cost is really low. Therefore, in order to avoid data communication between CPU and GPU, steps 5 and 6 are performed sequentially on the GPU, launching one GPU thread that performs edge creation and age increment. Finally, step 7 which corresponds to edge age checking was

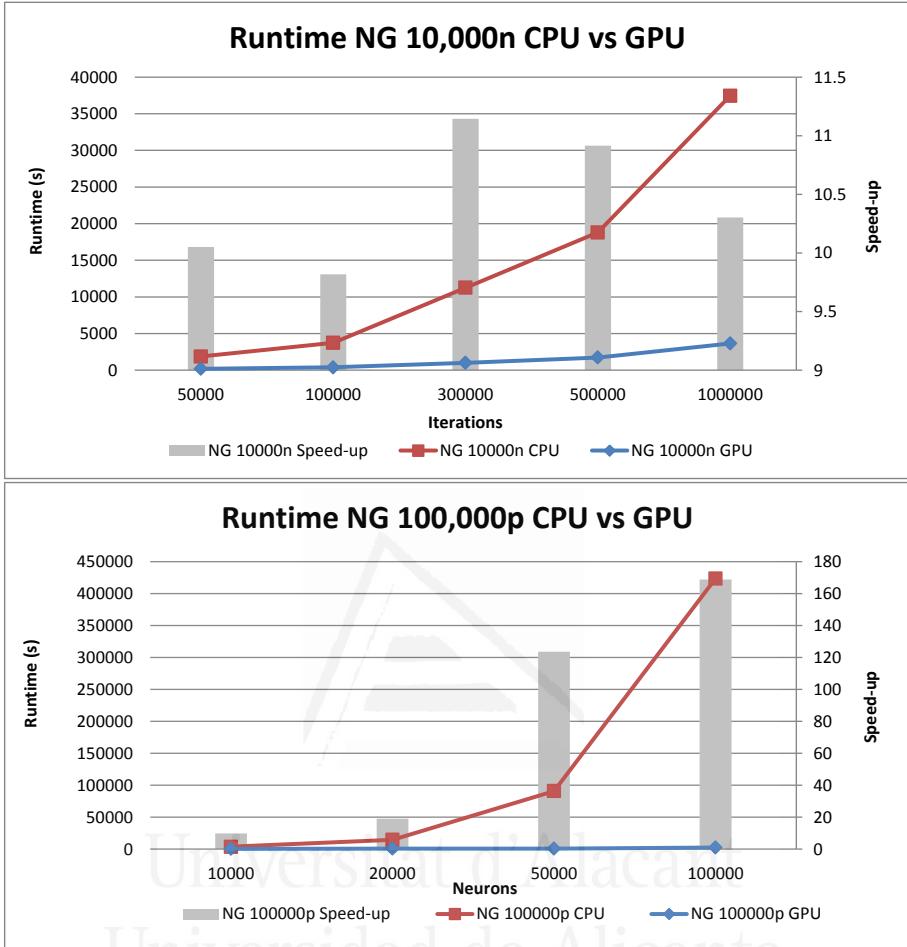
performed in parallel. Parallelizing the algorithm in this way, communication between CPU memory and GPU memory was avoided during the algorithm execution and therefore the main computational bottleneck was prevented.

#### 4.4.1 Experiments

First experiment demonstrated how speed-up obtained using GPU-based NG implementation was affected by the algorithm learning parameters. As we discussed in Section 4.3, parallelism of the GPU implementation is limited by the number of neurons. The number of iterations that the algorithm performs does not affect the speed-up obtained. However, experiments demonstrated that a larger number of neurons obtained higher acceleration. This statement is demonstrated in the Figure 4.13, where two different experiments were performed. First one (top), given a fixed number of neurons, runtime execution was measured using different number of iterations. Second one (bottom), runtime execution was measured establishing a fixed number of iterations and increasing the number of neurons. The speed-up obtained was in the range of 9-11x and it was not considerably affected as the number of iterations was increased, while on the top side it is shown how as the number of neurons was increased the speed-up obtained was considerably improved, obtaining a speed-up close to 180x for 100,000 neurons.

For the NG implementation, we omitted experiments related with the search of the optimum number of threads per block as the same values as in the GPU-based GNG implementation were empirically found.

Moreover, GPU-based implementation was tested exhaustively on different point clouds with different attributes and shapes using different learning parameters in order to compare speed-up obtained regarding to the sequential CPU version. Table 4.6 (top) shows runtime executions for different 3D models with different number of points and using different learning parameters. We concluded that as we increased the number of neurons and number of iterations, the speed-up achieved by the GPU-based NG implementation was considerably increased. Moreover, Table 4.6 shows that the number of neurons has more influence in the speed-up



**Figure 4.13:** (Top) Runtime execution (GPU GTX480) (lines) and speed-up (bars) for a fixed number of neurons (10,000) and different number of iterations. (Bottom) Runtime execution (GPU GTX480) (lines) and speed-up (bars) for a fixed number of iterations (100,000) and different number of neurons.

achieved than the number of iterations. These results also demonstrated how the proposed method is suitable for massively parallel architectures such as the GPU, where each thread processes one of the neurons.

Table 4.6 (bottom) shows the speed-ups achieved for different GPUs. For a small number of neurons, GPU devices behaved similarly, but as the number of neurons was increased and therefore the parallelism was also higher, the device with more number of cores (GTX480) achieved better performance (speed-up).

Model	Npoints	Nneurons	Iterations	CPU time	GTX480 time	Q2k time
Bunny	34,834	3,000	60,000	233.406	114.468	109.802
Bunny	34,834	5,000	80,000	611.39	192.213	160.771
Horse	48,485	5,000	80,000	605.561	153.669	169.053
Horse	48,485	8,000	100,000	1918.979	309.3116	256.864
Armadillo	172,974	10,000	150,000	4306.184	572.912	526.822
Armadillo	172,974	15,000	200,000	12881.386	1040.818	1005.559
Happy Buda	543,652	20,000	300,000	34189.155	2257.832	1096.175
Dragon	3,609,600	50,000	600,000	431100.45	4453.178	8011.181

Model	Npoints	Nneurons	Iterations	Speed-up GTX480	Speed-up Q2k
Bunny	34,834	3,000	60,000	<b>2.039x</b>	<b>2.125x</b>
Bunny	34,834	5,000	80,000	<b>3.180x</b>	<b>3.802x</b>
Horse	48,485	5,000	80,000	<b>3.940x</b>	<b>3.582x</b>
Horse	48,485	8,000	100,000	<b>6.204x</b>	<b>7.470x</b>
Armadillo	172,974	10,000	150,000	<b>7.516x</b>	<b>8.173x</b>
Armadillo	172,974	15,000	200,000	<b>12.376x</b>	<b>12.810x</b>
Happy Buda	543,652	20,000	300,000	<b>15.142x</b>	<b>31.189x</b>
Dragon	3,609,600	50,000	600,000	<b>96.807x</b>	<b>53.812x</b>

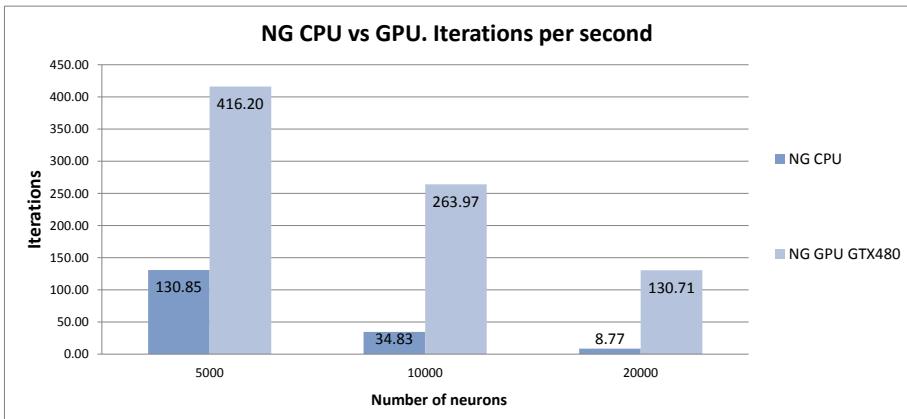
**Table 4.6:** Runtimes (top) and speed-ups (bottom) obtained using the accelerated version of the NG algorithm on different 3D models. All runtimes are measured in seconds.

#### 4.4.1.1 Adjustments per second

We also performed another experiment that showed how the accelerated version of the NG was not only capable of learning faster than the CPU implementation, but also obtained more adjustments per second than the sequential CPU implementation. For instance, learning a network of 20,000 neurons performed 130.71 adjustments per second using the GPU while the single-core CPU gets 8.77 adjustments per second. This means that the GPU implementation obtains a good topological representation with time constraints. Figure 4.14 shows the different adjustments rates per second performed by GPU compared to CPU implementation. It is also shown that when increasing the number of neurons in the CPU, it was not able to handle a large rate of adjustments per second.

#### 4.4.2 Discussion

In order to accelerate the NG algorithm we redesigned and implemented de NG learning algorithm to fit it onto a Graphics Processor Unit using CUDA. We obtained in the best case a speed-up of 180x compared to



**Figure 4.14:** Computed iterations per second. NG CPU vs GPU.(GTX480).

the CPU version. Moreover, experiments performed on the GPU-based implementation of the NG algorithm showed that the NG algorithm is even more parallel than the GNG. Therefore, it obtained larger speed-ups compared to the ones achieved in the GPU-based implementation. Although these speed-ups were larger, the overall time needed to complete the learning process was several times larger than the GNG algorithm, as the sorting distances stage implied a lot of computation.

## 4.5 GPU-based tensor extraction algorithm

In this section of the thesis we focused on time-constrained 3D feature extraction. Due to the necessity of extracting knowledge from the input data in order to solve different computer vision problems: e.g. registration, object recognition and scene understanding. Moreover, most feature extraction algorithms cannot be computed online due to their complexity and the context where they are applied. Therefore computing these features in real-time for many points in the scene is impossible. A hardware-based implementation of a 3D feature extraction algorithm and a 3D object recognition application was proposed in order to accelerate them and therefore the entire pipeline of RGB-D based computer vision systems where such features are typically used.

With the advent of the GPU as a General Purpose Graphic Processing

Unit (GPGPU) some methods related to 3D data processing have been implemented on GPUs in order to accelerate them. Examples of these can be found in the calculation of feature descriptors and keypoint extraction on the GPU. In [Griffin et al., 2012], the GPU performed curvature estimation of 3D meshes in real time. The work presented in [Prisacariu and Reid, 2009], provided a parallel implementation using the GPU and the CUDA language from NVIDIA to accelerate the computation of Histograms of Oriented Gradient (HoG) features. In [Himmelsbach et al., 2009], a Point Feature Histograms (PFH) GPU implementation was proposed allowing its computation in real-time on large point clouds. [Olesen et al., 2012] presented a real-time GPU-based method for patches extraction. These surfaces patches with associated uncertainties were extracted by means of Kinect cameras. Despite the methods mentioned above, which demonstrated the feasibility of the GPU for 3D feature extraction processes, there are still comparatively few methods implemented with respect to all those currently prevalent in the state of the art. Most 3D Local Shape Descriptors (LSDs) have not been implemented yet on the GPU. It can also be noted that the integration of these methods in complete systems, that require real-time constraints, is still very low. Kinect Fusion [Izadi et al., 2011] has been one of the first works where the GPU has been used as the main core processor, allowing the reconstruction of 3D scenes in real-time.

Other motivations for this work include the existing gap of 3D object recognition solutions based on models that support real-time constraints. Until now, most of the proposed works that supported real-time constraints were view-based. For example in [Blum et al., 2012], a local feature descriptor for RGB-D images was proposed. This descriptor combines color and depth information into one representation. However, 3D information possibilities are still negligible, it depends mostly on textures and illumination of the specific scene. In [Lee et al., 2011], a combined descriptor formed by 3D geometrical shape and texture information was used to identify objects and its pose in real-time. The proposed system is accelerated using GPU achieving real-time processing. However, 3D information is only used to extend shape information and it considerably relies on texture information, making it sensitive to scene illumination conditions.

Model-based approaches are less sensitive to illumination, shadows, and occlusions of the scene, allowing more robust object recognition systems. They also improve pose estimation. However, they also have some drawbacks: first, model-based descriptors usually require noise-free and dense data, which is not common in range cameras. Moreover, these type of systems require a large number of descriptors to be calculated and also their correspondences in the database have to be found at runtime. Consequently, most of them cannot be processed in real-time.

The GPU-based feature descriptor implementation proposed in this section was used and validated in a object recognition application in cluttered scenes (Section 5.3).

The feature descriptor proposed for implementation on the GPU is based on the one introduced in [Mian et al., 2006b]. This descriptor is based on the calculation of semi-local surface patches obtained by a range camera. It has been used successfully for different applications like global registration and 3D object recognition, where other descriptors like Spin Images [Johnson and Hebert, 1999] or Geometric Histograms [Hetzler et al., 2001] obtained worse results. Additionally, in [Mian et al., 2006a] it was demonstrated how this descriptor can be successfully applied in object recognition problems with high levels of occlusion. The main problem of this descriptor is its high computational cost, which is prohibitive for running in a CPU under real-time constraints.

This descriptor extracts a semi-local model of the scene, computing semi-local features that assist the local object recognition even under conditions of occlusion. This feature is referenced in the following sections as a tensor. A tensor is defined by the surface of the model that is intersected by each voxel on a centered grid. This set of values define a third order tensor.

To compute the required descriptor some preprocessing steps were needed to be computed a priori. These steps were implemented on the GPU in order to accelerate the entire processing pipeline. Therefore, it was necessary to implement on the GPU the following processes: depth map and color map (RGB-D) transformation to a coloured point cloud, noise removal, normal estimation and surface reconstruction. A general

system overview for semi-local surface patch extraction is shown in Figure 4.15. Moreover, Figure 4.16 shows the different steps required prior to the extraction of the tensor and their 3D visualization after each step. As we can appreciate, most steps are computed on the GPU taking advantage of parallel computing power of the GPU and avoiding transfers between CPU and GPU after each step. Pseudo-code of the entire GPU-based tensor extraction algorithm is presented in Algorithm 2. Moreover, pseudo-code snippets for all GPU-based preprocessing steps are presented in next sections.



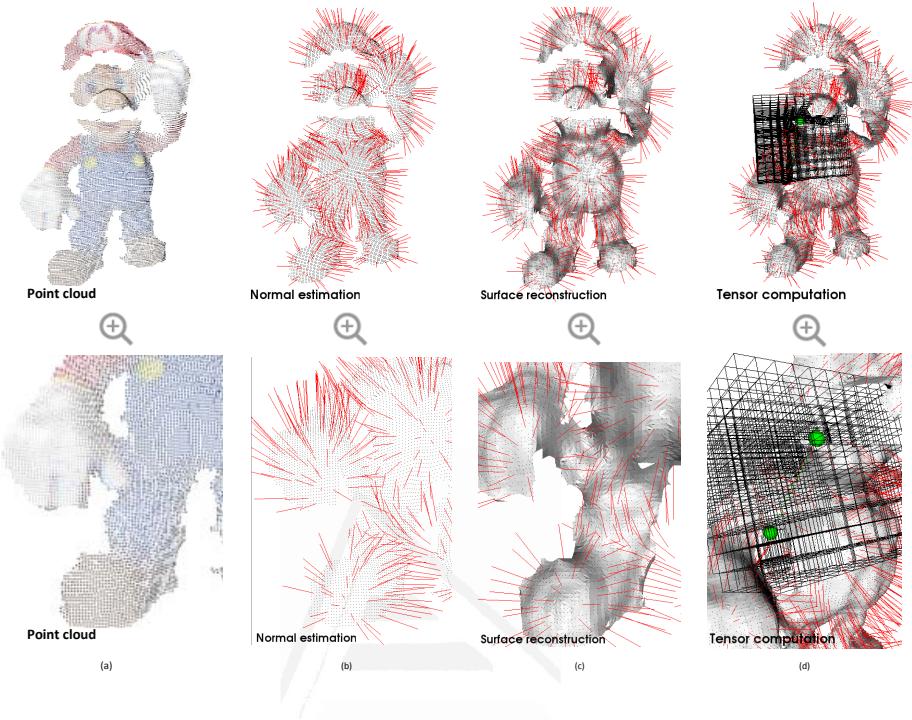
**Figure 4.15:** General system overview. Steps coloured in dark are computed on the GPU.

**input** : A depth map  $M_d$  of size  $640 \times 480$   
**output**: A set of 3D tensors  $T = \{t_0, t_1, t_2, \dots, t_N\}$  that describe the input data

```

1 Depth map is transferred to the GPU memory;
2 cudaMemcpyHostToDevice(d_Md, Md);
3 d_Mfiltered ← gpuBilateralFiltering(d_Md);
4 d_Pxyz ← gpuPointCloudProjection(d_Mfiltered);
5 d_Nxyz ← gpuNormalEstimation(d_Pxyz);
6 d_Tri ← gpuSurfaceTriangulation(d_Pxyz, d_Nxyz);
7 d_Vpairs ← gpuValidPairsComp(d_Pxyz, d_Nxyz);
8 cudaMemcpyDeviceToHost(Vpairs, d_Vpairs);
9 for i ← 0 to |Vpairs| do
10   | tensors are computed in parallel at thread level and tensor level;
11   | AsyncGpuTensorComp(vi,d_Pxyz,d_Nxyz,d_Tri, d_T);
12 end
13 cudaMemcpyDeviceToHost(T, d_T);
  
```

**Algorithm 2:** Pseudo-code of the GPU-based 3D tensor extraction algorithm.  $d\_$  prefix means that variable is allocated in the GPU memory.



**Figure 4.16:** (a) Point cloud obtained after transforming depth and color maps provided by the Kinect sensor. (b) Normal estimation. (c) Surface reconstruction. (d) Feature descriptor extraction: 3D tensor computed over a partial view

#### 4.5.1 RGB-D processing on the GPU

In this section we focused on the processing of 3D information provided by the Kinect sensor on the GPU. This processing is performed on the GPU with the aim of achieving a real-time implementation. Therefore, the overall goal is the implementation of systems that offer interaction with the user. The first step to carry out on the GPU, with the aim of accelerating future steps, was the projection of the depth and color information in a three-dimensional space, where the depth and colour information was aligned, allowing the production of a coloured point cloud that represents the scene.

This projection was computed independently for each pixel of the map, so it fitted perfectly on massively parallel architectures such as the GPU,

accelerating processing time related to the CPU implementation. As this transformation is often followed by other processing steps, it was not necessary to copy data back to the CPU memory and we therefore avoid the latency caused by these transfers by storing the projected 3D points on the GPU memory. Pseudo-code of the kernel executed onto the GPU is shown in Algorithm 3. Section 4.5.4.1 shows the acceleration factor and time of execution obtained by the GPU implementation. All these methods were developed in C++. Finally, 3D data management (data structures) and their visualization was done using the PCL library.

```

input : A depth map  $M_d$  of size  $640 \times 480$ 
output: Projected point cloud  $P_{xyz}$  into 3D space

1 __global__ void;
2 gpuPointCloudProjectionKernel(  $M_d$  );
3 {
4     This kernel is executed creating one thread for each pixel in parallel;
5     int u = threadIdx.x + blockIdx.x * blockDim.x;
6     int v = threadIdx.y + blockIdx.y * blockDim.y;
7     float z =  $M_d[v][u]$  / 1000.f;
8     // metres
9     float px = z * (u - cx) * fx_inv;
10    float py = z * (v - cy) * fy_inv;
11    float pz = z;
12 }
```

**Algorithm 3:** Pseudo-code of the GPU-based point cloud projection algorithm.

#### 4.5.1.1 Noise removal: Bilateral filtering

In structured light imaging a predefined light pattern is projected onto an object and simultaneously observed by a camera. The appearance of the light pattern in a certain region of the camera image varies with the camera-object distance. This effect is used to generate a distance image of the acquired scene. The predefined light patterns can be e.g. gray codes, sine waves, or speckle patterns. Speckle patterns are used in popular structured light (infrared) cameras like the Microsoft Kinect.

This method of obtaining 3D information from the scene presents problems when the surfaces have a high levels of specularity (reflection of the

incident light) making it impossible for the sensor to obtain depth information about some surfaces [Zhang, 2012]. The same problem occurs in the case of objects that are very far away from the sensor. Therefore, if we want to extract coherent information of the observed surfaces it is necessary to minimize this observation error.

In previous works, simple filters such as the mean or the median were used as they correct the error and run in real-time. As the computing power of the GPU can be applied in this step, this allows the application of more complex filters that are able to reduce the depth map error without removing important information, such as edge information.

An example of these filters, the Bilateral filter [Tomasi and Manduchi, 1998], is able to remove noise of the image whilst preserving edge information. This filter was used originally in color and grey scale images to reduce the noise while keeping edge information, but we can also use it to reduce the noise on depth maps obtained from 3D sensors like the Kinect. The basic idea underlying bilateral filtering is to do in the range of an image what traditional filters do in its domain. Two pixels can be close to one another, that is, occupy nearby spatial location (domain), or they can be similar to one another, that is, have nearby values (range). Therefore, a bilateral filter is a combination of a domain kernel, which gives priority to pixels that are close to the target pixel in the image plane, and a range kernel, which gives priority to the pixels which have similar values as the target pixel. The new value of a filtered pixel is given by:

$$P_f = \frac{1}{K_p} \sum_{q \in \omega} V_q f(||p - q||) g(||V_p - V_q||) \quad (4.2)$$

where  $K_p$  is a normalization factor,  $\omega$  is the neighbourhood of the target pixel,  $V_p$  and  $V_q$  are the depth values of the target pixel and the queried neighbour.  $P_f$  is the filtered value of pixel  $p$ . This equation also contains the domain kernel and the range kernel:  $f(||p - q||)$ ,  $g(||V_p - V_q||)$ . Often,  $f$  and  $g$  are Gaussian functions with standard deviation  $\sigma_s$  and  $\sigma_r$ .

Section 4.5.1.2 shows how the estimation of the normal vectors is improved after applying bilateral filtering, providing more stable normal vectors by removing original noise presented on the depth map. Most 3D

features extracted from the scene are based on the curvature of the geometry, which is calculated using information from normal vectors at each point in the scene, therefore obtaining more stable normal vectors leads to more accurate scene knowledge.

The calculation of filtered values at each pixel of the image can be calculated independently and therefore was well suited for parallel architectures like the GPU. In [Chan et al., 2008] and [Wasza et al., 2011] GPU implementations able to run in real time were proposed. The runtime was considerably improved, allowing filtering in real time depth maps generated by the sensor. In Section 4.5.4 GPU and CPU runtimes and speed-ups for our implementation on different graphics boards are presented.

#### 4.5.1.2 Normal estimation

Normal estimation was computed using the same approach that was presented in Section 2.3.3.1, but accelerating its computation on the GPU.

Once we had the organized point cloud stored in the GPU memory, the normal estimation process using PCA was performed efficiently on the GPU. The normal vector calculation was performed on the GPU independently at each point of the scene, considerably accelerating the runtime. Algorithm 4 shows the pseudo-code of the GPU-based normal estimation algorithm. Moreover, thanks to the previous noise removal process using bilateral filtering, normal vectors obtained were much more stable than normal vectors computed directly from the original depth map that did not take into account the borders and corner points of the scene. Figure 4.17 shows this effect.

#### 4.5.2 Surface triangulation on the GPU

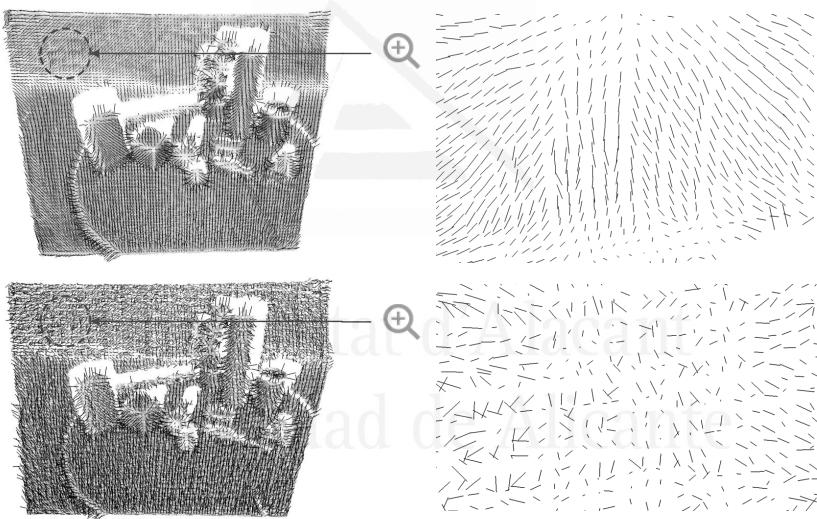
As we introduced before, the computation of the proposed descriptor required a polygonal mesh as input data. Although in Section 2.3.7 the GNG algorithm was successfully used for modelling triangular meshes of point clouds it was not capable of providing triangulation in real-time. The GPU-based implementation achieved a great acceleration but the surface reconstruction modification is really computationally demanding, being

```

input : A projected point cloud  $d\_P_{xyz}$ 
output: Point cloud of normals  $d\_N_{xyz}$ 

1 __global__ void;
2 gpuNormalEstimationKernel(  $P_{xyz}$ ,  $k$  );
3 {
4     This kernel is executed creating one thread for each point in parallel;
5     int u = threadIdx.x + blockIdx.x * blockDim.x;
6     int v = threadIdx.y + blockIdx.y * blockDim.y;
7     Compute Covariance matrix centered at point p using k neighbours;
8      $d\_N_{xyz}[u][v] = \text{compCovarianceMat}(u,v,k,N);$ 
9      $d\_N_{xyz}[u][v] = \text{checkOrientation}();$ 
10 }

```

**Algorithm 4:** Pseudo-code of the GPU-based normal estimation algorithm

**Figure 4.17:** Bottom row. Normal estimation using the original map. Top row. Normal estimation using the filtered map (bilateral filtering). It is observed that the normal estimation was improved resulting in more stable normal directions. This effect is visually observed on plane surfaces where the estimated normals using a noisy map were much less stable than normals computed over a filtered map.

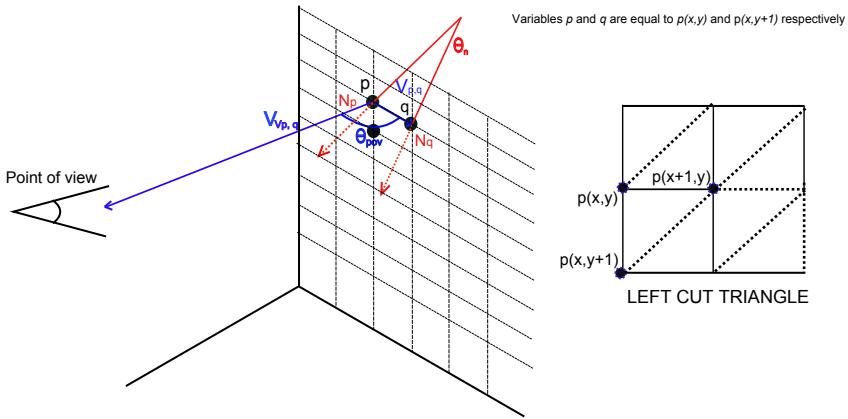
impossible to compute it in less than 1 sec. However, the GNG provides other advantages that are useful in other problems: it is capable to deal with non-organized point clouds and it also provides high accuracy recon-

structions.

In [Holz and Behnke, 2012] an efficient method to triangulate organised point clouds in real-time was presented. In this section we proposed an accelerated implementation of this method. In the original work a triangulation method for 3D points, obtained from range cameras or structured light, was proposed. Using sensors such as the Kinect, 3D points are accessed by their matrix organization using  $x$  for the row and  $y$  for the column. In this way, a 3D point  $p_{x,y}$  is accessed with a 2D indexing system. Using this representation it is possible to obtain the scene surface from the point cloud captured by the sensor. The method assumes that the viewpoint is known and in this way it is possible to calculate the angle formed by the viewpoint vector  $v_p$  and the target point  $p_{x,y}$  and the vector formed by the target point  $p_{x,y}$  and its neighbour points  $p_{x+1,y}$  or  $p_{x,y+1}$ . If points fall into a common line of sight with the viewpoint from where the measurements are taken, one of the underlying surfaces occludes the other. If all checks are passed the triangle is added to the mesh, otherwise a hole arises in the final reconstructed mesh. Moreover, if the sensor cannot acquire a valid depth measurement for a certain pixel that triangle is also rejected creating a hole. Figure 4.18 shows the proposed condition for point triangulation.

Our proposed method is more robust than the original method, as the normal information at each point of the scene is used as an additional condition for meshing the point cloud. As the triangulation of the points can be done independently, the algorithm was ported to the GPU, where each GPU thread tests the point we are targeting to form a triangle with its neighbourhood. As a result, we obtained a vector with all the triangles. Pseudo-code of the GPU-based surface triangulation algorithm is shown in Algorithm 5.

Invalid triangles are created on points which do not satisfy the proposed constraint to keep the organization of the point cloud. Finally, the condition to create an edge between two points is formulated as follows:



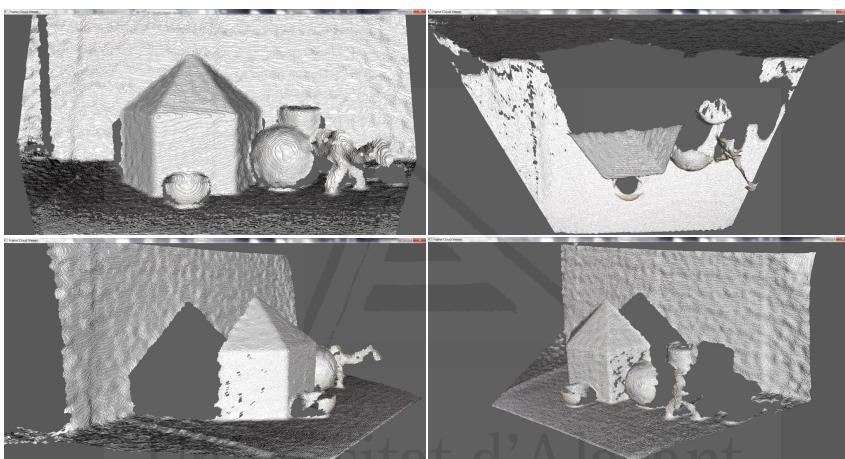
**Figure 4.18:** Left: Point triangulation condition. This image shows how the condition to create an edge establishes that the angle  $\theta_{pov}$  formed by vectors  $v_{v_{p,p}}$  and  $v_{p,q}$  must be over a threshold  $\epsilon_{\theta_{pov}}$ . This threshold assures that points are not occluded among themselves. The Euclidean distance between  $p$  and  $q$  also must be smaller than an established threshold  $T_d$ , dynamically calculated according to mesh resolution and its standard deviation. Right: Triangles are established by left cut checking constraints between points.

$$\begin{aligned} \text{edge}_{\text{valid}} = & (|v_{v_{p,p}} \cdot v_{p,q}| \leq \cos \epsilon_{\theta_{pov}}) \wedge \\ & (||p - q||^2 \leq T_d) \wedge \\ & (|n_p \cdot n_q| \leq \cos \epsilon_{\theta_n}) \end{aligned} \quad (4.3)$$

where  $\epsilon_{\theta_{pov}}$  is the angle existing between two points and the point of view establishing whether or not these points are occluded. This angle value was computed based on the visual analysis showed in the Figure 4.18 and also based on results provided in [Holz and Behnke, 2012]. The maximum distance between two points is  $T_d$ . This distance is obtained in real-time based on point cloud resolution. For that, the average distance between the targeted point and its neighbourhood  $k$  is calculated. Next, threshold  $T_d$  is given by:  $T_d = \bar{d}_k + \sigma_d$  where  $\bar{d}_k$  is the mean distance and  $\sigma_d$  is the standard deviation. Finally,  $\epsilon_{\theta_n}$  is the established threshold for the maximum angle between two normal vectors. This is calculated in the same way as  $T_d$ , obtaining an angle threshold.

The proposed method allows us to obtain fast approximate meshing

of the input point cloud. The proposed accelerated meshing method takes advantage of the knowledge about the point of view position and also takes advantage of having already calculated normal vectors on GPU memory for every point of the scene. The GPU implementation achieves run times considerably lower than the CPU. The GPU implementation achieved processing frame rates close to 30 fps for 640 by 480 depth maps while the CPU implementation achieves a frame rate close to 6 fps. Figure 4.19 shows a point cloud mesh obtained using the proposed method.



**Figure 4.19:** Point cloud meshing using the proposed method. Note that some holes and gaps still exist in the approximate surface reconstruction due to the noisy information obtained from the Kinect sensor.

### 4.5.3 Tensor computation on the GPU

Once point cloud normal information and surface triangulation are obtained, pairs of points along with their normals are selected to define local 3D coordinate basis for tensor computation. To avoid the  $C_2^n$  combinatorial explosion of the points, a distance constraint is used on their pairing. This distance constraint allows the matching between only those points that are within a previously specified distance. The distance constraint also ensures that the vertices that are paired are far enough apart so that the

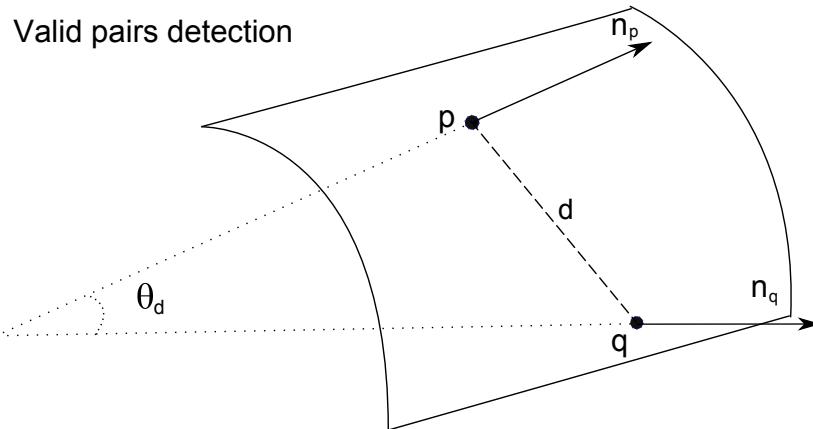
```
input : A projected point cloud  $d\_P_{xyz}$ 
input : A point cloud of normals  $d\_N_{xyz}$ 
output: List of triangles  $d\_Tri$ 

1 __global__ void;
2 gpuTriangulationKernel(  $d\_P_{xyz}$ ,  $d\_N_{xyz}$  );
3 {
4     This kernel is executed creating one thread for each point in parallel;
5     int u = threadIdx.x + blockIdx.x * blockDim.x;
6     int v = threadIdx.y + blockIdx.y * blockDim.y;
7     check constraints with neighbour points;
8     if (isValidTriangle (i, index_down, index_right));
9         addTriangle ( $d\_Tri$ );
10    if (isValidTriangle (index_right, index_down, index_down_right)) ;
11        addTriangle ( $d\_Tri$ );
12 }
```

**Algorithm 5:** Pseudo-code of the GPU-based surface triangulation algorithm.

calculation of the coordinate basis is not sensitive to noise but close enough to maximize their chances of being inside the same surface. The maximum and minimum distances between points are based on point cloud resolution, being  $d_{min} = pcl_{res} * 5$  and  $d_{max} = pcl_{res} * 14$ .  $pcl_{res}$  was calculated for each point cloud captured by the Kinect sensor in real time, allowing the movement of the sensor. In addition to this distance constraint, an angle constraint  $\theta_d$  is defined between valid pairs of points, so that points with approximately equal normals are not paired (since their cross product will result in zero). This angle between vectors must be higher than  $15^\circ$  degrees allowing the use of the mean value of these normals as an axis for the coordinate bases. Figure 4.20 shows how valid pairs are selected considering mutual angle and mutual distance constraints. Each point is paired with only its three closest neighbours, limiting the number of possible pairs to  $3n$  per view. In practice, due to the constraints this number is often lower than  $3n$ .

Pair point calculation is accelerated using as many threads as points in the point cloud, in this way each GPU thread checks its corresponding point pair with its neighbours. Moreover, the matrix organization of the point cloud is used for improving this search. In this way, each



**Figure 4.20:** Detecting a valid pair using mutual angle  $\theta_d$  and mutual distance  $d$  constraints. The midpoint of the pair will be used later to define a 3D grid for tensor computation.

thread of the GPU performs the search of valid pairs only in a defined window around the targeted point. The size of this window is based on the maximum distance constraint  $d_{max}$  and the point cloud resolution  $pcl_{res}$ :  $windows_{radius} = d_{max}/pcl_{res}$  giving the radius of the windows in pixels. As  $d_{max}$  is defined as a constant based on the  $pcl_{res}$  (millimeters), the  $windows_{radius}$  can be simplified as the constant 14. Section 4.5.4.1 presents a runtime comparison, where the CPU implementation applied the same technique for search acceleration.

Once a valid list of point pairs is obtained, a local 3D basis is defined for each valid pair in the following manner: the center of the line joining the two vertices defines the origin of the new 3D basis. The average of the two normals defines the z-axis. The cross product of the two normals defines the x-axis and finally the cross product of the z-axis with the x-axis defines the y-axis. This 3D basis is used to define a 3D grid centered at its origin. This step is also computed in parallel on the GPU for each valid pair of points.

For the grid computation, which will define the feature descriptor, it is necessary to define two additional parameters. The first one is the number of voxels that compose the grid  $n_{voxels}$  and the size of each of these voxels  $voxel_{size}$ . Modifying the number of voxels and the size of the

grid causes the obtained descriptor to contain local, semi-local or global information of the scene. In the experiments done in [Mian et al., 2006a] it was demonstrated how for the object recognition task, a size of  $10 \times 10 \times 10$  grid allows the extraction of a descriptor with semi-local information of the object allowing identification even under a high levels of occlusion. The size of the voxel  $voxel_{size}$  is defined dynamically according to the point cloud resolution. Once the grid is defined, the surface area of the mesh surface intersecting each voxel of the grid is stored in a third order tensor. This tensor is a local surface descriptor which corresponds to a semi-local representation of the object where the pair of points are lying. Sutherland Hodgman's polygon clipping algorithm [Foley et al., 1990] was used for calculating area intersections between polygons and voxels. In this way an entry is made at the corresponding element position in the tensor. Since more than one triangulated facet can intersect a single voxel, the calculated area of intersection is added to the area already present in that voxel as a result of intersection with another triangulated facet. To avoid checking all triangles that compose the scene, a growing approach is used, which starts by checking the triangles that lie in the pair of points selected and growing along its neighbourhood until all the checked triangles are not intersected with the corresponding voxel. This approach was used in both CPU and GPU versions, allowing a fair runtime comparison. Finally extracted tensors are compressed by squeezing out the zero elements and retaining the non-zero values and their index positions in the tensor. These compressed tensors together with their respective coordinate basis and the mutual angle between their normals are called a tensor representation of the view.

The computation of each tensor is considerably accelerated using the GPU because there is no dependency between the calculation of the intersected area in each voxel of the grid. Therefore,  $Dim_x \times Dim_y \times Dim_z$  threads are executed on the GPU organized as a three dimensional grid. Each thread calculates the intersected area between the mesh and its corresponding voxel, storing the calculated area in the position accessed by its indexes. See Figure 4.21. Due to the 3D index organization that the CUDA framework provides, the calculation of corresponding indexes is

greatly accelerated. Sutherland Hodgman's polygon clipping algorithm is also executed by each thread in parallel. Pseudo-code of the GPU-based 3D tensor computation algorithm is shown in Algorithm 6. Additionally, there is also no dependency between the computation of different tensors, thereby the computation of different tensors is overlapped occupying all the available resources on the GPU. Performance results are shown in Section 4.5.4.1.

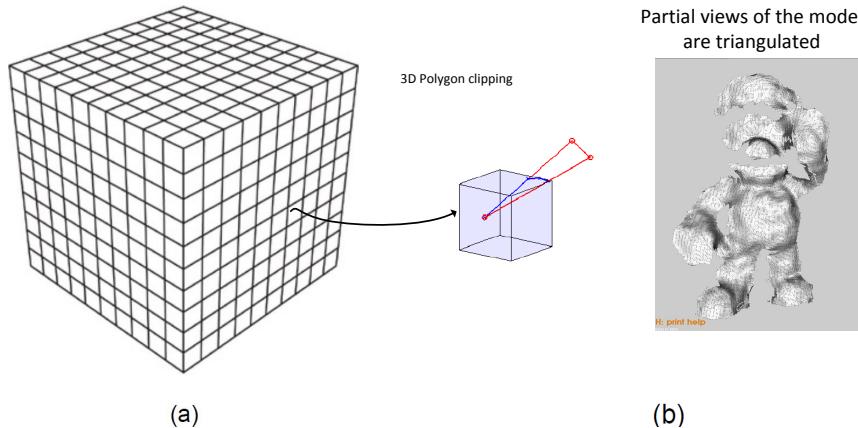
```

input : A projected point cloud  $d\_P_{xyz}$ 
input : A valid pair of points  $d\_N_{xyz}$ 
input : List of triangles  $d\_Tri$ 
output: 3D tensor  $t_i$ 

1 global void;
2 gpuTensorCompKernel(  $v_i, d\_P_{xyz}, d\_N_{xyz}, d\_Tri$  );
3 {
4     This kernel is executed creating one thread for each bin of the grid in parallel;
5     int x = threadIdx.x + blockIdx.x * blockDim.x ;
6     int y = threadIdx.y + blockIdx.y * blockDim.y ;
7     int z = threadIdx.z + blockIdx.z * blockDim.z ;
8     binLimits = computeBinLimits(cloud,tri) ;
9      $d\_Neigh\_Tri$  = compIndexNeighTriangles() ;
10    calculate area that clip with the corresponding bin;
11    for  $i \leftarrow 0$  to  $|d\_Neigh\_Tri|$  do
12        area += clipTriangle(cloud,tri,binLimits) ;
13    end
14     $t_i[x][y][z] = area$  ;
15 }
```

**Algorithm 6:** Pseudo-code of the GPU-based 3D tensor computation algorithm

All extracted tensors from 3D observation (scene, object, etc) are stored with their coordinate basis allowing the use of this information for grouping all tensors with similar angle between their normals. In this way an efficient matching is possible for different applications such as partial view registration and object recognition. This collection of tensors is stored during a training phase creating a hash table for efficient retrieval during test phase.



**Figure 4.21:** (a) Launching  $Dim_x \times Dim_y \times Dim_z$  threads in parallel where each GPU thread represent a voxel of the grid. (b) Each thread with indexes  $i, j, k$  calculates the area of intersection between the mesh and its corresponding voxel using Sutherland Hodgman’s polygon clipping algorithm. Taking advantage of thread indexes, the calculated area is stored in a flattened vector.

#### 4.5.4 Experimental results

GPU version of the proposed method described in this section were tested on a desktop machine with an Intel Core i3 540 3.07Ghz and different CUDA capable devices. GPU implementations were first developed on a laptop machine equipped with an Intel Core i5 3210M 2.5 Ghz and a CUDA compatible GPU. Table 4.7 shows different models that have been used and their main features. We used different models ranging from the integrated GPU on a laptop to a more advanced model, demonstrating that the GPU implementations can be executed on different GPUs and that they obtains good execution times on different graphic boards with different number of cores.

Device Model	CUDA cores	Global Mem	Bandwidth Mem
Quadro 2k	192	1 GB	41.6 GB/s
GeForce GTX 480	480	1.5 GB	177.4 GB/s
GeForce GT630M	96	1 GB	32 GB/s

**Table 4.7:** CUDA capable devices used in experiments

#### 4.5.4.1 Performance

The performance obtained by the GPU implementation allowed us to compute the proposed methods under real-time constraints. Table 4.8 shows the different steps that were accelerated using the GPU and their different runtime and the speed-ups achieved for the different GPU devices. The obtained acceleration is relative to a CPU implementation of the proposed method. In general the best performance was obtained with the graphics board with the largest number of CUDA cores (GTX480) and the largest memory bandwidth.

These results demonstrated how the proposed methods are suitable for massively parallel architectures such as the GPU, where each thread processes one of the points of the scene. Another interesting aspect of the results shown in Table 4.8 was that GPU implementations allowed us to compute operations that are prohibitively slow on the CPU in real-time such as normal estimation, keypoint detection or surface triangulation. Moreover, Table 4.8 shows how the entire computation of 200 tensors in the GPU was performed in less than 0.5 seconds for the fastest device achieving a 93x performance boost related to the CPU implementation. This allowed the computation of the descriptor at different points of a scene in real-time.

Another remarkable aspect of the performance obtained for the overall system is that tensor computation is not only parallelized at thread level, it is also parallelized at task level computing simultaneously different tensors. As tensor computation is not dependent, it can be parallelized using different CUDA streams on the GPU. This technology allows executing in parallel as many kernels as possible in different queues and therefore allows to exploit available resources on the GPU [Wang et al., 2011]. We decided to exploit the possibility of launching multiple kernels concurrently using CUDA streams, overlapping the paradigm of task parallelism to that of data parallelism. In order to analyse and confirm stream parallel execution we profiled the algorithm using the NVIDIA Visual Profiler [NVIDIAb, 2013], which allows to visually appreciate how stream computation is performed along the time and also multiprocessors occupancy on the GPU. Figure 4.22 shows algorithm computation timeline using and not

Step	GT630M time	GTX480 time	Q2k time	CPU time
Bilateral filtering of depth map	11ms	<b>5ms</b>	8ms	1008 ms
Point cloud projection	2ms	<b>1ms</b>	1ms	50ms
Normal estimation	9ms	<b>1ms</b>	8ms	190ms
Compute surface triangulation	5ms	<b>2ms</b>	4ms	121ms
Compute cloud resolution	7ms	<b>4ms</b>	6ms	330ms
Compute valid pairs	71ms	<b>9ms</b>	35ms	4479ms
Compute third order tensor	6ms	3ms	4ms	130 ms
Total GPU time for extracting 200 tensors	854 ms	<b>490ms</b>	687ms	45887ms

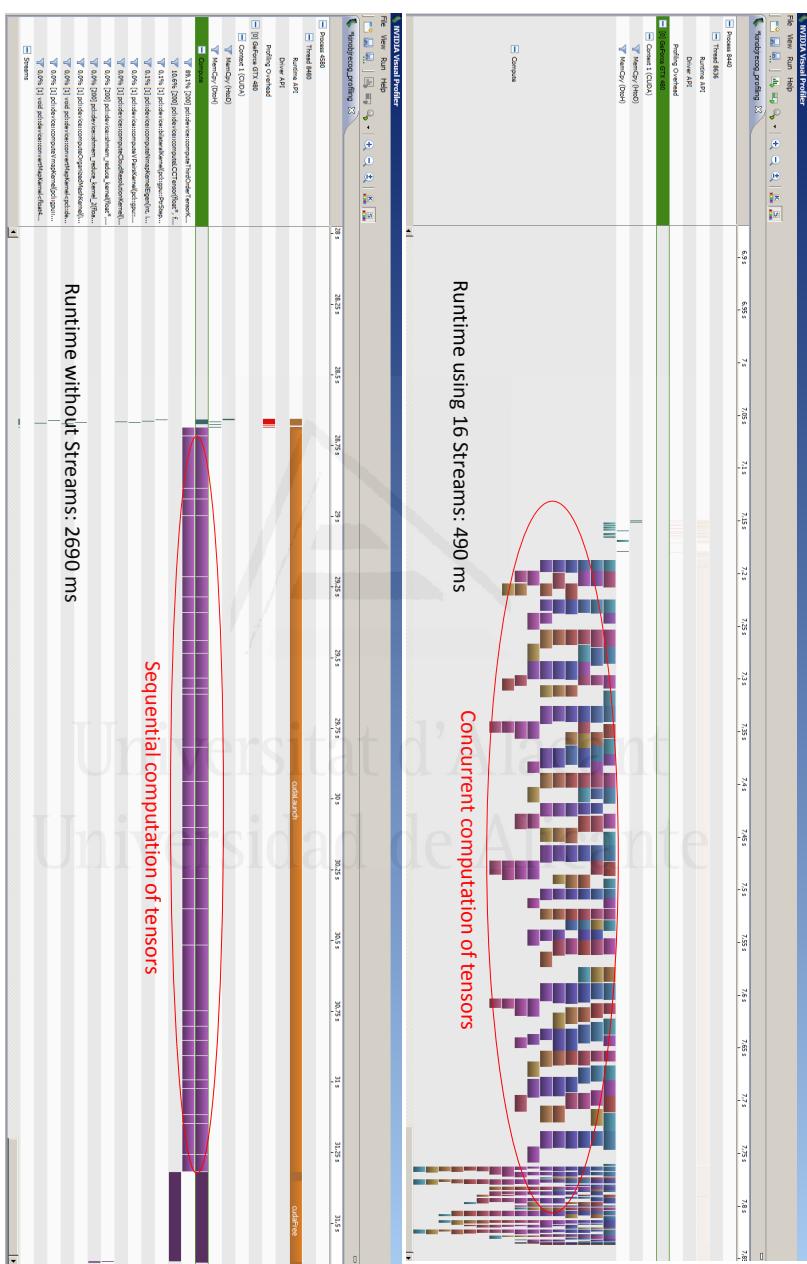
Step	GT630M Speed-up	GTX480 Speed-up	Q2k Speed-up
Bilateral filtering of depth map	91.63x	<b>201.6x</b>	126x
Point cloud projection	25x	<b>50x</b>	50x
Normal estimation	21.11x	<b>190x</b>	23.75x
Compute surface triangulation	24.25x	<b>40.3x</b>	30.25x
Compute cloud resolution	47.14x	<b>82.5x</b>	55x
Compute valid pairs	63x	<b>497x</b>	127.97x
Compute third order tensor	31.6x	<b>43.33x</b>	32.5x
Total GPU time for extracting 200 tensors	53.72x	<b>93.64x</b>	66.79x

**Table 4.8:** Runtime comparison and speed-up obtained for proposed methods using different graphics boards. The fastest run times were achieved by the graphics board NVIDIA GTX480. Runtimes are averaged over 50 runs.

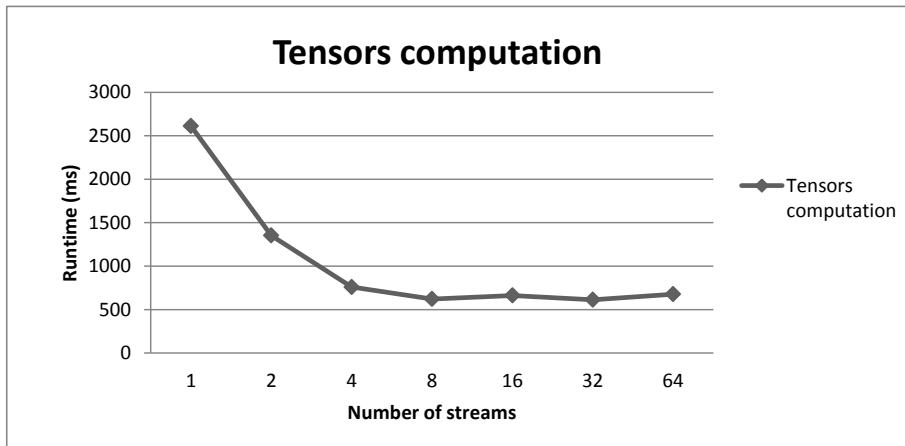
using streams for overlapping computations. Runtime execution and multiprocessors occupancy was greatly improved thanks to concurrent kernel execution using streams. Runtime was improved by a speed-up factor of 5x overlapping tensors computation with many kernels enqueued in different streams and launched concurrently.

We set the number of streams to 16 after performing several experiments with different number of streams. Figure 4.23 shows how the runtime was improved as the number of streams was increased obtaining maximum performance and occupancy on the GPU using values larger than 4. Indeed, Figure 4.22 shows how the maximum number of tensors that were calculated simultaneously was 8 without taking into consideration the maximum number of streams specified. This happened due to the occupancy of all the available resources by the kernels running concurrently.

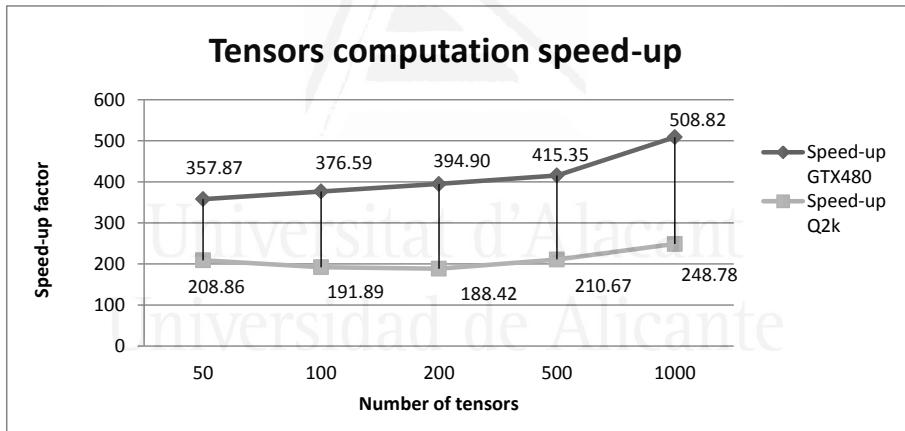
Finally, Figure 4.24 shows an experiment computing different number of tensors. From results presented in Figure 4.24 we concluded that the speed-up obtained by the GPU version was increased as the number of tensors was also increased achieving a larger speed-up factor. Computing times obtained using the CPU version were prohibitive for time-constrained applications.



**Figure 4.22:** Profiling computation of tensors using streams (top) and without streams (bottom). On the top of the figure (using streams) it can be seen how up to 6 kernels run simultaneously occupying all available resources on the GPU.



**Figure 4.23:** Tensors computation runtime using different number of streams. Number of tensors is fixed to 200 and the device used is the NVIDIA GTX 480. Using CUDA streams runtime was considerably improved.



**Figure 4.24:** Speed-up achieved compared to sequential CPU version for the computation of a different number of tensors.

## 4.6 Conclusions

This chapter has demonstrated that the GPGPU paradigm allowed to considerably accelerate algorithms traditionally executed on the CPU and often to run these under hard time constraints.

Moreover, this chapter proposed a GPU-based implementation of the

GNG algorithm in order to obtain a more efficient version suitable for operations with time constraints. Experimental results showed that the GPU implementation significantly reduced learning time compared with single-threaded and multi-threaded CPU implementations of the GNG. Furthermore, an hybrid implementation was proposed combining CPU and GPU and achieving a speed-up factor up to  $6x$  compared to the sequential CPU implementation for a network containing 20,000 neurons and performing  $1000\lambda$  adjustments per iteration. In addition, the NG algorithm was also successfully accelerated on the GPU obtaining a speed-up factor of  $15x$  compared to the sequential CPU version for a network containing also 20,000 neurons. Larger speed-up factors were achieved for larger number of neurons, up to  $95x$  for 50,000 neurons. It was demonstrated that the NG algorithm is more computationally demanding than the GNG and also it is intrinsically more parallel than the GNG algorithm. In fact, the overall runtime for the NG learning is larger compared to GNG runtime. However, GPU-based implementation of the NG algorithm achieved a higher acceleration factor.

Finally, some advantages were obtained by the use of the GPU to accelerate the computation of a 3D descriptor based on the calculation of 3D semi-local surface patches (3D local shape), thus allowing its computation at several points of a scene in less than 0.5 seconds. Within the 3D data algorithms used in the proposed pipeline for the descriptor computation, some progress were made towards a faster and more robust point cloud triangulation, normal estimation and in general in the integration of 3D processing algorithms on the GPU.

---

## Chapter 5

---

# Applications

---

In this chapter are presented different cases of study where the GPU-based 3D version of the GNG algorithm and the accelerated 3D descriptor proposed in this PhD thesis were successfully applied. Section 5.2 presents a 6DoF pose registration application commonly used in robotics where the model created by the GNG algorithm has been used to improve the accuracy of the method, improving the reconstruction process. Section 5.3 shows a real-time 3D object recognition application where the GPU-based implementation of the proposed descriptor in Section 4.5 was successfully applied. Section 5.4 shows a prototype of a surveillance multi-camera system where multiple GPU-based GNG implementations are executed on different GPUs and applied to different video streams performing people tracking. Finally, in Section 5.5 and 5.6 the GNG is used for 3D modelling tasks in the CAD/CAM area. First, topology preservation capabilities of the GNG where used for landmarking noisy scans of different shoe lasts and then performing accurate 3D reconstructions. Moreover, the GNG extension for surface reconstruction is directly applied to raw data captured using the Kinect sensor to produce 3D object models that can be used in virtual reality applications.

## 5.1 Introduction

In this chapter are presented different cases of study where the GNG-based method proposed in this PhD thesis was successfully applied to different areas. Within these areas we find robotics, computer vision or CAD/CAM, helping to solve some state-of-the-art problems that most researchers are still facing. In most cases, the improved 3D GNG method has been used as a representation model, this intermediary representation helped to carry out post-processing steps in a more efficient and effective way. Moreover, the real-time implementation of the 3D descriptor presented in Section 4.5 was successfully applied to a 3D object recognition application. These cases of study are detailed in next sections.

## 5.2 Robotics: 6DoF pose registration

In the last years I have been involved in different projects with the Robotics and 3D Vision research group of the University of Alicante. Within one of these projects it was demanded a representation model capable of dealing with noise produced by typical 3D sensors as Time of Flight (ToF) and RGB-D cameras. The main goal of this project was to achieve cooperative Simultaneous Localization and Mapping (SLAM) in large scale environments. Moreover, as the SLAM problem involves solving other problems as the registration of multiple scene views, down-sampling and filtering steps become crucial for the establishment of a valid matching during this process. Therefore, a 3D representation model able to cope with all these mentioned constraints and able to keep the geometry of real-word scenes is required.

### 5.2.1 Accelerating 6DoF egomotion using GNG

In this section, we show an application where the use of the accelerated GNG improved its solution. The main goal of this application is to perform six degrees of freedom (6DoF) pose registration in semi-structured environments, i.e., man-made indoor and outdoor environments. This registration provides a good starting point for Simultaneous Location and

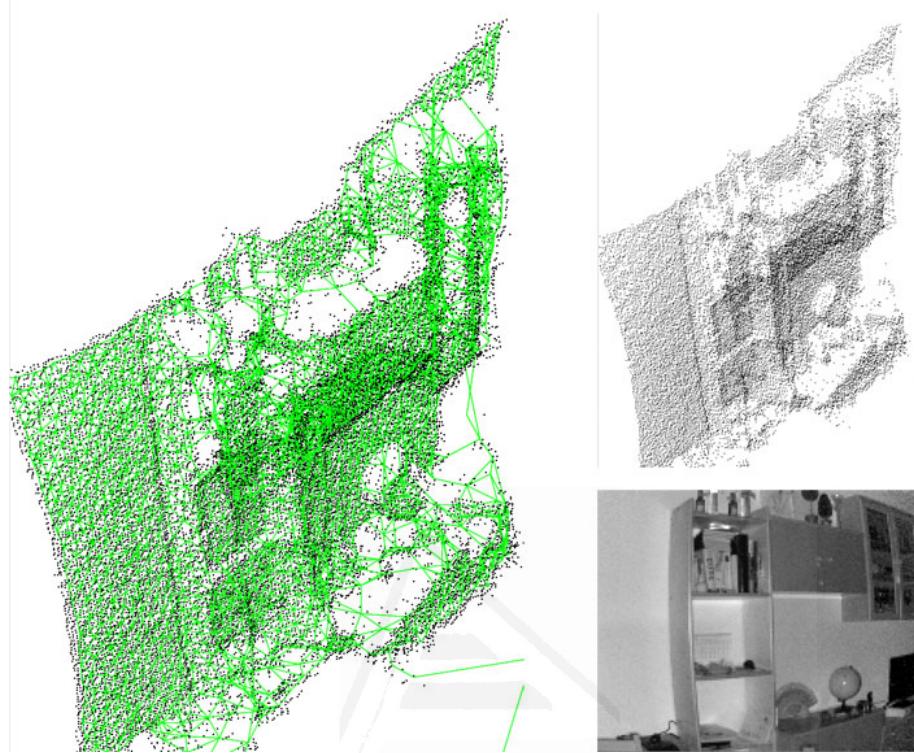
Mapping (SLAM).

We combined our accelerated GNG-based algorithm with the method proposed in [Viejo and Cazorla, 2013]. For our experiments, we have used data from an infrared time-of-flight camera SR4000, but in that work there are examples of this method applied to other 3D devices, like a sweeping unit with a 2D laser Sick and a Digidlops stereo camera, mounted on a mobile robot. This method was developed to manage 3D point sets collected by any kind of sensor.

We were also interested in noise reduction and outliers removal, i.e., environments with people or non-modeled objects. This task is hard to overcome as classic algorithms, like the Iterative Closest Point algorithm (ICP) [Besl and McKay, 1992] and its variants, are very sensitive to outliers. Finally, the huge amount of data makes necessary the acceleration of the overall process in order to obtain the results in real time. The GNG method produces a 3D Delaunay Triangulation which was used as a representation of the points neighbourhood. GNG was applied directly to 3D data. Figure 5.1 shows the result of applying the GNG algorithm to 3D points from a SR4000 camera.

Here we briefly describe the method proposed in [Viejo and Cazorla, 2013] to manage 3D data and to use it for 6DoF egomotion calculation. In that work, a feature extraction process is applied to the raw 3D data in order to obtain a complexity reduction. These features are planar patches which are models representing surfaces from the 3D data. This feature extraction method is based on neighbour searching. We can improve and accelerate the neighbour searching using the GNG structure as it produces a more detailed and accurate planar patches descriptions. Figure 5.2 shows planar patches extraction from a 3D image obtained by a SR4000 camera. The bottom image shows the results of combining GNG with the features extraction procedure. It is compared with the top image in which no GNG has been used. The more number of planar patches we have, the more accurate result we obtain.

In particular, 3D scene reconstruction is a time consuming task, that is fundamental in most mobile robotic systems [May et al., 2009, Hähnel et al., 2002, Grisetti et al., 2007, Weingarten et al., 2004, Rusu et al.,

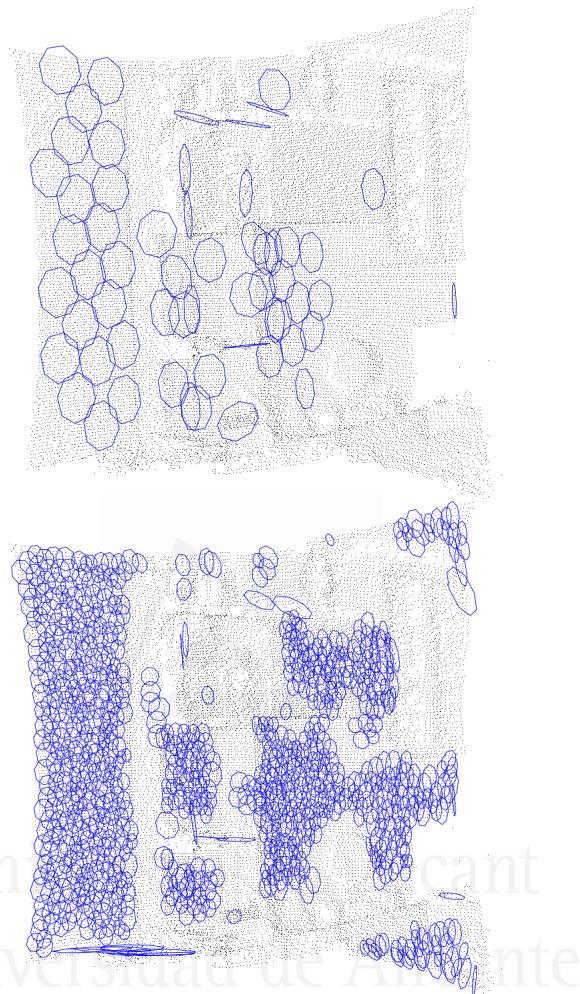


**Figure 5.1:** Applying GNG to SR4000 data. The original data set is shown at the top right corner. The intensity image captured by the camera is shown at the bottom right corner. Left, the resulting GNG (green) can be observed over the original data set.

2008b, Steder et al., 2009, Steder et al., 2010]. However, most of these works do not deal with real time restrictions.

To validate our work we applied our GNG accelerated implementation to the extraction of features from 3D raw data [Viejo and Cazorla, 2013, Villaverde and Graña, 2009, Muñoz-Salinas et al., 2008, Katz et al., 2010]. Moreover, using this method, apart from accelerating the overall runtime, we achieved two other advantages: a complexity reduction (when comparing with raw data) and an improvement of speed-up in the extraction step without decreasing the quality of the obtained representation.

For this reason, we would like to use these models to achieve further mobile robot applications in real 3D environments. The basic idea is to take advantage of the extra knowledge that can be found in 3D models

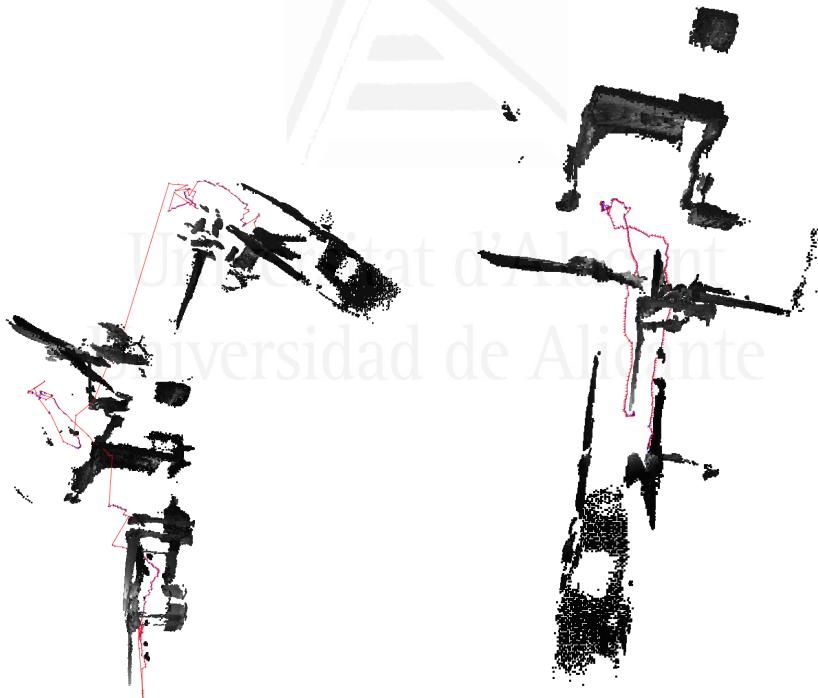


**Figure 5.2:** Top, planar patches extracted from SR4000 camera. Bottom, use of GNG to improve planar patches extraction.

such as surfaces and its orientations. This information is introduced in a modified version of an ICP-like algorithm in order to reduce the outliers incidence in the results. ICP [Besl and McKay, 1992] is widely used for geometric alignment of a pair of three-dimensional points sets. From an initial approximate transformation, ICP iterates the next three steps until convergence is achieved: first, closest points between sets are stated; then, best fitting transformation is computed from paired points; finally, transformation is applied. In mobile robotics area, the initial transformation

usually comes from odometry data.

Nevertheless, [Viejo and Cazorla, 2013] did not need an initial approximate transformation as ICP based methods do. We used the global model structure to recover the correct transformation. This feature is useful for those situations where no odometry is available, or it is not accurate enough, such as legged robots. In our case, we exploit both the information given by the normal vector of the planar patches and its geometric position. Whereas original ICP computes both orientation and position at each iteration of the algorithm, the method used can take an advantage of the knowledge about planar patches orientation for decoupling the computation of rotation and translation. They first register the orientation of planar patches sets and when the two planar patches sets are aligned we address the translation registration.



**Figure 5.3:** Planar based 6DoF egomotion results. Left image shows map building results without using GNG while the results shown on the right are obtained after computing a GNG mesh.

In Figure 5.3, we show an example of 3D map building using this

6DoF egomotion approach. For this experiment, 100 3D images from a 5 meter range SR4000 camera were used. The image on the left shows a 3D view of the reconstructed environment using 6DoF egomotion from planar patches. In the right image, the same scene is reconstructed but GNG was used to improve features extraction. While in the first experiment the registration of the sequence was almost impossible, in the second one the reconstruction was reasonably good. Computing time for obtaining planar patches descriptions after applying GNG is almost the same as without GNG and is about 300 ms per image.

The use of GPU acceleration provides a lower reconstruction time per each data acquisition, 50 ms for an adjustment of a neural network composed by 20.000 neurons and 1000  $\lambda$  patterns. This makes our system suitable to deal with time constraints.

### 5.3 Computer Vision: 3D object recognition

In this section, we show an application where the use of the accelerated semi-local surface feature extraction process presented in Chapter 4 allows to detect and recognize objects under cluttered conditions in real-time. The main goal of this application is the recognition of objects under real-time constraints in order to integrate the proposed algorithm in mobile robotics. Our method is designed to use only depth information since most computer vision systems need to work under poor or no illumination conditions.

We tested the proposed feature in a similar application as it was done in the original work [Mian et al., 2006a] where the semi-local surface feature is successfully used to recognize objects in cluttered scenes. For our experiments we have captured data from a Kinect sensor and tested the accelerated feature with some cluttered scenes. To do that, first a small library of models is constructed offline, storing all extracted 3D tensors in an efficient way using a hash table. Next, online object recognition is performed using cluttered scenes. Although the accelerated feature is tested using 3D data obtained from the Kinect sensor, this method is developed for managing 3D point sets collected by any kind of sensor and could be

extended to other datasets.

We created a toy dataset to validate our proposal since the main goal of this work is to achieve real-time performance and integrate 3D data processing onto the GPU. Further analysis on recognition rates and feature parameters are already presented in the original work [Mian et al., 2006a].

### 5.3.1 Offline learning

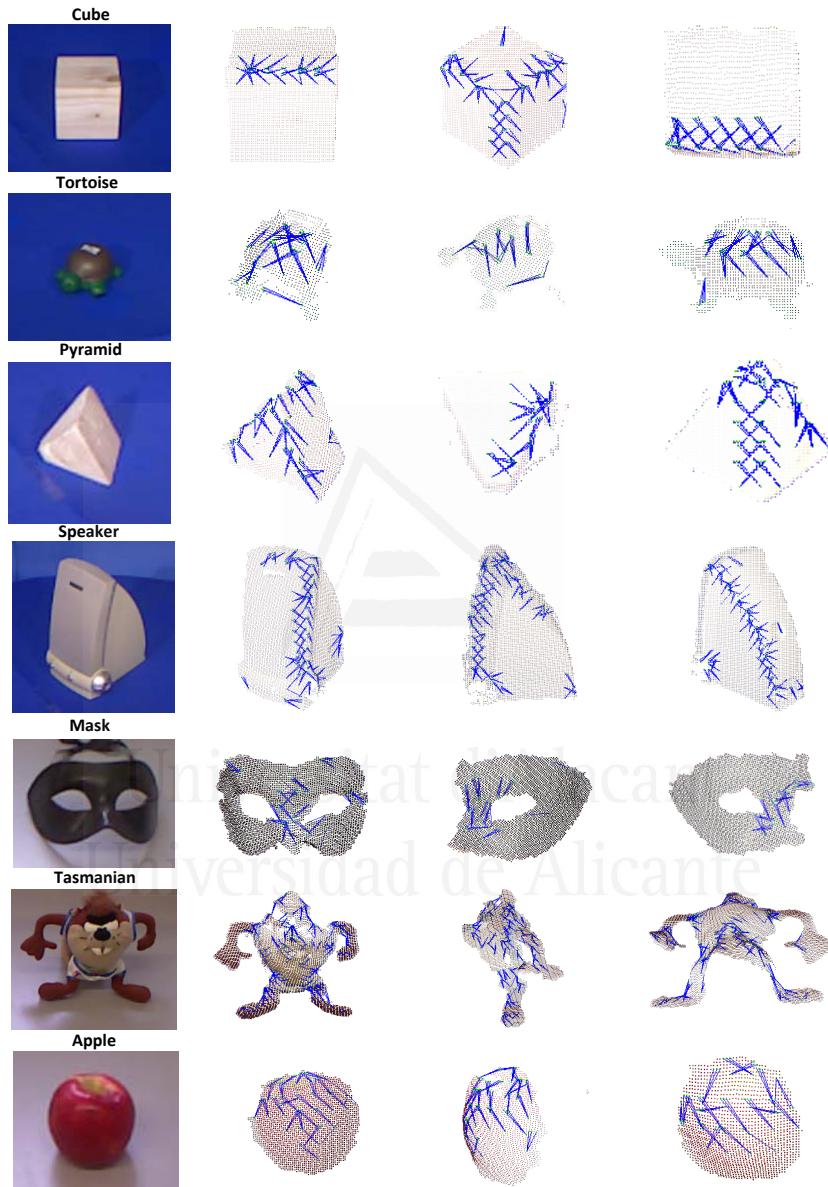
To recognize objects using our real-time tensor extraction algorithm, first a model library is built extracting tensors from different views of free-form objects. Each partial view is represented with tensors and they are stored in an efficient way for being used after in an online recognition phase. Figure 5.4 shows some partial views of the models used to build the model library. Moreover, Figure 5.4 shows tensors extracted for some of the views. For each of these views the process explained in Section 4.5.1 was computed, obtaining as a result a set of tensors that describe each partial view.

Since multi-view correspondence using linear matching methods algorithms would be unaffordable, a hash map is introduced using the angle  $\theta_d$  of the tensors as a key for the retrieval. The hash map is quantized in bins of  $\theta_b$  degrees obtaining a good balance of tensors per bin and boosting the query performance. For this application the hash map is quantized into bins of 5 degrees.

In contrast to the original implementation of this kind of hash map presented in [Mian et al., 2006a] and to integrate the matching process onto the GPU pipeline, the tensor library is stored in the GPU memory performing the tensors matching in parallel on the GPU and considerably accelerating its performance.

### 5.3.2 Online recognition

Once the model library is built and loaded in the GPU memory, the application is ready to start recognizing objects from a scene captured in real-time. Therefore, the input to our application is a point cloud of a scene. The point cloud is processed following the pipeline presented in



**Figure 5.4:** Model library consisted of 7 real models. Each object consists of several partial views. For every partial view of a model, tensors are computed (blue lines) describing the model by extracting 3D surface patches (tensors).

Section 4.5.1. Once tensors from the scene are computed these are matched against the model library previously stored on the GPU memory. The matching process, as it was introduced in previous section, is performed in parallel onto the extracted tensor using the angle as a key for the hash map. For the extracted tensor there are launched as many threads as tensors are stored in that bounded bin. Next, correlation coefficient originally presented in [Mian et al., 2006a] are computed in parallel. The correlation coefficient measures the similarity between the scene tensors and possible candidates stored in the model library. The correlation coefficient in the overlapped area between two tensors is calculated as follows:

$$C_c = \frac{n_q \sum_{i=1}^{n_q} p_i q_i - \sum_{i=1}^{n_q} p_i \sum_{i=1}^{n_q} q_i}{\sqrt{n_q \sum_{i=1}^{n_q} p_i^2 - (\sum_{i=1}^{n_q} p_i)^2} \sqrt{n_q \sum_{i=1}^{n_q} q_i^2 - (\sum_{i=1}^{n_q} q_i)^2}} \quad (5.1)$$

where  $p_i$  and  $q_i$  ( $i = 1 \dots n_q$ ) are the respective elements of the model tensor  $T_m$  and scene tensor  $T_s$  in their region of overlap. Matchings whose  $C_c < t_c$  are discarded ( $t_c = 0.45$  based on results presented in [Mian et al., 2006a]). The remain tensors are considered as possible correspondences.

Once all correlation coefficient has been calculated in parallel for a scene tensor  $T_s$  and considered as true possible correspondences, minimum correlation coefficient value is found and considered as the true correspondence. The reduction operation [Harris, 2008] to obtain the minimum value in parallel is also performed on the GPU pipeline using traditional divide and conquer approach to find the minimum value.

### 5.3.3 3D object recognition results

We performed out different experiments where the accuracy and the performance of the proposed GPU implementation are studied. The algorithm is initially validated on scenes with a single object. Then, a more complex study on noise scenes with multiple objects and occlusions is performed.

### 5.3.3.1 Recognition on scenes with a single object

In this first experiment, single views of each object model are used for testing descriptor recognition accuracy. Tested scenes only contained the object, so there are no occlusions caused by other objects, only self-occlusions. The experiment was performed on different views of each object. Different views from arbitrary viewing directions were selected. The confusion matrix showing the performance and recognition rates for extracted tensors is shown in Figure 5.5. The total rate of tensor recognition obtained for the experiment was 84% (TP). The rate of False Positives (FP) was 16% and False Negatives (FN) 0%. It is important to notice how some tensors extracted from tortoise views were wrongly assigned to the apple model and vice versa. This was caused by the lack of information of some tortoise views from these viewing directions and the similar geometric shape they have. It is also important to notice how observations from the top of two objects may be visually similar as occurs with the front view of the pyramid and the speaker. Therefore, some tensors extracted from the speaker views were wrongly classified as the pyramid and cube model.

### Universitat d'Alacant

### 5.3.3.2 Recognition on scenes with multiple objects and occlusions

In this experiment, we used objects from the previously constructed dataset and other non-stored objects. There were multiple objects occluding each other and causing clutter in the scene. Constructed scenes were used as input data for the proposed GPU implementation in order to perform object recognition. Moreover, no prior information was provided to the algorithm regarding models placed in the scene. In Figure 5.6 (Top) it is shown a first experiment with occlusions where three different objects (2 pyramids and a tortoise) are occluded by two objects non-stored in the database. In this experiment different percentage of occlusions are considered. We defined occlusion according to the next formula as it was done in a similar way in the original work:

		tortoise	speaker	pyramid	cube	mask	tasmanian	apple
tortoise	v1	97.62	0.00	0.00	0.00	0.00	2.38	0.00
	v2	90.00	8.00	0.00	0.00	0.00	0.00	2.00
	v3	83.33	3.33	0.00	3.33	0.00	6.67	3.33
	v4	77.94	0.00	0.00	2.94	0.00	2.94	16.18
	v5	92.98	0.00	1.75	0.00	0.00	0.00	5.26
	v6	92.16	0.00	0.00	3.92	0.00	0.00	3.92
speaker	v1	0.00	60.00	17.78	6.67	0.00	2.22	13.33
	v2	0.00	96.70	2.20	1.10	0.00	0.00	0.00
	v3	0.00	73.33	6.67	6.67	0.00	6.67	6.67
	v4	0.00	94.69	0.00	0.00	0.97	2.42	1.93
	v5	1.89	76.42	9.43	2.83	0.00	5.66	3.77
	v6	2.05	94.87	0.51	0.00	0.00	2.05	0.51
pyramid	v1	9.43	13.21	64.15	11.32	0.00	1.89	0.00
	v2	0.00	11.11	85.71	1.59	0.00	0.00	1.59
	v3	0.00	4.69	84.38	9.38	0.00	0.00	1.56
	v4	0.00	3.77	92.45	0.00	0.00	0.00	3.77
	v5	0.00	11.94	80.60	4.48	0.00	2.99	0.00
	v6	6.06	27.27	60.61	0.00	0.00	6.06	0.00
cube	v1	0.00	7.06	0.00	92.94	0.00	0.00	0.00
	v2	0.00	14.29	3.57	82.14	0.00	0.00	0.00
	v3	0.00	5.80	0.00	92.03	0.00	0.72	1.45
	v4	0.00	1.53	0.00	96.95	0.00	0.00	1.53
	v5	0.00	13.73	13.73	58.82	0.00	0.00	13.73
	v6	0.00	2.53	0.00	0.00	92.41	5.06	0.00
mask	v1	0.00	3.77	0.00	0.00	86.79	3.77	1.89
	v2	7.69	0.00	0.00	0.00	89.74	0.00	2.56
	v3	2.17	8.70	2.17	0.00	86.96	0.00	0.00
	v4	9.68	4.84	0.00	0.00	77.42	3.23	4.84
	v5	7.32	2.44	7.32	0.00	75.61	2.44	4.88
	v6	0.44	0.00	3.07	0.00	1.75	90.79	3.95
tasmanian	v1	1.64	0.00	1.97	2.30	3.93	90.16	0.00
	v2	1.40	0.00	1.40	0.00	2.33	90.70	4.19
	v3	2.83	0.00	8.49	0.00	0.00	88.68	0.00
	v4	0.39	0.00	2.36	0.00	1.18	95.28	0.79
	v5	1.71	0.85	3.85	0.00	0.00	93.59	0.00
	v6	8.62	0.00	0.00	1.72	1.72	87.93	
apple	v1	25.81	0.00	0.00	1.61	3.23	69.35	
	v2	17.86	0.00	3.57	1.19	9.52	0.00	67.86
	v3	3.13	3.13	9.38	0.00	3.13	0.00	81.25
	v4	8.00	4.00	6.00	0.00	8.00	2.00	72.00
	v5							

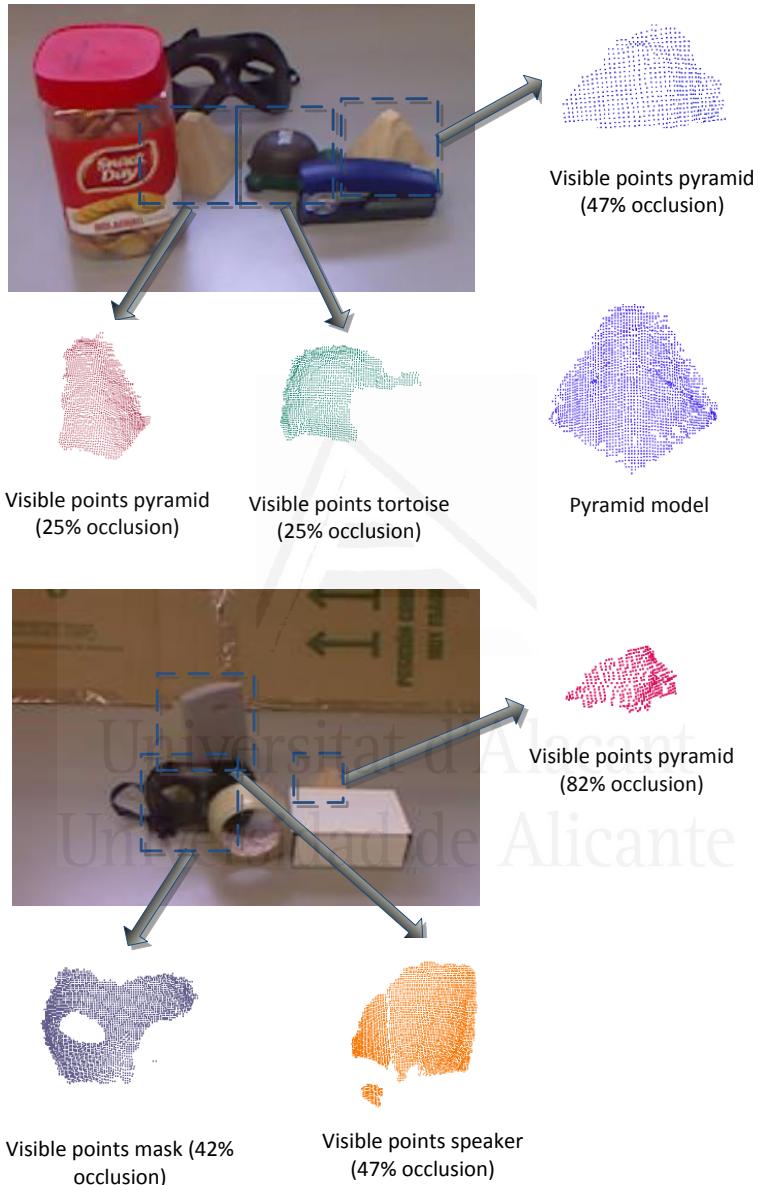
**Figure 5.5:** Confusion matrix for extracted tensors from different views of each object model contained in the dataset. The averaged recognition rate (TP) is 84%, wrong matches (FP) 16% and false negatives (FN) 0%. Cells numbers indicate the number of times (percentage) that an extracted tensor was successfully classified to the corresponding model. Presented results show that method provides high discrimination capability.

$$occlusion = 1 - \frac{\text{object points visible in the scene}}{\text{total object points}} \quad (5.2)$$

From all the extracted tensors from the scene presented in Figure 5.6 (Top) the recognition rate (TP) achieved was 85 percentage, only being wrongly classified the non-stored objects. They were classified as the cube and speaker model as the flat sides of the observation resembles to the cube model. All extracted tensors from the occluded objects were successfully classified. In addition, this experiment shows how the proposed implementation can differentiate between multiple instances of the same object, while other techniques [Johnson and Hebert, 1999] are only able to detect single instances in the scene. In Figure 5.6 (Bottom) an scene with different levels of occlusion is presented. The recognition rate achieved for this scene was 76%. As the level of occlusion is increased some objects as the pyramid which has a level of occlusion close to 82% was wrongly classified, but the rest of objects in the scene were successfully classified. Moreover, some objects present in the scene but non-stored in the library as the paper box was classified as the cube model. After testing the GPU implementation with a total of 8 scenes (Figure 5.7) with different levels of occlusion the average recognition rate achieved was 82%. We noticed that the capability to recognize objects with different levels of occlusion was related to the part of the object that was visible and if this part of the object had areas with high discriminant features. Moreover, the area of overlap for the correlation coefficient between two tensors was established to 45%. If higher levels of occlusions are demanded this factor could be set to a lower values allowing the detection of tensors with higher level of occlusions, but this would considerably affect to the recognition rate, classifying a higher number of tensors in the wrong class.

Finally, Figure 5.8 shows two arbitrary scenes computed using the proposed method. Tensors are calculated randomly over the scenes and matched tensors are labelled with the closest model in the library. Multiple labels are shown in Figure 5.8 as all tensors present in the scene are evaluated and matched against the library model. Voting strategies within clusters may be performed in order to further accelerate the object

### Scenes with multiple objects and occlusions



**Figure 5.6:** Object recognition is performed on scenes with different level of occlusion. The models are occluded by objects stored and non-stored in the library.



**Figure 5.7:** Scenes with multiple object and occlusions. These scenes were used in experiments presented in Section 5.3.3.2 achieving a recognition rate of 82%. Scenes presented a different level of clutter and occlusions caused by stored and non-stored objects.

recognition process. In addition, since the main goal of this work is to achieve real-time geometric feature extraction, the final hypothesis verification step has not been implemented yet on the GPU, so it remains as a future work. This is the reason why multiple labels are drawn.

The number of tensors evaluated over the scene is 200 as experiments have demonstrated that evaluating over this number in most of cases achieves the recognition of all objects in the scene. A similar study was made in the original work [Mian et al., 2006a]. Some wrong labelling appears in Figure 5.8 (top) for the speaker as the partial view of the scene does not have enough geometric information to find a correspondence in the database. However, in the Scene 2 (bottom) as the partial view contains more geometric information of the speaker, it is correctly recognized. For other objects as the tortoise, cube and pyramid, as similar views of the objects are present in the database and partial view of the scene has enough geometric information, the algorithm does correctly find tensors that match the model stored in the library. Future hypothesis verification and feature grouping steps will improve the classification accuracy of the object recognition task.

### 5.3.3.3 Performance

In this section, some experiments related to the performance of the parallel matching performed on the GPU are presented, comparing performances obtained by the GPU and CPU versions.

<b>Model library size (tensors)</b>	2000	4000	16000	64000
<b>Runtime CPU (ms)</b>	215	398	1589	6414
<b>Runtime GTX480 (ms)</b>	75	140	534	2130
<b>Speed-up GTX480</b>	2.87x	2.84x	2.98x	3.01x

**Table 5.1:** Runtime comparison and speed-up obtained for matching process. As the size of the model library is increased the speed-up achieved is slightly larger. Runtimes are averaged over 50 runs.

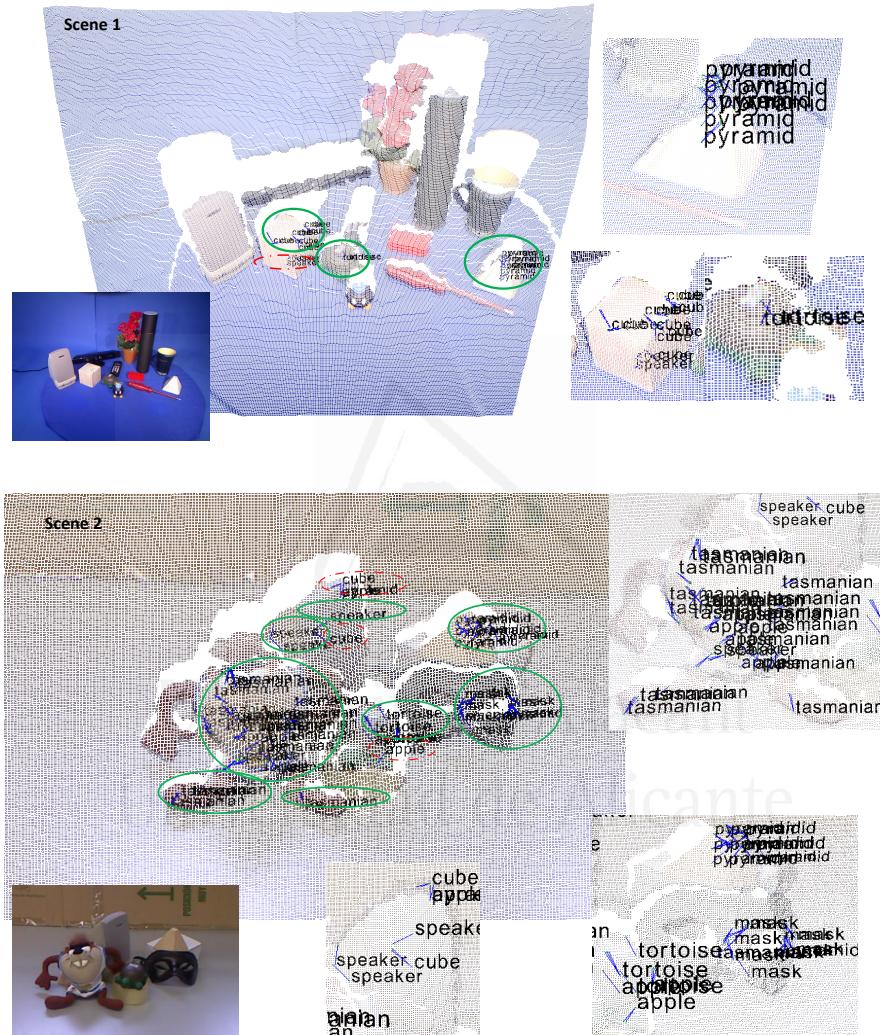
Table 5.1 shows how the matching process is computed faster on the GPU. The speed-up achieved is close to 3x but the most important is that the computing matching process on the GPU also avoid transferring data

back to the CPU side after computing tensors on the GPU, step which obtains an important acceleration factor compared to the CPU as it is shown in Section 4.5.4.1. In this experiment, matching process is tested using different sizes of the model library, ranging from 4 to 64 objects. The number of objects was simulated copying the real model library comprised of 4 objects. It is considered that every object is described extracting tensors from 6 partial views obtaining an average of 150 tensors per model.

Regarding GPU memory limitations, since modern GPUs have large global memories, we can find in the market consumer GPUs equipped with a 6 GB RAM memory, storage of the model library for the matching process it is not a problem. Moreover, tensor representation is compressed by squeezing out the zero elements and retaining the non-zero values and their index positions in the tensor. Even not compressing tensors, the overall storage of 120 models using an average of 6 views per model will results in less than 40 MB of storage space.

Finally, total computation for the GPU took around 800ms using the NVIDIA GTX480, managing 3D object recognition problem in real-time and therefore enabling its integration in mobile robotics applications. We believe that this GPU implementation its an important step towards the computation of 3D geometric descriptors in real-time.

Experiments performed in this case of study have demonstrated that GPGPU paradigm allows to considerably accelerate algorithms regard to CPU implementations and to run these in real-time. Moreover, advantageous results are obtained in the use of the GPU to accelerate the computation of a 3D descriptor based on the calculation of 3D semi-local surface patches of partial views, thus allowing descriptor computation at several points of a scene in real-time. Matching process have also been accelerated onto the GPU, taking advantage of the GPU pipeline and achieving a speed-up factor of 3x regard the CPU implementation. Finally, the implemented prototype obtained satisfactory results in terms of accuracy and performance. We showed that implemented prototype took around 800 ms with a GPU implementation to perform 3D object recognition of the entire scene.



**Figure 5.8:** 3D object recognition in cluttered scenes. Different partial views of two scenes are showed. Multiple labels are shown as all computed tensors are evaluated and matched against the library model.

## 5.4 Computer vision: Multi-GPU Based Camera Network System using GNG

In this case of study we present a GPU multi-camera surveillance system based on the use of self-organizing neural networks to represent events in video. The goals include: identifying and tracking persons or objects in the scene or the interpretation of user gestures for interaction with services, devices and systems implemented in the digital home. Additionally, the system process several tasks in parallel using GPUs (Graphic Processor Units). Addressing multiple vision tasks of different levels such as segmentation, representation or characterization, analysis and monitoring of the movement to allow the construction of a robust representation of their environment and interpret the elements of the scene.

It is also necessary to integrate the vision module into a global system that operates in a complex environment by receiving images from multiple acquisition devices at video frequency and offering relevant information to higher level systems. The global system monitors and takes decisions in real time, and must accomplish a set of requirements such as: time constraints, high availability, robustness, high processing speed and re-configurability. Based on our previous work with Growing Neural Gas (GNG) models, we have built a system able to represent and analyse the motion in several image sequences acquired by a multi-camera network and also able to process multi-source data in parallel onto a Multi-GPU architecture. All these features allow to create a multi-objective visual surveillance system. Proposed system is able to keep the privacy of the persons under observation by using the graph representation provided by the GNG. Several experiments are presented demonstrating the validity of the architecture to manage images from different cameras simultaneously.

### 5.4.1 Related works

Development of the process of visual surveillance in dynamic scenes often includes steps for modelling the environment, motion detection, classification of moving objects, tracking and recognition of actions developed.

Most works are focused on applications related to track people or vehicles that have a large number of potential applications such as: controlling access to special areas, people identification, traffic analysis, anomaly detection and management alarms or interactive monitoring using multiple cameras [Hu et al., 2004a, Velastin and Remagnino, 2006].

Recognition of actions has been extensively investigated [Collins et al., 2000, Howarth and Buxton, 2000]. The analysis of the trajectory is also one of the basic problems in understanding the actions [Hu et al., 2004b]. Relevant works on tracking objects can also be found in [Tian et al., 2002, Wu et al., 2000] among others. Moreover, the majority of visual surveillance systems for scene analysis and surveillance depend on the use of knowledge about the scenes where the objects move in a predefined manner [Howarth and Buxton, 1992, Brand and Kettner, 2000].

In recent years, related works to the analysis of behaviours has increased because of the use of effective and robust techniques for detecting and tracking objects and people. That has allowed researchers to focus on higher levels of scene understanding. Moreover, thanks to the proliferation of low-cost vision sensors, embedded processors, and efficiency of wireless networks, a large amount of research has focused on the use of multiple sources of information for the analysis of behaviour.

In particular, multi-camera networks are used for interpreting the dynamics of objects moving in wide areas or for observing objects from different viewpoints to achieve 3D interpretation. Multiple viewpoints help in dealing with ambiguities and occlusions and can lead to more reliable analysis of the scene. Third generation surveillance systems are usually referred to systems conceived to deal with a large number of cameras, a geographical spread of resources and many monitoring points and to mirror the hierarchical and distributed nature of human process of surveillance [Velastin and Remagnino, 2006].

In the deployment of camera networks in end-user environments such as private homes or public places, awareness of privacy, confidentiality, and general security issues is rising. Emphasis should be given to special requirements of camera networks systems, including privacy and continuous real-time operation. To guarantee data authenticity and protect sensitive

and private information, a wide range of mechanisms and protocols should be included in the design of camera networks surveillance systems.

### 5.4.2 Multi-source Information Processing with GPU

GPU-based GNG implementation presented in Chapter 4 has been integrated in a previous single camera surveillance system [García-Rodriguez et al., 2011]. It also has been extended to a muti-camera video surveillance system with strong temporal constrain. In addition, it has been combined multi-core CPU architecture and multiple GPU devices to manage several streams in parallel and also to accelerate each stream using different GPUs.

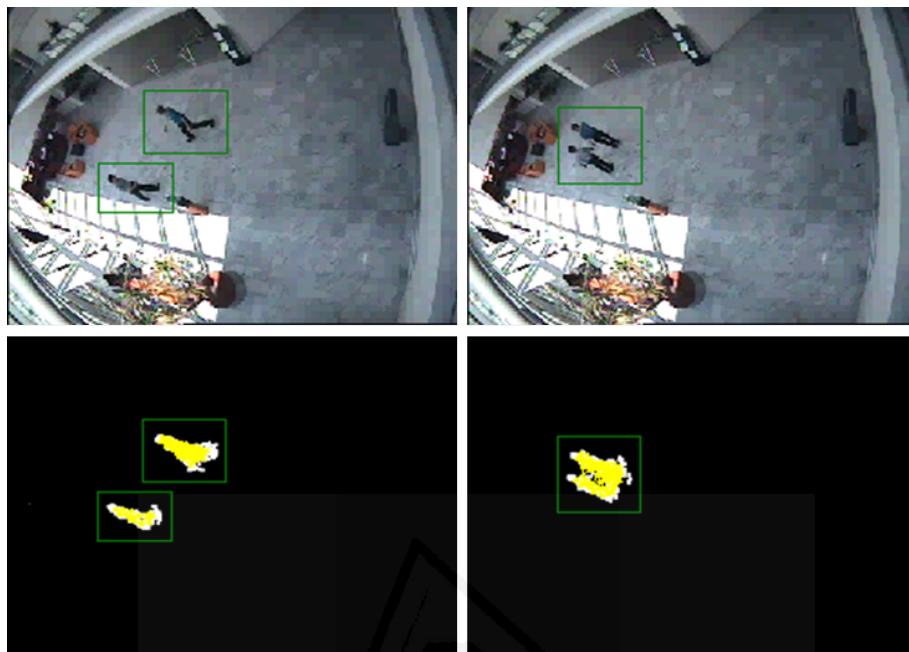
We have considered the following domain-specific constraints regarding a typical video surveillance application in our study. In order to achieve real-time processing, the number of frames processed per second (fps) was determined depending on the number of cameras. The maximum number of cameras that can be supported in the multi-GPU architecture is obtained by calculating the number of video frames that can be processed simultaneously with this constrain.

#### 5.4.2.1 Privacy and security. Real time constraint

Real-time processing is of great importance, particularly in the security domain. A high number of cameras, combined with an increase in security risks in crowded public places, such as in airports, underground stations or terminals, and town squares, necessitate the automation of surveillance process in real time because it is no longer possible to carry out this process manually. Real-time processing is also required in other video applications, such as automated video content analysis and robotics.

Figure 5.9 presents the representation of people appearing in the image with our system. The system is able to keeps the privacy of the persons under observation by using the graph representation provided by the GNG instead of offering to system users original video images.

The importance of GPUs has recently been recognized for different applications such as video and image processing algorithms. In particular, real-time video surveillance applications [Schreiber and Rauter, 2009,



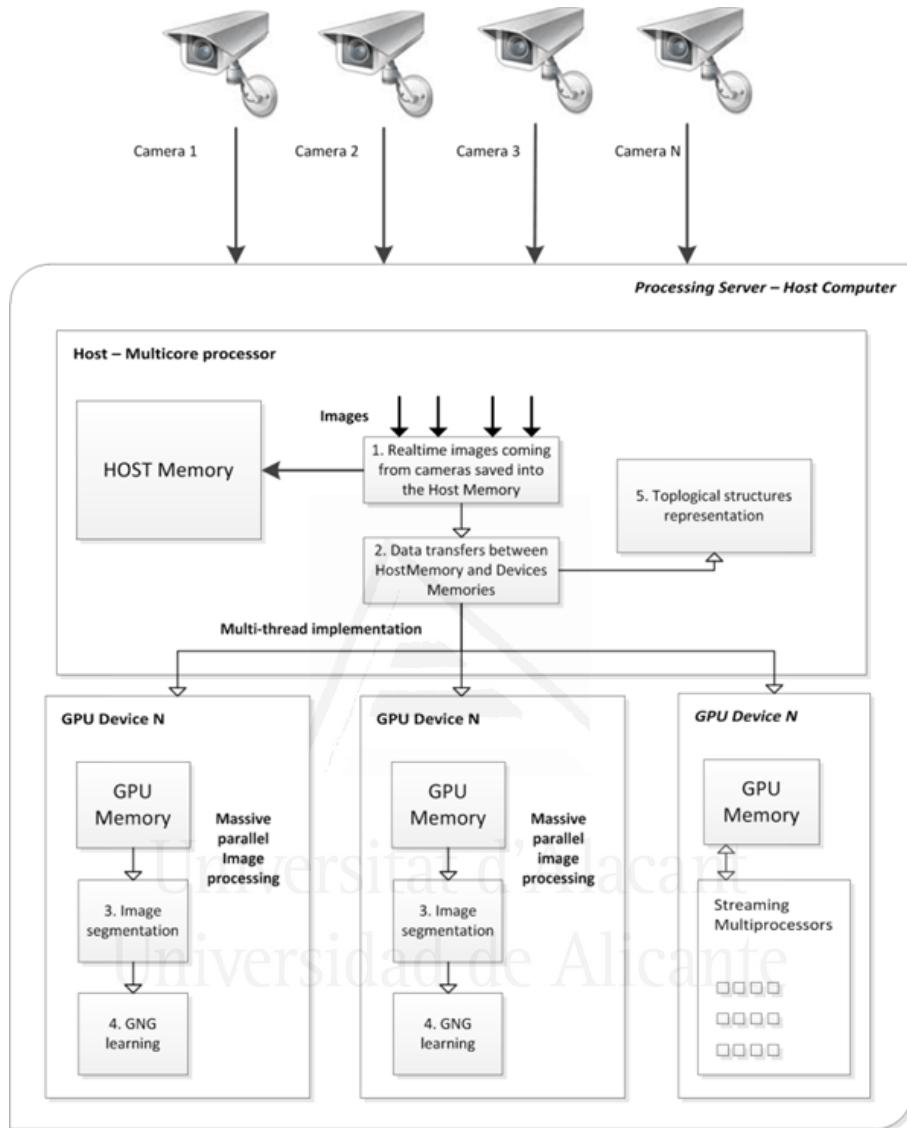
**Figure 5.9:** GNG graph representation

Wang et al., 2006], where it is possible to manage a number of cameras information [Temizel et al., 2011] thanks to the parallel processing capabilities of GPUs.

#### 5.4.2.2 Multicore CPU and multiGPU implementation

Information processing from different cameras in a GPGPU architecture allows fast access to data from different sources allocated in centralized shared memories and allows to solve problems like occlusions or to interpret simultaneous actions in different areas under surveillance. That is possible since the compact representation obtained with the GNG allows storing a large amount of historical video information.

We first developed a single GPU version where the algorithm was parallelized and accelerated via CUDA technology. After accelerating image processing on a single node using a GPU we implemented a higher level of abstraction. This new implementation includes the use of a multi-core processor and various GPUs connected to the same host system (Figure 5.10).

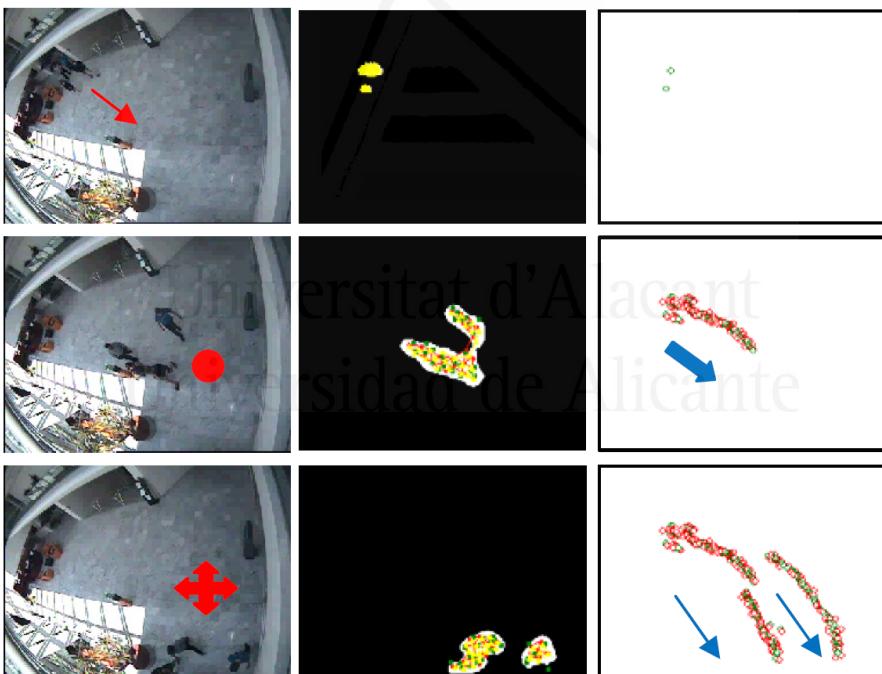
**Figure 5.10:** MultiGPU multi-camera workflow

Through an efficient management of multi-core CPU threads and assigning different streams on different GPUs, we performed parallel processing of streams faster than using a traditional processing system composed by a single CPU. This proposed system is also fully scalable, allowing the user to add more CPU cores and GPU cards to further improve the number of

streams that can manage in parallel. It was used the POSIX pthreads<sup>1</sup> standard to manage several CPU threads in parallel performing different processing task in each GPU device.

### 5.4.3 Experiments

In this section we present some results of experiments with the GNG used to learn features in 2D images with different number of cameras, neurons and input patterns based on our previous GNG visual surveillance system with a single camera [García-Rodríguez and García-Chamizo, 2011]. Figure 5.11 shows the multi-target representation using our previous system and the CAVIAR dataset (Context Aware Vision using Image-based Active Recognition) [Fisher, 2004].



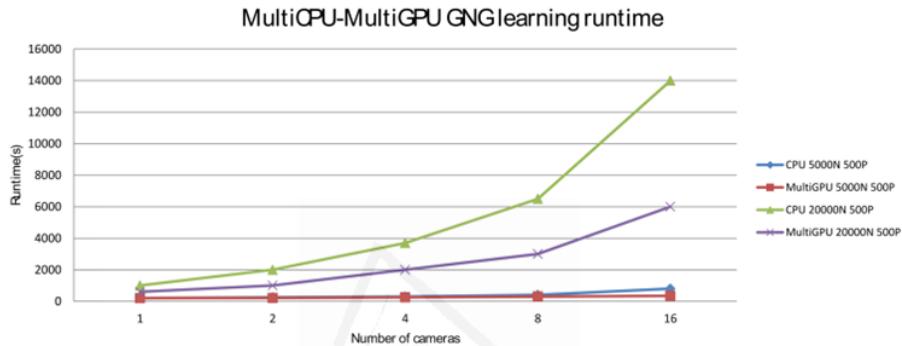
**Figure 5.11:** Example of single image multi-target visual surveillance system

Figure 5.12 shows results of experiments with different number of cam-

---

<sup>1</sup>POSIX Threads, usually referred to as Pthreads, is a POSIX standard for threads. The standard, POSIX.1c, Threads extensions (IEEE Std 1003.1c-1995), defines an API for creating and manipulating threads.

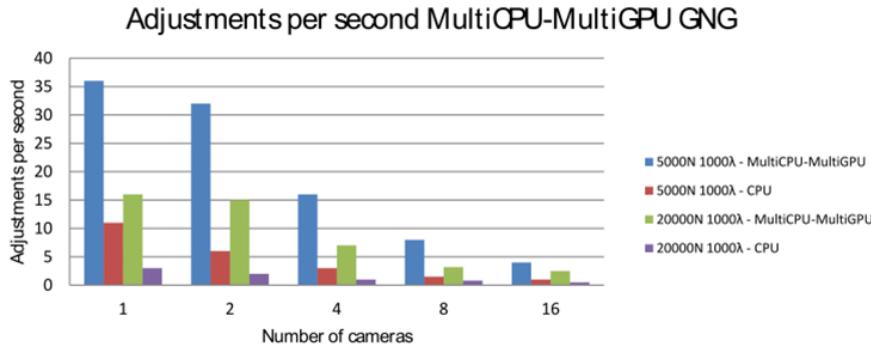
eras to compare time spent to characterize entities with GNG using 5,000 and 20,000 neurons respectively and training the maps with 500 and 1000 input pattern per iteration. Experiments were developed onto computing node with a heterogeneous CPU-GPU architecture with multicore CPUs and two GPU devices.



**Figure 5.12:** Execution times and speed-up for different GNG learning parameters and different number of cameras

In addition, in Figure 5.12 it can be seen how the CPU version time complexity grows much faster than the multi-core CPU and multi-GPU version and is therefore an inadequate solution for a large number of cameras. However, the proposed parallel version increases the number of cameras without degrading performance significantly. We also appreciated that with a high number of cameras the speed-up obtained using the proposed architecture is about 2.5x faster compared with the single threaded CPU version.

Figure 5.13 demonstrates how the proposed implementation is capable to process in parallel up to 16 streams from different cameras and to perform 3 adjustments per second on each stream compared with single threaded CPU version that only can perform adjustments on one stream. Also it is shown how CPU version obtains significantly lower rate of adjustments compared with the proposed version, obtaining rates between 5 and 10 times higher.



**Figure 5.13:** Rate of adjustments per second for different GNG learning parameters and different number of cameras using MultiGPU-MultiCPU and single threaded CPU versions

## 5.5 CAD/CAM: Growing Neural Gas Landmark. Application to Rapid Prototyping in Shoe Last Manufacturing

Customizing shoes manufacturing is one of the great challenges in the footwear industry. It is a production model change where design adopts the main role but also the main bottleneck. It is therefore necessary to accelerate this process by improving the accuracy of current methods. Rapid prototyping techniques are based on the reuse of manufactured footwear lasts so that they can be modified with CAD systems leading rapidly to new shoes models.

With the advent of CAD/CAM and rapid acquisition devices it is possible to digitize old raised shoe lasts for reusing them in the shoe last design software. The first scanners were mechanical copiers (see Figure 5.14), slow and inflexible but very accurate and robust, and also offered a spatial ordered 3D information as shaped sections that were easily treated by the CAD software. The main problem was the slowness that makes them not suitable for rapid prototyping environments, but yet still used in the industry [Jimeno-Morenillo et al., 2011].

The alternative to the mechanical digitizers are the optical ones (see Figure 5.14 on the right). The current optical digitizers are very flexible

and provide lots of information quickly. However, the information provided is not sorted in the 3D space, so it is the CAD which should sort and adapt it to the reconstruction process.

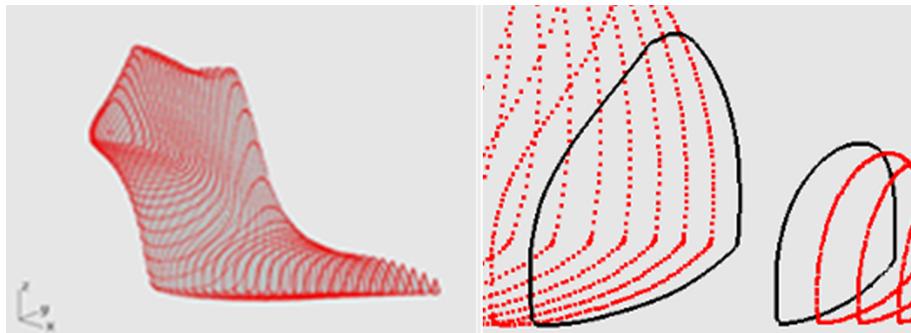


**Figure 5.14:** Mechanical (left) and optical (right) digitizers for shoe lasts.

The current footwear software requires ordered information defining the shoe last surface. The usual method to achieve such surface reconstruction is the interpolation of a grid of points by a tensor surface. This is due to the inheritance of mechanical copiers supplying a fixed number of points of the shoe last per step, but also because shoe last manufacture machines are turning lathes and to maintain constant angular velocity in the rotation axe it is required a trajectory with a fixed number of points per revolution.

The mesh to interpolate (Figure 5.15) is formed by a sequence of sections represented by a fixed set of points (depending on the accuracy, this amount may vary between 90 and 300 3D points). However, in order to meet the requirements of accuracy for this surface, it is necessary a points distribution according to the topology of the object, without losing key points as corners and maintaining a uniform distribution throughout the shoe last.

The main goal of this research is to obtain a grid of points that is adapted to the topology of the footwear shoe last from a sequence of sections with disorganized points acquired by sweeping an optical laser digitizer. The process should be quick and precise to ensure both the accuracy requirements in the design of shoe lasts and its viability in rapid prototyping processes. The presented method is based on the use of the Growing Neural Gas method to fit topological features of the objects and mark a



**Figure 5.15:** Typical sequence of sections on a shoe last and detail.

fixed number of points across each section.

### 5.5.1 Modified GNG for Landmarking and 3D Reconstruction

Automated landmark extraction is accomplished through the use of the self-organising network, the Growing Neural Gas (GNG), which is able to topographically map the low dimensionality of the network to the high dimensionality of the contour manifold without requiring a priori knowledge of the input space structure. Moreover, our GNG landmark method is tolerant to noise and eliminates outliers.

In our research, landmark localization is considered as a cluster-seeking problem in which the goal is finding a finite number of points that describe the contour precisely. It should find the structure of the nodes (locations) automatically by minimizing the error of the self organising map adaptation.

For the automatic extraction and correspondence of landmark points we use the GNG-based algorithm. GNG allows us to extract in an autonomous way the contour of any object as a set of edges that belong to a single polygon and form a topology preserving map.

Our method is able to find a fixed number of landmarks placing them in an accurate way. The method is tolerant to noise and automatically delete outliers by using edge length average and reorder landmarks by using the own neural network neighbourhood structure. The landmarks obtained for each of the acquired section serves automatically for building a tensor that

represents the 3D surface.

### 5.5.2 Experiments

In this section we present the experiments performed to test the validity and efficiency of the developed method. First, a comparison is made with the Voxel Grid method. This method has good features for efficiently and precisely obtaining an arrangement of the sections provided by the digitizer. Second, our method is compared with filtering and reconstruction performed by commercial software for shoe lasts design. Finally, reconstruction and filtering results provided by the CAD software are shown

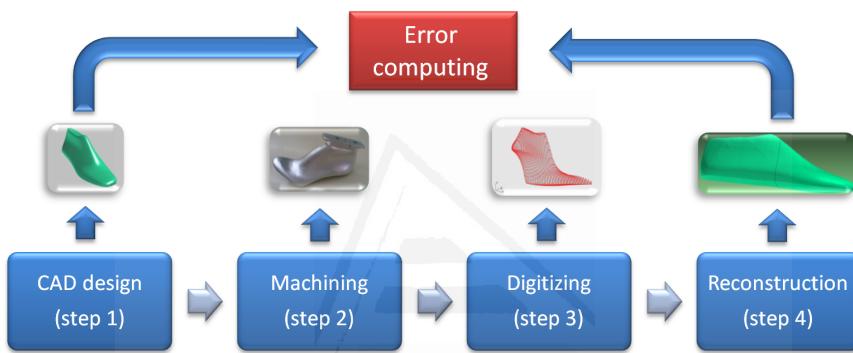
Several experiments have been carried out to demonstrate the validity of our proposal. GNG Landmarking results have been compared with Voxel Grid method. Different parameters for GNG have been tested and compared using quality measures. First experiment demonstrates the accuracy of the representation generated by the GNG network structure compared with other filtering methods such as the Voxel Grid. Three real shoe lasts were used for the experiments with different shapes (see Figure 5.16).



**Figure 5.16:** Shoe lasts used in the experiments. On the left, last 1; center, last 2; on the right, last 3

The way that error is computed is shown in Figure 5.17. First, some shoe lasts have been designed in the CAD synthetically (step 1). These shoe lasts were manufactured using CNC (step 2). The machines used in the manufacture process guarantee an error below  $+/- 0.1\text{mm}$ . Third,

the physical lasts, once manufactured, are scanned using optical sensors resulting in an unorganized point cloud (step 3). The same point of clouds is the input to the algorithms to be compared. Once the results of both algorithms are obtained, the resulting point mesh is compared with the original synthetic surface (step 4) and thus gives a cumulative measure of the error. Therefore, the shoe lasts designed in CAD (step 1) are used as a comparative reference, minimizing the effect of noise introduced by the optical scanner.



**Figure 5.17:** Error calculation process.

Table 5.2 shows the accumulated mean error for all the sections of the shoe last expressed in millimetres. The GNG method provides a lower mean error and therefore better adaptation to the original input space, maintaining a better quality of representation in areas with a high degree of curvature and eliminating the noise generated by the sensor. The Voxel Grid method eliminates noise with the sacrifice of information loss of the input space and therefore a worse adaptation. The experiments were performed with a fixed number of points, and in the case of GNG it has been tested with different number of input signals generated by iteration, obtaining better results with a higher number of adjustments with the sacrifice of higher computation times.

Figure 5.18 (top) visually shows the results discussed in Table 5.2. It can be observed how the adaption of the filtered points obtained with the Voxel Grid method produces less accurate results than the ones obtained

Last	Points	VG	GNG 200 $\lambda$	GNG 1000 $\lambda$	GNG 2000 $\lambda$
1	200	21.68	19.72	19.98	18.45
2	200	20.2	18.5	18.66	18.61
3	200	22.95	21.05	21.08	21.06

**Table 5.2:** Accumulated mean error for the whole shoe last (millimeters): Voxel Grid versus GNG. Different number of input patterns for the GNG method.

using the GNG method. This improper adjustment for all the sections finally generates a less accurate markers respect to the original input space.

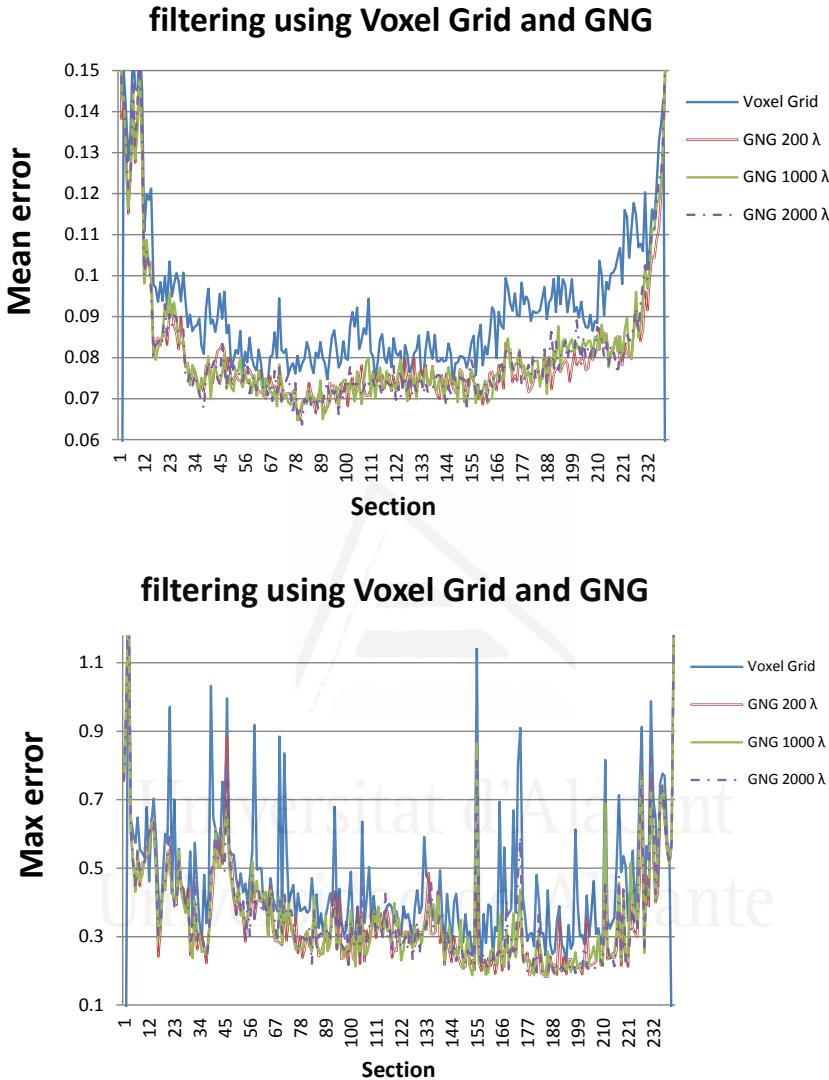
Voxel Grid method produces worse results compared to GNG method. It shows how for a large number of sections the mean error obtained using the Voxel Grid method is higher than the one obtained using GNG. Table 5.3 shows how the GNG method is more robust to noise than the Voxel Grid method. The maximum error for the shoe last indicates the robustness of the method for noisy inputs. The maximum error is calculated as the highest error for all the sections of the shoe last. The lowest maximum error is obtained using the GNG method with different number of input signals. Figure 5.18 (bottom) visually shows the results discussed in Table 5.3 and how highest peaks of error are produced by the Voxel Grid method.

Last	Points	VG	GNG 200 $\lambda$	GNG 1000 $\lambda$	GNG 2000 $\lambda$
1	200	1.376	1.299	1.318	1.336
2	200	1.429	0.926	0.942	0.860
3	200	1.666	2.123	1.468	1.596

**Table 5.3:** Maximum error for the whole shoe last (millimeters): Voxel Grid versus GNG. Different number of input patterns for the GNG method.

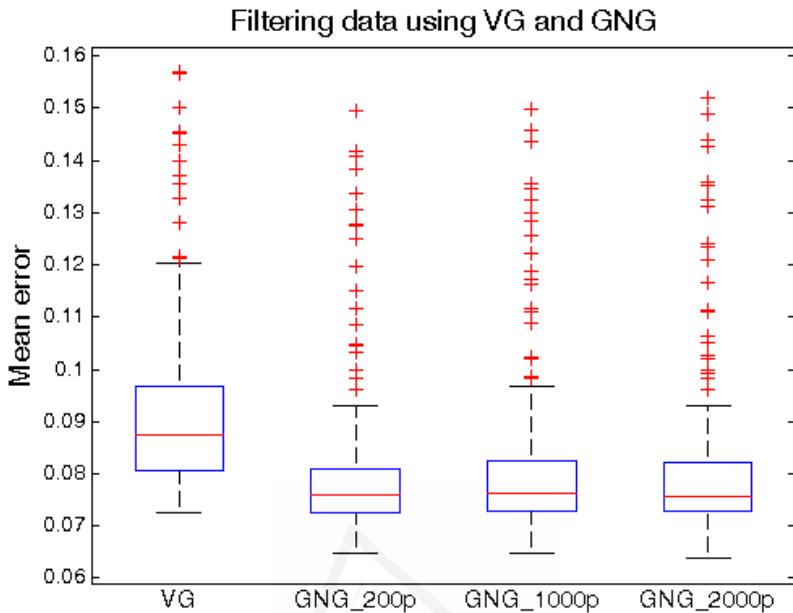
Figure 5.19 shows the boxplot of the mean error for all the sections of the shoe last. The boxplot shows that filtered sections using Voxel Grid method generates less accurate results compared to GNG (larger mean error). Moreover, the boxplot also shows how the Voxel Grid method produces more outliers and then higher errors for all the analysed sections.

In Figure 5.20, the mean distribution error for all the sections of the shoe last is presented. The gauss curve fitted to the error frequency data shows how GNG method produces a larger amount of sections with less error while the gauss curve fitted to the error frequency data produced



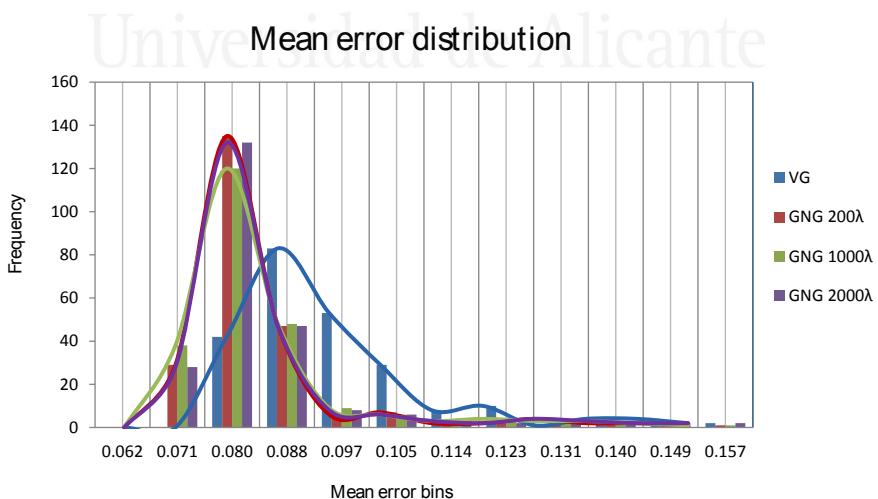
**Figure 5.18:** Voxel Grid versus GNG: Mean error and max error (millimetres). Different number of input signals  $\lambda$  for the GNG method. Left: mean error for all the sections. Right: maximum error for all the sections.

by the VG method produces a wider curve and therefore a shoe last reconstruction with larger errors. GNG method produces similar error for different values of parameter, obtaining the best filtered shoe last with  $\lambda$



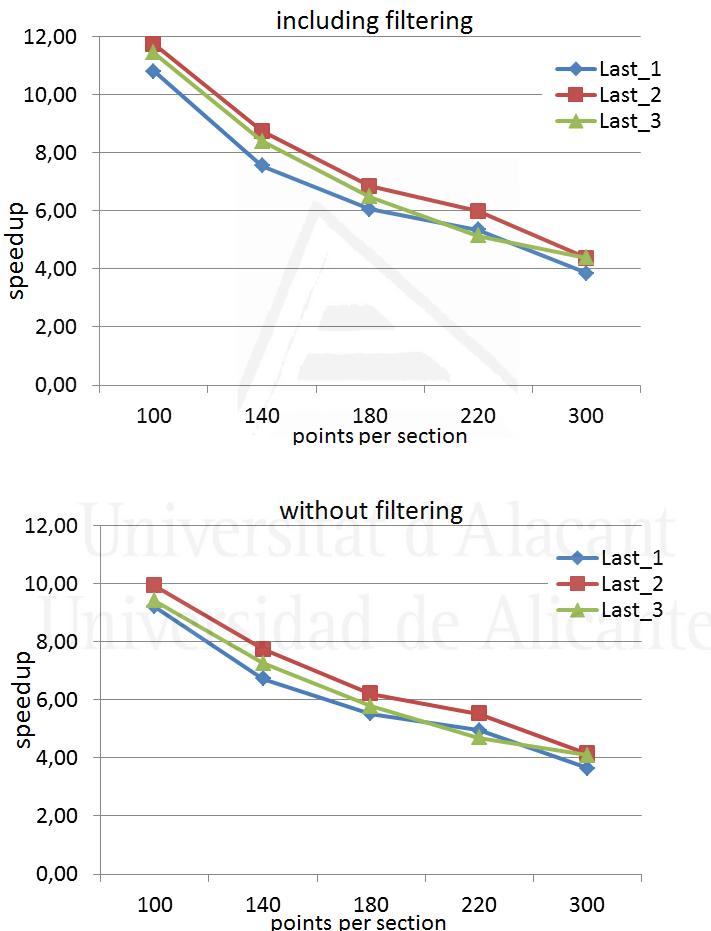
**Figure 5.19:** Voxel Grid versus GNG mean errors.

equals to 200.



**Figure 5.20:** Voxel grid versus GNG distribution of error.

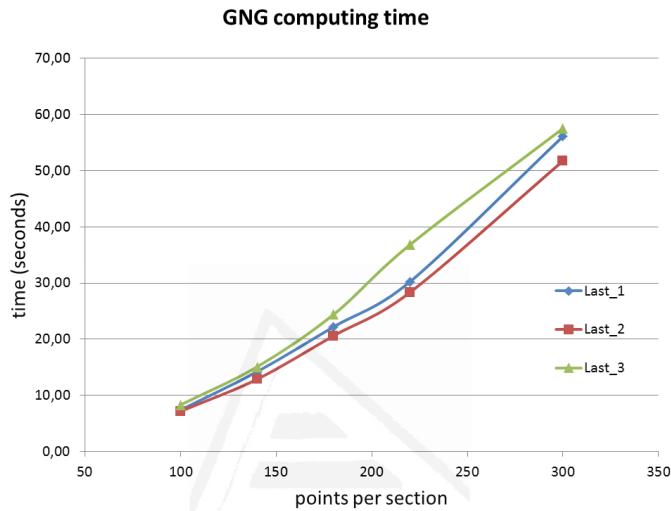
It has been made a comparison of the method presented with the morphological method that uses one of the shoe lasts design software longer used in the industry. Virtual Digitizing does not incorporate filtering process, so Figure 5.21 shows results obtained for three different shoe lasts by using the proposed method regardless filtering time and filtered using the method that the CAD system includes. The shoe lasts used for the experiments are the same that those used in the previous experiment.



**Figure 5.21:** Comparison of reconstruction times without filtering (left) and filtering (right).

As it can be inferred from the graphs, the efficiency of the proposed method obtains an acceleration of  $4x$  in the higher precision cases and

arrives to an acceleration factor of  $10x$  in cases of low accuracy. When the filtering time is considered, the proposed method increases efficiency, ranging from  $6x$  to  $12x$  times faster than the morphological method. Figure 5.22 shows the computation time for the different models and sections.



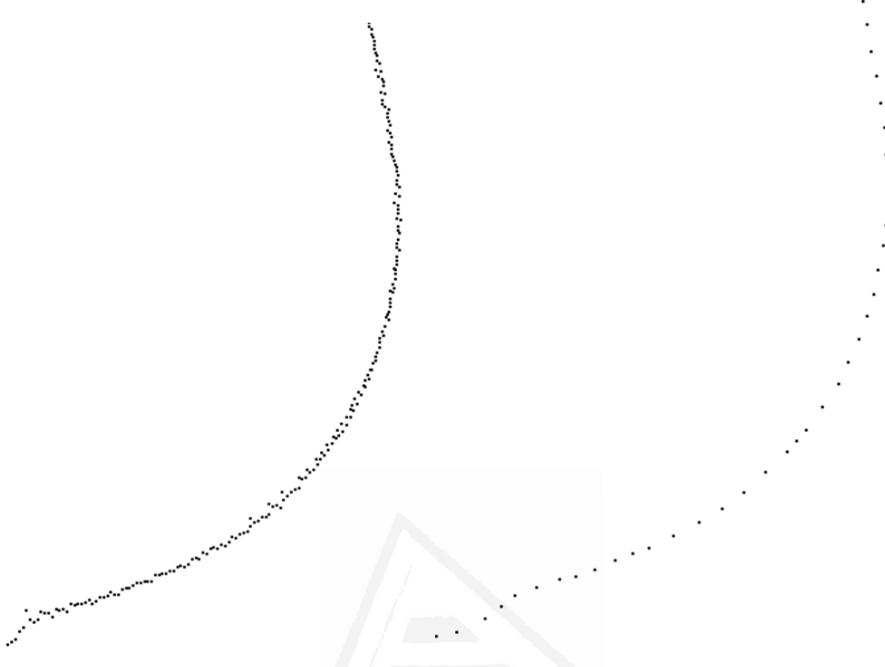
**Figure 5.22:** Computation time for different models and sections.

### 5.5.2.1 2D filtered quality results

The optical scanner employed in the experiments uses a linear laser beam for each section of the shoe last. As the linear beam cannot cover the entire section, it takes 3 shots to complete them. Therefore, it should be noted that each section the digitizer gets is a superposition of three different runs that obviously contain overlapped areas (see Figure 5.23).

Figure 5.24 shows various sections of shoe lasts where the method has been applied using a different number of points. The results obtained show that the sections preserve the input space topology, regarding important points in the design of shoe lasts as corners.

In Figure 5.25, some topological preservation experiments are performed in order to validate our proposal. Reduced sections using GNG method generates smoother results and deals better with the presence of



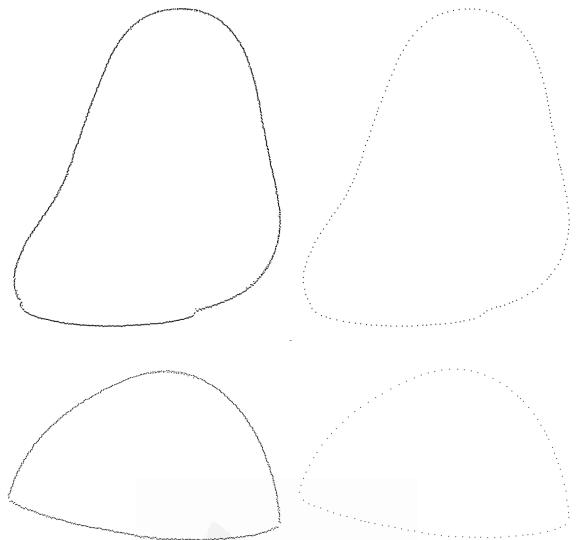
**Figure 5.23:** Detail section: overlay (top) areas with noise. GNG sorting and filtering (bottom).

noise. Special interest areas from left (GNG) and right (VG) part of the Figure 5.25 are remarked to visually show how GNG method produces better topological adaptation results.

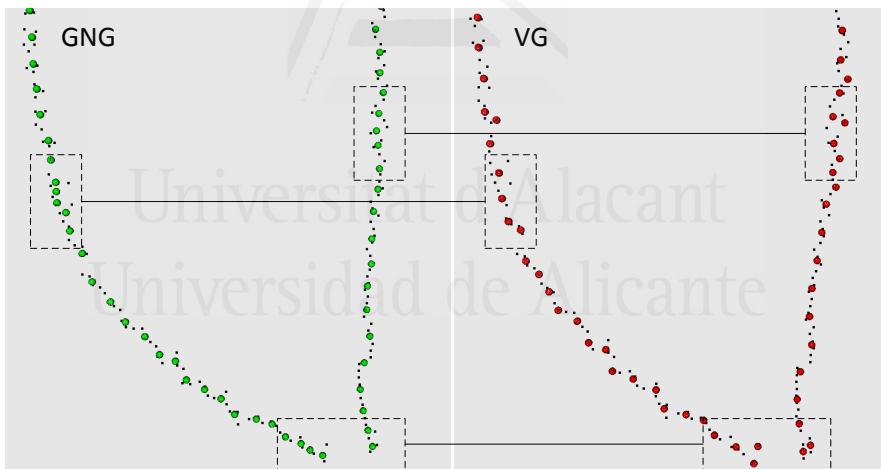
### 5.5.2.2 3D reconstruction quality

This section shows the result of the interpolated mesh obtained by the three tested methods: our GNG, Voxel Grid and Virtual Digitizing. Figure 5.26 and Figure 5.27 shows how the three reconstruction methods provide similar qualities. However, the Voxel Grid method has a significant loss of accuracy in the marked area which corresponds with an area with bottom corners. This difference is not visually relevant but it is crucial to maintain the rigorous requirements of precision in shoe lasts manufacturing.

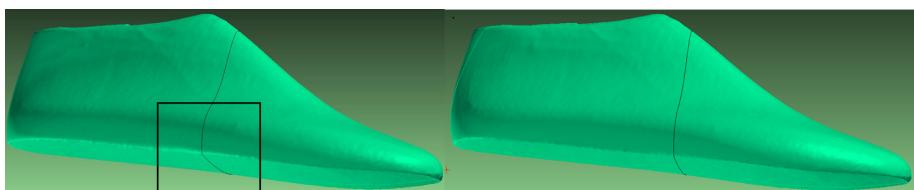
Regarding the comparison with Virtual Digitizing method (see Figure 5.27), significant differences are appreciated at the toe part, located in the area of the bottom corners. This part usually contains scanning errors



**Figure 5.24:** Landmark points using GNG for a section with 180 and 200 points.

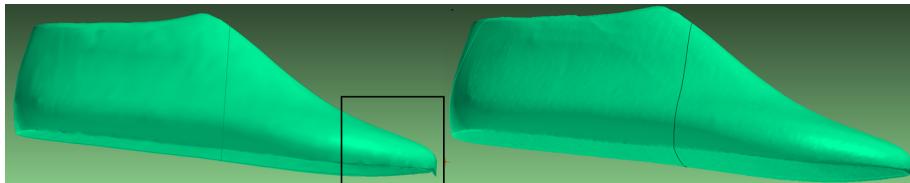


**Figure 5.25:** GNG vs VG topological preservation comparison.



**Figure 5.26:** Voxel Grid (left) vs GNG (right).

because the laser impinges tangentially on the toe, and the bottom corners of the shoe last. For the Virtual Digitizing method, some points were not well filtered and then wrongly located. These errors do not appear in the GNG method.



**Figure 5.27:** Virtual Digitizing (left) vs GNG (right).

We can conclude that the GNG network provides a fixed number of markers in topologically appropriate areas, it is also tolerant to noise, and eliminates outliers. GNG also automatically improves simple placement methods based on distance (impossible to apply since scanner data is noisy).

Moreover, the method has been successfully tested with real shoe lasts and results have been compared with the ones provided by a real CAD/CAM software for shoe lasts. Although the method has been tested on shoe lasts, it is generic and easily extensible to other industrial objects that need to be digitized and reconstructed with efficiency and high precision requirements.

## 5.6 CAD/CAM: 3D Model Reconstruction using Neural Gases

The process of 3D reverse engineering takes a physical device or structure, digitizes it and imports all of its parts and measurements into a computer-aided design (CAD) software or similar building software using a 3D scanner. By using 3D reverse engineering, parts can be improved, the device can be remade with smaller parts, and the user can store the design for later use. These scanners, which often use a laser, scan the device and record the measurements, colors, texture and shape of the piece. From this information, the CAD or similar software can import the shape into

its interface. Recently, the use of scanned 3D models has become popular due to the decreasing cost of 3D sensors and 3D printers. Moreover, realistic 3D geometric models are becoming increasingly important to many computer applications using virtual and augmented reality, particularly in architecture, design, heritage, simulators and entertainment applications. The ability to construct such models automatically from real structures opens up a diversity of additional opportunities.

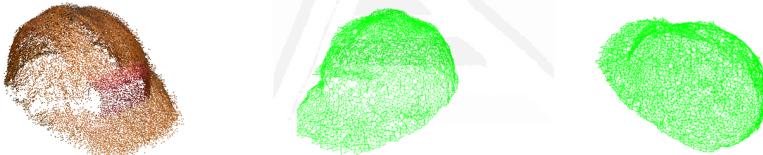
Sometimes only the shell of the device is scanned, but more often, for proper 3D reverse engineering, the device is opened and the parts are removed. The individual parts are then scanned in the same way the entire device was, and the parts are then imported. This allows for a complete and comprehensive 3D model acquisition of the entire device and all of its internal components.

A 3D model of a free-form object [Freeman, 1990] is constructed by acquiring multiple viewpoints so that its surface is completely covered. These views are then registered in a common coordinate basis and the implicit surface of 3D data is calculated obtaining a 3D mesh. Although many algorithms have already been proposed for mesh generation from an unorganized point cloud, existing techniques have various limitations mainly in terms of their applicability to free-form objects, accuracy, efficiency, and the discriminating capability of the used feature representation. An excellent survey of free-form object representation and recognition techniques are found in [Campbell and Flynn, 2001, Flynn and Jain, 1991].

The three-dimensional representation of a product provides a higher degree of realism. This factor has been implemented at different stages of production processes in order to improve productivity and product quality. The combination of the acquired and reconstructed 3D models with virtual and augmented reality environments allows the users to interact with them and also developing a virtual manufacturing system as an application of augmented reality. This combination improves several process stages [Khan and Cheng, 2011], including quality control, human-machine interface and information flows, practising e-commerce and possibility of implementing different production philosophies.

### 5.6.1 Object reconstruction using low cost sensors

In this case of study, we applied the method proposed in Section 2.3.7 to an unorganized point cloud of a builder helmet captured using the Kinect sensor (Figure 5.28). Multiple point clouds have been obtained from different views. These views were aligned using a marker-based approach [Kramer et al., 2012]. Since the Kinect is essentially a RGB-D, the Kinect’s depth measurement expected error is proportional to the squared distance. It ranges from a few millimetres up to about 4 cm at the maximum range of the sensor. Therefore, the proposed method is able to deal with noisy data provided by the Kinect sensor obtaining an accurate 3D surface reconstruction of the captured object without applying other filtering or downsampling additional techniques. A 3D reconstructed model using a low cost sensor is presented in Figure 5.29.



**Figure 5.28:** Builder helmet reconstructed using the proposed method. Left: Point cloud obtained using the Kinect sensor and the marker based-approach. Center and right: Different views of the 3D reconstructed surface using the NG method.

The builder helmet model reconstructed in Figure 5.28 and the textured model present in Figure 5.29 could be used in virtual and augmented reality applications, allowing its rendering in a very realistic way. The same occurs in entertainment applications as video-games. The proposed method allows designers to capture real objects that are present in our environment using low cost sensors like the Kinect and introduce the reconstructed model in their CAD applications. The three-dimensional representation of the models provides a higher degree of realism.

Figure 5.29 shows the final reconstructed model of the object captured using the Kinect sensor. Model texture was transferred from color points of the neurons to mesh triangles interpolating point colors.



**Figure 5.29:** (Left) RGB image of the builder helmet. (Right) Different views of the final reconstructed builder helmet using the proposed method.

## 5.7 Conclusions

In this chapter, the proposed 3D GNG-based method has been validated in several cases of study, demonstrating its capabilities to deal with problems from different areas: mobile robotics, computer vision and CAD/CAM. Small changes were applied to the proposed method in order to solve presented problems. All cases of study where the method was validated presented a common requirement: noise removal. Moreover, the reduced representation created by the GNG method not only improved later post-processing steps but also computing performance. Another interesting feature presented in most cases of study was the capability to preserve the topology of presented objects and scenes. The geometry of sampled objects was preserved with a small error (input space adaptation).



Universitat d'Alacant  
Universidad de Alicante

---

# Chapter 6

# Conclusions

---

This chapter discusses the main conclusions extracted from the work presented in this dissertation. The chapter is organized in four different sections: Section 6.1 presents and discusses the final conclusions of the work presented in this thesis. Section 6.2 enumerates the most relevant contributions made in the topic of research. Next, Section 6.3 lists my publications derived from the presented work. Finally, Section 6.4 presents future works: open problems and research topics that remain for future research.

## 6.1 Conclusions

In this thesis, we first presented a computing method based on Growing Self-Organizing Maps (GSOM) to create 3D models from unorganized raw noisy 3D data. The proposed method does not need information about the kind of sensor used to obtain data so the method works with most 3D sensors. Moreover, the capacity of GSOMs to represent noisy 3D data distributions was demonstrated. Topology preservation capabilities of GSOMs were studied and compared with other methods, showing several advantages for 3D data representation.

The Growing Neural Gas algorithm (GNG) is considered one of the most convenient methods to solve the presented problem as it has some beneficial attributes: flexibility, rapid adaptation, topology preservation, noise removal, dimensionality reduction, etc. We demonstrated the valid-

ity of our proposal testing our method with several models and performed a study of the network parameterization by calculating the quality of representation and also comparing results with other methods like Voxel Grid (VG). We also demonstrated that the proposed method obtained better adaptation to the input space than other filtering methods, obtaining lower error on simulated scenes and CAD models with different levels of noise.

In addition, we modified the original GNG algorithm to represent 3D data sequences, which accelerated the learning algorithm and allowed the method to work faster. This is possible because the algorithm does not restart the map for each new acquisition, but instead readjusts the network structure starting from a previously generated representation and without inserting or deleting neurons. Moreover, we improved original method considering colour information. Neurons' reference vectors were modified including colour components so during the learning stage network structure not only adapt to the input space topology but also learns colour information from the input space. This avoids the necessity to add post-processing steps to include colour information in the final representation. Finally, Competitive Hebbian Learning (CHL) process was extended allowing the GNG algorithm to generate faces during the learning stage and therefore producing a polygonal mesh representation of the input space.

Furthermore, we demonstrated the capabilities of the GNG algorithm to improve keypoint detection process in computer vision applications. Noisy 3D data captured using low cost sensors like the Kinect device was filtered and downsampled using the GNG algorithm. The GNG network provided a reduced and compact 3D structure which has less information than the original data, but keeping the topology. It was demonstrated that state-of-the-art keypoint detection algorithms performed better on filtered point clouds using the proposed method. Moreover, the efficiency of the method was improved since less iterations were required to reject false matchings and to improve its precision. The proposed method was validated in a 3D scene registration process, obtaining lower transformation errors in most detector-descriptor combinations that used the GNG representation as an input data. The most accurate transformations between different point clouds were obtained using the proposed method.

Finally, it was also quantitatively demonstrated that the GNG method overperformed other related filtering and downsampling techniques as VG and Uniform Sampling (US).

Moreover, the GNG algorithm was extended and accelerated in order to obtain a more efficient version, suitable for operations with time constraints. It was implemented using the GPU as the main processor to compute the learning stage. As demonstrated in the experiments, the runtime of sequential GNG algorithm grew with the number of neurons as the network increased. In contrast, in the parallel version implemented on the GPU, as we increased the number of neurons, we obtained a higher acceleration over the sequential version. Experimental results showed that the GPU implementation significantly reduces learning time compared with single-threaded and multi-threaded CPU implementations of the GNG algorithm. In addition, we proposed an hybrid implementation to achieve even a higher acceleration factor of the GNG algorithm. We carried out the computation of the algorithm on the CPU during the first iterations, and then started processing data on the GPU only when there was an acceleration regarding the CPU implementation, thus achieving a higher overall acceleration of the algorithm.

The proposed 3D GNG-based method was validated in several cases of study, demonstrating its capabilities to deal with problems from different areas: mobile robotics, computer vision and CAD/CAM. Small changes were applied to the proposed method in order to solve the presented problems. Specifically, the cases of study presented a common requirement: noise removal, topology preservation and time constraints. Moreover, the reduced representation created by the GNG method not only improved later post-processing steps but also computing performance obtaining considerably accelerated runtimes. The geometry of sampled objects was preserved with a small error (input space adaptation) and without the necessity of establishing topological constraints during the learning process.

Moreover, taking advantage of the GPGPU paradigm, we proposed a real-time implementation of a 3D descriptor. A local shape 3D descriptor based on the extraction of surface patches was implemented on the GPU achieving real-time processing rates. Indeed, we also integrated

on the GPU the preprocessing 3D data algorithms which were necessary for the descriptor computation. Results demonstrated that 3D processing algorithms were considerably accelerated compared to CPU implementations. Within the 3D data algorithms used in the proposed pipeline for the descriptor computation, some progresses have been made towards faster implementations of point cloud acquisition, normal estimation and point cloud triangulation algorithms.

The entire process to extract descriptors on real scenes was accelerated achieving a runtime lower than 1 sec. In fact, the implemented prototype of the proposed descriptor was applied to a object recognition application extracting 200 tensors in less than 0.5 seconds and successfully recognizing objects in the scene.

## 6.2 Contributions of the Thesis

The contributions made in this body of research are as follows:

- A new method to create compact, reduced and efficient 3D representations from noisy data. The method is based on the Growing Neural Gas algorithm, which has been extended and redesigned providing several useful capabilities.
  - The application of the GNG-based algorithm to represent three-dimensional noisy observations obtained with different sensors.
  - The extension of the original GNG algorithm to create full 3D coloured meshes captured using low-cost sensors, including colour and geometric properties.
  - The integration of the proposed method in 3D keypoint detection algorithms. The performance of the detectors was improved using as an input data the reduced and compact representations generated by the GNG method instead of the raw point cloud.
  - The application of the GPU as a general purpose processor to accelerate the learning process of the GNG and NG algorithms. These methods have been redesigned and optimized in order

to achieve the best speed-up. Several analyses have been performed demonstrating which stages have to be parallelized and how to fit them onto a GPU architecture.

- An hybrid implementation of the GNG algorithm that takes advantage of the CPU and the GPU processors in order to achieve the best performance.
- The integration of 3D data processing algorithms in complex computer vision systems. Experiments have demonstrated that the GPGPU paradigm allows to considerably accelerate algorithms compared to CPU implementations and to run these in real-time.
  - Normal estimation has been ported to the GPU considerably decreasing its runtime. GPU computation allowed its integration in real-time applications.
  - Point cloud triangulation, that takes advantage of the point cloud organization, has been also ported to the GPU accelerating its runtime and allowing its execution also in real-time applications.
  - A GPU real-time implementation of a 3D semi-local surface patch extraction algorithm.
- An extensive study of the application of GNG-based methods in various computer vision applications.
  - The integration of the GNG representation to improve 6DoF pose registration in mobile robotics applications.
  - The proposed real-time implementation of 3D data processing algorithm along with the presented real-time semi-local surface patch descriptor have been applied to a 3D object recognition system that works under cluttered conditions.
  - The GNG representation has been used in CAD/CAM applications improving the accuracy of reconstructed models. Moreover, it has also been successfully used in the 3D reconstruction of captured objects using low-cost 3D sensors.

## 6.3 Publications

The following articles were published as a result of the research carried out by the doctoral candidate:

- Published articles in scientific journals:
  - Sergio Orts-Escalano, Vicente Morell, Jose Garcia-Rodriguez, Miguel Cazorla, Robert B Fisher: **Real-time 3D semi-local surface patch extraction using GPGPU**. Journal of Real-Time Image Processing (2014). (Accepted). Impact Factor (JCR 2012): 1.156.
  - Antonio Jimeno-Morenilla, Jose García-Rodriguez, Sergio Orts-Escalano, Miguel Davia-Aracil: **3D-based reconstruction using growing neural gas landmark: application to rapid prototyping in shoe last manufacturing**: The International Journal of Advanced Manufacturing Technology: May 2013. Vol 69. 657-668. Impact Factor (JCR 2012): 1.205.
  - Sergio Orts, José García Rodríguez, Diego Viejo, Miguel Cazorla, Vicente Morell: **GPGPU implementation of growing neural gas: Application to 3D scene reconstruction**. J. Parallel Distrib. Comput. 72(10): 1361-1372 (2012). Impact Factor (JCR 2011): 1.135.
  - José García Rodríguez, Anastassia Angelopoulou, Juan Manuel García Chamizo, Alexandra Psarrou, Sergio Orts-Escalano, Vicente Morell-Giménez: **Autonomous Growing Neural Gas for applications with time constraint: Optimal parameter estimation**. Neural Networks 32: 196-208 (2012). Impact Factor (JCR 2012): 1.927.

- International conferences:
  - Bas Boom, Sergio Orts-Escalano, Xi Ning, Steven McDonagh, Peter Sandilands, Robert Fisher: **Point Light Source Estimation based on Scenes Recorded by a RGB-D camera.** British Machine Vision Conference, BMVC 2013, Bristol, UK.
  - Sergio Orts-Escalano, Vicente Morell, Jose Garcia-Rodriguez and Miguel Cazorla: **Point Cloud Data Filtering and Down-sampling using Growing Neural Gas.** International Joint Conference on Neural Networks, IJCNN 2013, Dallas, Texas.
  - Anastassia Angelopoulou, José García Rodríguez, Alexandra Psarrou, Markos Mentzelopoulos, Bharat Reddy, Sergio Orts-Escalano, Jose Antonio Serra: **Natural User Interfaces in Volume Visualisation Using Microsoft Kinect.** International Conference on Image Analysis and Processing, ICIAP 2013, Naples, Italy: 11-19.
  - Horacio Perez-Sanchez, Gines D. Guerrero, Jose M. Garcia, Jorge Pena, Jose M. Cecilia, Gaspar Cano, Sergio Orts-Escalano and Jose Garcia-Rodriguez: **Improving Drug Discovery using a neural networks based parallel scoring functions.** International Joint Conference on Neural Networks, IJCNN 2013, Dallas, Texas.
  - Jose Antonio Serra-Perez, Jose Garcia-Rodriguez, Sergio Orts-Escalano, Juan Manuel Garcia-Chamizo, Anastassia Angelopoulou, Alexandra Psarrou, Markos Mentzeopoulos, Javier Montoyo Bojo: **3D Gesture Recognition with Growing Neural Gas.** International Joint Conference on Neural Networks, IJCNN 2013, Dallas, Texas.
  - José García Rodríguez, Miguel Cazorla, Sergio Orts-Escalano,

Vicente Morell: **Improving 3D Keypoint Detection from Noisy Data Using Growing Neural Gas.** International Work-Conference on Artificial Neural Networks, IWANN 2013, Puerto de la Cruz, Tenerife, Spain: 480-487.

- Jose Antonio Serra, José García Rodríguez, Sergio Orts-Escalano, Juan Manuel García Chamizo, Anastassia Angelopoulou, Alexandra Psarrou, Markos Mentzelopoulos, Javier Montoyo-Bojo, Enrique Domínguez: **3D Hand Pose Estimation with Neural Networks.** International Work-Conference on Artificial Neural Networks, IWANN 2013, Puerto de la Cruz, Tenerife, Spain: 504-512
- Sergio Orts-Escalano, José García Rodríguez, Vicente Morell, Jorge Azorín López, Juan Manuel García Chamizo: **Multi-GPU based camera network system keeps privacy using Growing Neural Gas.** International Joint Conference on Neural Networks (IJCNN), Brisbane, Australia, June: 1-8.
- Vicente Morell, Miguel Cazorla, Diego Viejo, Sergio Orts, José García Rodríguez: **A study of registration techniques for 6DoF SLAM.** International Conference of the Catalan Association for Artificial Intelligence, CCIA 2012, University of Alacant, Spain: 111-120.
- José García Rodríguez, Anastassia Angelopoulou, Juan Manuel García Chamizo, Alexandra Psarrou, Sergio Orts, Vicente Morell: **Fast Autonomous Growing Neural Gas.** International Joint Conference on Neural Networks, IJCNN 2011, San Jose, California: 725-732.
- José García Rodríguez, Anastassia Angelopoulou, Vicente Morell, Sergio Orts, Alexandra Psarrou, Juan Manuel García Cha-

mizo: **Fast Image Representation with GPU-Based Growing Neural Gas.** International Work-Conference on Artificial Neural Networks, IWANN 2011, Torremolinos-Málaga, Spain: 58-65.

- José García Rodríguez, Enrique Domínguez, Anastassia Angelopoulou, Alexandra Psarrou, Francisco José Mora-Gimeno, Sergio Orts, Juan Manuel García Chamizo: **Video and Image Processing with Self-Organizing Neural Networks.** International Work-Conference on Artificial Neural Networks, IWANN 2011, Torremolinos-Málaga, Spain: 98-104

- National conferences:

- Sergio Orts-Escalano, José García-Rodríguez, Vicente Morell-Giménez. **Procesamiento de múltiples flujos de datos con Growing Neural Gas sobre Multi-GPU.** Jornadas de Parallelismo JP, Elche, España, 2012.

- Book chapters:

- Vicente Morell-Gimenez, Sergio Orts-Escalano, José García Rodríguez, Miguel Cazorla, Diego Viejo. **A Review of Registration Methods on Mobile Robots.** Robotic Vision: Technologies for Machine Learning and Vision Applications. IGI GLOBAL.
- José García-Rodríguez, Juan Manuel García-Chamizo, Sergio Orts-Escalano, Vicente Morell-Gimenez, José Serra-Perez, Anastassia Angelopoulou, Miguel Cazorla, Diego Viejo. **Computer Vision Applications of Self-Organizing Neural Networks.** Robotic Vision: Technologies for Machine Learning and Vision Applications. IGI GLOBAL.

- Poster presentation:

- Sergio Orts, Jose Garcia-Rodriguez, Diego Viejo, Miguel Cazorla, Vicente Morell, Jose Serra: **6DoF pose estimation**

**using Growing Neural Gas Network.** 5th International Conference on Cognitive Systems, Cogsys 2012, TU Vienna, Austria.

- Sergio Orts, Jose Garcia-Rodriguez, Vicente Morell. **GPU Accelerated Growing Neural Gas Network.** Programming and Tuning Massively Parallel Systems, PUMPS 2011, Barcelona, Spain. (**Honorable Mention by NVIDIA**).

## 6.4 Future Work

Regarding GPU-based implementations of the GNG and NG algorithms further works will include other improvements on their acceleration: generating random patterns using the GPU and using multi-GPU computation to improve performance and to manage several neural networks learning different features of the same input space simultaneously.

Another interesting extension of the current GPU-based implementations will be its acceleration using distributed computing. Thanks to the recent integration of the GPU into current clusters of computers, we think that a distributed implementation of our GPU-based GNG algorithm may considerably decrease the overall runtime for clustering high dimensional data. Visual exploration and analysis of multi-dimensional data have become really popular these days due to the amount of information generated on the Internet by most kind of web applications. This new area of research is also known as "big data".

In addition, as a recent new massive parallel architecture called Xeon Phi [Cramer et al., 2012, Fang et al., 2013b, Fang et al., 2013a] has recently been launched by Intel, we plan to port the GPU-based implementations to this new architecture. Further study and comparison on both architectures will be performed showing advantages and disadvantages of both hardware accelerators.

Furthermore, more applications of the accelerated GNG algorithm will be studied in the future.

We also plan to extend the real-time implementation of the 3D local

shape descriptor. Since the current descriptor only considered geometric information and that caused in some experiments wrong results in object recognition, we plan to improve the current descriptor adding visual features extracted from the RGB information. This colour information will help solving ambiguity in cases where the geometry of the recognized objects was very similar. Moreover, the keypoint detector algorithm used for computing the proposed 3D descriptor could be improved using visual detectors on the colour information.



Universitat d'Alacant  
Universidad de Alicante



Universitat d'Alacant  
Universidad de Alicante

# Appendices

Universitat d'Alacant  
Universidad de Alicante



---

## Appendix A

# Computed RMS errors

---

In this appendix all computed RMS errors for different combination of keypoint detectors and feature descriptors are shown. Different search radius for keypoint detectors were tested. Search radius for keypoint detectors range from 0.02 to 0.15 metres. Search radius for feature extractors was set to 0.2 since we are focus on the performance of keypoint detectors. Tables A.1, A.2, A.3 and A.4 show RMS errors for different configurations of search radius in the keypoint detector algorithm.

Furthermore, figure A.1 shows minimum, maximum, median and mean values of all previously computed RMS errors. We wanted to know which representation obtained the minimum error computing the transformation. That figure shows how the GNG representation obtained the smallest errors compared to other filtering methods as VG or US, obtaining an error lower than 1.6 centimetres. In addition, the maximum errors were also smaller than the maximum errors computed with other methods. Mean and median errors were similar for tested methods although GNG representation obtained slightly lower errors than those calculated with VG, US or the raw data.

	SIFT3D				Harris3D			
	FPFH	CSHOT	PFH	PFHRGB	FPFH	CSHOT	PFH	PFHRGB
GNG 5000p	0.168	0.052	0.185	0.106	0.209	0.044	0.031	0.057
GNG 10000p	0.092	0.040	0.367	0.029	0.166	0.026	0.088	0.021
GNG 20000p	0.231	0.019	0.255	0.057	0.130	0.024	0.086	0.067
VG 5000p	0.239	0.086	0.540	0.080	0.111	0.073	0.197	0.025
VG 10000p	0.073	0.061	0.171	0.050	0.072	0.058	0.100	0.028
VG 20000p	0.139	0.022	0.375	0.027	0.139	0.023	0.106	0.028
US 5000p	0.558	0.102	0.202	0.093	0.145	0.063	0.118	0.021
US 10000p	0.065	0.079	0.113	0.062	0.168	0.044	0.349	0.041
US 20000p	0.146	0.026	0.239	0.019	0.210	0.032	0.158	0.022
Raw PCL	0.103	0.028	0.082	0.041	0.162	0.045	0.118	0.031

	Tomasi3D				Noble3D			
	FPFH	CSHOT	PFH	PFHRGB	FPFH	CSHOT	PFH	PFHRGB
GNG 5000p	0.209	0.049	0.126	0.069	0.209	0.044	0.031	0.057
GNG 10000p	0.244	0.026	0.081	0.077	0.231	0.017	0.154	0.069
GNG 20000p	0.212	0.031	0.144	0.044	0.146	0.035	0.071	0.039
VG 5000p	0.111	0.052	0.197	0.025	0.111	0.028	0.277	0.041
VG 10000p	0.115	0.043	0.115	0.073	0.109	0.032	0.080	0.035
VG 20000p	0.121	0.018	0.205	0.039	0.325	0.024	0.078	0.061
US 5000p	0.084	0.075	0.871	0.047	0.075	0.076	0.893	0.083
US 10000p	0.201	0.046	0.121	0.068	0.135	0.058	0.120	0.046
US 20000p	0.285	0.021	0.535	0.054	0.107	0.026	0.044	0.043
Raw PCL	0.046	0.056	0.254	0.024	0.199	0.066	0.159	0.055

	Lowe3D				Curvature3D			
	FPFH	CSHOT	PFH	PFHRGB	FPFH	CSHOT	PFH	PFHRGB
GNG 5000p	0.209	0.097	0.031	0.057	-	-	-	-
GNG 10000p	0.231	0.025	0.154	0.069	0.802	0.757	0.435	0.396
GNG 20000p	0.146	0.021	0.071	0.039	0.110	0.081	0.153	0.037
VG 5000p	0.111	0.077	0.277	0.041	0.227	0.227	0.227	0.227
VG 10000p	0.109	0.041	0.080	0.025	0.227	0.227	2.672	0.227
VG 20000p	0.325	0.024	0.078	0.061	0.243	0.101	0.180	0.038
US 5000p	0.075	0.062	0.893	0.083	0.227	0.125	0.227	0.403
US 10000p	0.135	0.058	0.120	0.046	0.227	0.107	0.072	0.113
US 20000p	0.107	0.028	0.154	0.033	0.584	0.033	0.088	0.053
Raw PCL	0.429	0.034	0.198	0.060	-	-	-	-

**Table A.1:** RMS deviation error (meters) is obtained using different detector-descriptor combinations. Combinations are computed on the original point cloud (raw), and different filtered point clouds using Voxel Grid, Uniform Sampling and the proposed method. Keypoint detector search radius equals to 0.02. Feature extractor search radius equals to 0.02. (metres)

	SIFT3D				Harris3D			
	FPPFH	CSHOT	PFH	PFHRGB	FPPFH	CSHOT	PFH	PFHRGB
<b>GNG 5000p</b>	0.168	0.052	0.185	0.106	0.151	0.085	0.109	0.054
<b>GNG 10000p</b>	0.092	0.037	0.367	0.029	0.114	0.046	0.177	0.047
<b>GNG 20000p</b>	0.231	0.019	0.255	0.057	0.079	0.052	0.153	0.042
<b>VG 5000p</b>	0.239	0.063	0.540	0.080	0.404	0.038	0.305	0.033
<b>VG 10000p</b>	0.073	0.070	0.171	0.050	0.088	0.033	0.097	0.058
<b>VG 20000p</b>	0.139	0.037	0.375	0.027	0.180	0.069	0.469	0.043
<b>US 5000p</b>	0.558	0.089	0.202	0.093	0.156	0.090	0.127	0.072
<b>US 10000p</b>	0.065	0.056	0.113	0.062	0.074	0.036	0.215	0.053
<b>US 20000p</b>	0.146	0.036	0.239	0.019	0.151	0.038	0.152	0.082
<b>Raw PCL</b>	0.103	0.039	0.082	0.041	0.128	0.066	0.144	0.093

	Tomasi3D				Noble3D			
	FPPFH	CSHOT	PFH	PFHRGB	FPPFH	CSHOT	PFH	PFHRGB
<b>GNG 5000p</b>	0.049	0.043	0.189	0.112	0.143	0.098	0.273	0.188
<b>GNG 10000p</b>	0.054	0.023	0.308	0.121	0.186	0.052	0.127	0.117
<b>GNG 20000p</b>	0.270	0.063	0.319	0.066	0.199	0.063	0.239	0.064
<b>VG 5000p</b>	0.383	0.022	0.067	0.066	0.387	0.085	0.073	0.096
<b>VG 10000p</b>	0.127	0.047	0.258	0.049	0.188	0.059	0.244	0.082
<b>VG 20000p</b>	0.148	0.046	0.123	0.078	0.289	0.048	0.188	0.079
<b>US 5000p</b>	0.076	0.078	0.262	0.055	0.137	0.039	0.278	0.062
<b>US 10000p</b>	0.393	0.063	0.106	0.039	0.202	0.068	0.122	0.051
<b>US 20000p</b>	0.326	0.054	0.142	0.079	0.045	0.049	0.210	0.065
<b>Raw PCL</b>	0.659	0.049	0.765	0.062	0.114	0.060	0.099	0.077

	Lowe3D				Curvature3D			
	FPPFH	CSHOT	PFH	PFHRGB	FPPFH	CSHOT	PFH	PFHRGB
<b>GNG 5000p</b>	0.143	0.065	0.273	0.188	0.099	0.048	0.262	0.139
<b>GNG 10000p</b>	0.117	0.052	0.120	0.135	0.228	0.032	0.151	0.071
<b>GNG 20000p</b>	0.076	0.062	0.239	0.016	0.113	0.022	0.123	0.053
<b>VG 5000p</b>	0.387	0.040	0.073	0.077	0.228	0.033	0.244	0.068
<b>VG 10000p</b>	0.188	0.059	0.244	0.082	0.103	0.042	0.101	0.083
<b>VG 20000p</b>	0.093	0.107	0.167	0.076	0.093	0.050	0.057	0.023
<b>US 5000p</b>	0.137	0.096	0.278	0.062	0.151	0.055	0.369	0.053
<b>US 10000p</b>	0.202	0.093	0.169	0.066	0.200	0.076	0.307	0.050
<b>US 20000p</b>	0.365	0.036	0.097	0.031	0.151	0.052	0.123	0.037
<b>Raw PCL</b>	0.203	0.067	0.240	0.027	-	-	-	-

**Table A.2:** Same experiment that presented in Table A.1 but changing the keypoint detector search radius to 0.05 and the feature extractor search radius equalss to 0.02. (metres)

	SIFT3D				Harris3D			
	FPFH	CSHOT	PFH	PFHRGB	FPFH	CSHOT	PFH	PFHRGB
GNG 5000p	0.168	0.052	0.185	0.106	0.256	0.193	0.149	0.060
GNG 10000p	0.092	0.037	0.367	0.019	0.316	0.095	0.093	0.067
GNG 20000p	0.231	0.044	0.255	0.057	0.147	0.057	0.082	0.066
VG 5000p	0.239	0.037	0.540	0.080	0.457	0.152	0.487	0.052
VG 10000p	0.073	0.070	0.171	0.050	0.438	0.079	0.227	0.077
VG 20000p	0.139	0.022	0.375	0.027	0.084	0.071	0.025	
US 5000p	0.558	0.105	0.202	0.093	0.415	0.062	0.311	0.047
US 10000p	0.065	0.056	0.113	0.062	0.227	0.057	0.304	0.076
US 20000p	0.146	0.026	0.239	0.039	0.204	0.093	0.101	0.054
Raw PCL	0.103	0.028	0.082	0.041	0.128	0.123	0.227	0.087

	Tomasi3D				Noble3D			
	FPFH	CSHOT	PFH	PFHRGB	FPFH	CSHOT	PFH	PFHRGB
GNG 5000p	0.084	0.177	2.288	0.098	0.410	0.059	0.458	0.086
GNG 10000p	2.025	0.074	0.157	0.158	0.227	0.101	0.159	0.211
GNG 20000p	0.352	0.030	1.008	0.066	0.146	0.087	0.166	0.072
VG 5000p	0.133	0.090	0.193	0.038	0.457	0.068	1.004	0.069
VG 10000p	0.505	0.106	0.422	0.088	0.328	0.111	0.295	0.077
VG 20000p	0.174	0.220	0.074	0.096	0.084	0.052	0.539	0.025
US 5000p	0.275	0.105	0.159	0.145	0.415	0.102	0.227	0.077
US 10000p	0.577	0.159	0.227	0.075	1.124	0.066	0.304	0.085
US 20000p	0.177	0.074	0.251	0.051	0.227	0.087	1.220	0.054
Raw PCL	0.227	0.099	0.095	0.095	0.267	0.109	0.227	0.087

	Lowe3D				Curvature3D			
	FPFH	CSHOT	PFH	PFHRGB	FPFH	CSHOT	PFH	PFHRGB
GNG 5000p	0.410	0.141	0.458	0.086	0.106	0.124	0.163	0.106
GNG 10000p	0.119	0.105	0.159	0.083	1.451	0.079	0.095	0.042
GNG 20000p	0.146	0.122	0.166	0.072	0.176	0.058	0.248	0.062
VG 5000p	0.457	0.407	0.282	0.069	0.493	0.032	0.052	0.041
VG 10000p	0.328	0.113	0.295	0.077	0.289	0.048	0.066	0.053
VG 20000p	0.084	0.539	0.025		0.104	0.047	0.121	0.057
US 5000p	0.415	0.134	0.227	0.077	0.227	0.073	0.227	0.111
US 10000p	0.546	0.076	0.304	0.085	0.121	0.060	0.160	0.048
US 20000p	0.227	0.040	0.101	0.054	0.096	0.054	0.205	0.040
Raw PCL	0.267	0.109	0.227	0.087	-	-	-	-

**Table A.3:** Same experiment that presented in Table A.1 but changing the keypoint detector search radius to 0.1 and the feature extractor search radius equalss to 0.02. (metres)

	SIFT3D				Harris3D			
	FPPFH	CSHOT	PFH	PFHRGB	FPPFH	CSHOT	PFH	PFHRGB
GNG 5000p	0.168	0.052	0.185	0.106	0.244	0.078	0.249	0.078
GNG 10000p	0.092	0.037	0.367	0.029	0.619	0.081	0.101	0.091
GNG 20000p	0.231	0.031	0.255	0.057	0.114	0.222	0.150	0.077
VG 5000p	0.239	0.063	0.540	0.080	0.605	0.096	0.081	0.052
VG 10000p	0.073	0.070	0.171	0.050	0.152	0.921	0.173	0.086
VG 20000p	0.139	0.060	0.375	0.027	0.062	0.030	0.087	0.069
US 5000p	0.558	0.112	0.202	0.093	0.227	0.252	0.031	0.088
US 10000p	0.065	0.079	0.113	0.062	0.186	0.047	0.052	0.060
US 20000p	0.146	0.025	0.239	0.029	0.078	0.039	0.227	0.207
Raw PCL	0.103	0.028	0.082	0.041	1.608	0.064	0.154	0.058

	Tomasi3D				Noble3D			
	FPPFH	CSHOT	PFH	PFHRGB	FPPFH	CSHOT	PFH	PFHRGB
GNG 5000p	0.412	0.102	1.708	0.153	0.187	0.408	0.249	0.068
GNG 10000p	0.047	0.137	3.389	0.050	0.384	0.062	0.101	0.088
GNG 20000p	0.165	0.066	0.158	0.082	0.246	0.632	0.066	0.105
VG 5000p	0.172	0.245	0.280	0.148	0.833	0.088	0.081	0.052
VG 10000p	1.629	0.057	2.285	0.075	0.110	0.333	0.274	0.051
VG 20000p	1.974	0.155	0.110	0.239	0.062	0.097	0.033	0.239
US 5000p	0.196	0.100	0.227	0.097	2.869	0.227	0.031	0.085
US 10000p	0.744	0.125	0.052	0.108	0.047	0.045	0.060	0.108
US 20000p	0.118	0.206	0.260	0.261	0.078	0.051	0.151	0.043
Raw PCL	0.227	0.139	0.049	0.137	1.608	0.099	0.154	0.058

	Lowe3D				Curvature3D			
	FPPFH	CSHOT	PFH	PFHRGB	FPPFH	CSHOT	PFH	PFHRGB
GNG 5000p	0.187	0.408	0.249	0.068	0.227	0.258	0.227	0.235
GNG 10000p	0.384	0.106	0.101	0.088	0.099	0.023	0.071	0.125
GNG 20000p	0.246	0.536	0.066	0.105	2.091	0.198	0.058	0.090
VG 5000p	0.833	0.057	0.081	0.052	0.387	0.104	3.374	0.077
VG 10000p	0.110	0.422	0.274	0.051	0.227	0.085	0.034	0.048
VG 20000p	0.084	0.097	0.033	0.752	0.061	0.122	0.052	0.227
US 5000p	2.869	3.252	0.031	0.085	0.227	0.085	0.238	0.154
US 10000p	0.047	0.045	0.060	0.127	0.052	2.146	0.166	0.227
US 20000p	0.078	0.081	0.151	0.043	1.795	0.076	0.106	0.060
Raw PCL	2.201	0.099	0.154	0.058	-	-	-	-

**Table A.4:** Same experiment that presented in Table A.1 but changing the keypoint detector search radius to 0.15 and the feature extractor search radius equalss to 0.02. (metres)

---

	Minimum					Maximum			
	0.02	0.05	0.1	0.15		0.02	0.05	0.1	0.15
GNG 5000p	0.031	0.126	0.052	0.052	GNG 5000p	<b>0.209</b>	0.273	2.288	1.708
GNG 10000p	<b>0.017</b>	0.023	<b>0.019</b>	<b>0.023</b>	GNG 10000p	0.802	0.367	2.025	3.389
GNG 20000p	0.019	<b>0.016</b>	0.030	0.031	GNG 20000p	0.255	0.319	1.008	2.091
VG 5000p	0.025	0.022	0.032	0.052	VG 5000p	0.540	0.540	1.004	3.374
VG 10000p	0.025	0.033	0.048	0.034	VG 10000p	<b>2.672</b>	0.258	0.505	2.285
VG 20000p	0.018	0.023	0.022	0.027	VG 20000p	0.375	0.469	0.539	1.974
US 5000p	0.021	0.039	0.047	0.031	US 5000p	0.893	0.558	0.558	3.252
US 10000p	0.041	0.036	0.048	0.045	US 10000p	0.349	0.393	1.124	2.146
US 20000p	0.019	0.019	0.026	0.025	US 20000p	0.584	0.365	1.220	3.593
Raw PCL	0.024	0.027	0.028	0.028	Raw PCL	0.429	0.765	0.267	2.201

	Median					Mean			
	0.02	0.05	0.1	0.15		0.02	0.05	0.1	0.15
GNG 5000p	<b>0.063</b>	0.126	0.145	0.207	GNG 5000p	0.102	0.134	0.268	2.288
GNG 10000p	0.090	0.115	0.103	0.096	GNG 10000p	0.192	0.117	0.264	2.025
GNG 20000p	0.071	<b>0.071</b>	0.105	0.132	GNG 20000p	<b>0.094</b>	0.120	0.163	1.008
VG 5000p	0.111	0.079	0.143	0.100	VG 5000p	0.150	0.168	0.247	1.004
VG 10000p	0.076	0.083	0.109	0.110	VG 10000p	0.201	<b>0.108</b>	0.183	0.505
VG 20000p	0.078	0.086	0.084	0.092	VG 20000p	0.116	0.125	0.136	0.539
US 5000p	0.098	0.094	0.152	0.175	US 5000p	0.233	0.149	0.200	0.558
US 10000p	0.093	0.075	0.099	0.064	US 10000p	0.108	0.120	0.207	1.124
US 20000p	<b>0.053</b>	0.081	0.094	0.094	US 20000p	0.127	0.114	0.161	1.220
Raw PCL	<b>0.063</b>	0.088	0.106	0.101	Raw PCL	0.109	0.156	<b>0.136</b>	0.267

**Figure A.1:** Mean, median, min and max RMS errors of the estimated transformations. These values have been extracted from results presented in Tables A.1, A.2, A.3 and A.4.

---

## Appendix B

# Resumen

---

### B.1 Antecedentes y estado actual

La visión artificial o visión por computador es un campo en constante evolución. Desde los años 80 esta área ha tratado la capacidad de “entender” una escena de forma automática mediante la utilización de un computador. Esta capacidad de entender se desglosa en varias etapas, por ejemplo, la localización y el reconocimiento de objetos que aparecen en una escena, dónde fue capturada, la reconstrucción de un modelo 3D a partir de varias capturas, etcétera. Estas escenas han sido tradicionalmente capturadas utilizando sensores de imagen, permitiendo analizar características sobre el color o la disposición 2D de los elementos en la escena [Skrypnyk and Lowe, 2004].

Los sensores 3D también han sido utilizados pero en menor medida debido a que tradicionalmente su coste ha sido muy elevado, siendo inaccesible para muchos grupos de investigación. Por lo tanto, la mayoría de trabajos en visión artificial se han centrado en la utilización de imágenes RGB como principal fuente de datos. Con la reciente aparición de sensores 3D de bajo coste dotados de sensores de imagen y de información 3D la tendencia está cambiando. Ahora la mayoría de esfuerzos de investigación en el área de visión por computador se están dirigiendo hacia la utilización de datos 3D como principal fuente de información. Además, estos nuevos sensores son capaces de ofrecer información a una frecuencia de vídeo. Destacar que la utilización de datos 3D está permitiendo resolver muchos problemas que hasta ahora permanecían sin solución: reconocimiento de

gestos en 6 grados de libertad [Oikonomidis et al., 2011], reconstrucción e interacción con entornos 3D [Izadi et al., 2011], o la localización y mapeado simultáneo de robots en exteriores [Kerl et al., 2013]. Por otro lado, el procesamiento de información geométrica 3D acarrea un mayor coste computacional por lo que varias de las aplicaciones mencionadas anteriormente han sido implementadas sobre dispositivos aceleradores como la GPU. Kinect Fusion [Izadi et al., 2011] es un claro ejemplo de este tipo de implementaciones donde Graphics Processing Unit (GPU) actúa como procesador principal de cómputo.

Tradicionalmente, en visión por computador se han utilizado técnicas de representación intermedias capaces de resolver problemas de reconocimiento y seguimiento de entidades en la observación [Boeres et al., 2004]. Además, estas representaciones permitían abstraer la información de los sensores en unidades abstractas de alto nivel, así como zonas de la imagen de especial interés (ROI), grafos y etiquetas [Wiskott et al., 1997, García-Rodríguez et al., 2006b, Cootes and Taylor, 2001, Florez-Revuelta et al., 2002, Kass et al., 1987]. Al movernos a un espacio 3D, son necesarios nuevos modelos de representación para poder seguir aplicando técnicas similares. Sin embargo, la mayoría de técnicas de representación 3D existentes están demasiado enfocadas hacia la industria del videojuego (realidad virtual) o la fabricación de modelos 3D siendo la mayoría demasiado costosas computacionalmente o poco robustas para modelar el ruido de estos nuevos sensores [Pomerleau et al., 2012, Khoshelham and Elberink, 2012]. Como acabamos de introducir, estos nuevos sensores presentan niveles de ruido elevados, sobre todo cuando se empieza a operar en los límites del rango de los mismos, por lo que es necesario un modelo de representación capaz de manejar ruido y posible información errónea.

La investigación descrita en esta tesis está motivada por la necesidad de plantear un modelo robusto de representación 3D capaz de representar la información obtenida con sensores 3D de bajo coste, así como el modelado del ruido que estos sensores producen. Además, debido a que estos sensores son capaces de proveer un flujo de datos 3D en tiempo real, a semejanza de las cámaras de vídeo tradicionales, surge un nuevo problema asociado al coste computacional de los nuevos métodos de computación geométrica:

la computación en tiempo real. Por lo tanto, una nueva restricción se incorpora en la búsqueda de un modelo de representación 3D para visión por computador, la necesidad de encontrar un modelo computacionalmente robusto y eficiente.

En esta tesis se propone la utilización de mapas auto-organizativos [Kohonen, 1995] como modelo de representación 3D. En concreto proponemos la utilización de gases neuronales crecientes (GNG) [Fritzke, 1995], que han sido utilizados con éxito tanto para clustering y reconocimiento de patrones [Sledge and Keller, 2008, Doherty et al., 2005, Qin and Suganthan, 2004] como para la representación topológica de conjuntos de datos principalmente en 2 dimensiones [Florez-Revuelta et al., 2002, Cselenyi, 2005, García-Rodríguez and García-Chamizo, 2011]. Hasta ahora, los mapas auto-organizativos han sido principalmente utilizados sin restricciones temporales y su utilización sobre datos 3D se ha centrado en modelos sin ruido y sin restricciones temporales [Cretu et al., 2008, Cretu et al., 2009, Holdstein and Fischer, 2008, Alonso-Montes and González Penedo, 2004]. Los gases neuronales crecientes tienen la capacidad de representar de forma adecuada el espacio de entrada que se les proporcione. Además, son un modelo eficaz debido a su flexibilidad, rápida adaptación y excelente calidad de representación. Sin embargo, este tipo de modelos son muy costosos computacionalmente, especialmente para datos de entrada con alta dimensionalidad. Debido a que muchas aplicaciones de visión por computador se ejecutan bajo restricciones temporales, es necesario adaptar el proceso de aprendizaje de la red neuronal para ser completado bajo un tiempo predefinido. En esta tesis se propone una implementación hardware aprovechando la capacidad de cómputo de las GPUs actuales bajo el paradigma GPGPU [?, Kirk and Hwu, 2010, Che et al., 2008, Wagner et al., 2013]. Finalmente se propone también la implementación de un descriptor característico capaz de extraer información geométrica de la escena que pueda ser posteriormente utilizado para describirla. El método propuesto debe ser aplicable a distintos problemas o aplicaciones dentro del área de visión por computador tales como el reconocimiento, localización de objetos, la vigilancia visual o la reconstrucción 3D.

## B.2 Objetivos de la investigación

Una vez presentada la motivación de esta tesis y analizado el estado del arte sobre modelos de representación 3D que actualmente existen (Ver Capítulo 1), se ha identificado un área todavía no explorada en el campo de aplicación de estos modelos 3D para la representación y reconstrucción de datos en el área de visión por computador. La mayoría de modelos de representación y reconstrucción 3D están actualmente orientados hacia el área de gráficos por computador (videojuegos, renderizado, visualización de datos, etc.). Además, entre los pocos trabajos que se encuentran publicados actualmente la mayoría están enfocados hacia el procesamiento de datos 2D, sin considerar el uso de estas técnicas de representación intermedia sobre distribuciones de datos 3D y en particular para datos obtenidos de sensores de bajo coste. Estos nuevos sensores presentan ventajas y desventajas, como se demuestra a lo largo de esta tesis. Una de las principales desventajas es la cantidad de ruido que presentan las tomas de datos obtenidas con estos sensores. Por otra parte, una de las ventajas es su capacidad de ofrecer hasta 25 capturas por segundo en ambos sensores, tanto en el de imagen como en el de profundidad (3D). Esta ventaja ha permitido solucionar problemas que hasta ahora permanecían sin resolver en el área de visión por computador.

De esta forma, basándonos en nuestra experiencia en la utilización de Gases Neuronales para la representación de formas 2D en varios problemas de visión por computador, proponemos extender este modelo de representación basado en mapas auto-organizativos. Esta extensión contempla el manejo de adquisiciones de datos 3D capturadas con distintos sensores. Además, se propone que este modelo sea capaz de operar bajo restricciones temporales y de manejar secuencias de nubes de puntos. Los gases neuronales, también conocidos como mapas auto-organizativos crecientes, son intrínsecamente paralelos, por lo tanto, gracias a este atributo también se propone su implementación sobre GPUs utilizando técnicas de computación de propósito general sobre GPUs (GPGPU). Esperamos obtener una considerable aceleración en el tiempo de ejecución respecto a las tradicionales implementaciones sobre CPU. La computación paralela

y las redes neuronales son un nuevo paradigma de computación que está atrayendo gran atención por parte de los investigadores en los últimos años.

Además, gracias a las propiedades topológicas del mapa creado por este método, proponemos su uso para la adquisición de modelos en problemas de Diseño Asistido por Computador (DAC) y también para la representación de escenas, siendo capaz de manejar la presencia de ruido causada por la mayoría de sensores 3D. Adicionalmente, utilizaremos esta estructura para una posterior detección de puntos de interés y extracción de características, permitiéndonos resolver problemas relacionados en el área de visión por computador.

El objetivo principal de esta investigación es la propuesta y validación de un método de representación 3D para objetos y escenas, el cual debe ser capaz de ayudar a resolver problemas de visión por ordenador 3D como los ya mencionados en la Sección B.1. El modelo de representación propuesto tiene que ser capaz de operar bajo restricciones temporales y será flexible para representar no sólo observaciones estáticas, sino también los cambios dinámicos en la observación. De esta forma, el modelo de representación seleccionado se utilizará como representación intermedia para la detección de puntos 3D significativos y algoritmos de extracción de características. Además, con la llegada de nuevos sensores 3D de bajo coste, el modelo propuesto será capaz de operar directamente sobre estos flujos de datos que por lo general contienen ruido. Por lo tanto, se propone una implementación hardware basada en la GPU como procesador principal con el fin de lograr una considerable aceleración respecto a implementaciones ya existentes en la CPU.

Como objetivo secundario, pero directamente relacionado con el primero, el modelo propuesto se validará en diferentes aplicaciones de visión por computador: visión robótica, sistemas de visión para CAD, vigilancia visual y reconstrucción 3D.

### B.3 Algoritmo red GNG

GNG es un algoritmo de clustering incremental no supervisado que dada una distribución de entrada en  $\mathbb{R}^d$  crea incrementalmente un grafo,

o red de nodos, donde cada nodo en el grafo tiene una posición en  $\mathbb{R}^d$ . El modelo puede ser utilizado para la cuantización de vectores encontrando así las diferentes agrupaciones presentes en una determinada distribución. Estos vectores significativos son representados por los vectores referencia (posición) de los nodos. Puede ser también utilizado para localizar estructuras topológicas que reflejan cierta similitud respecto a la estructura del espacio de entrada. El aprendizaje de la red GNG es un algoritmo dinámico, en el sentido de que si la distribución de entrada cambia ligeramente en el tiempo, la red tiene la capacidad para adaptarse moviendo los nodos hacia la nueva posición del espacio de entrada.

Empezando con dos nodos, el algoritmo crea un grafo en el cual los nodos son vecinos si están conectados por una arista. La información de vecindad es mantenida a lo largo de la ejecución por medio de una regla de aprendizaje competitivo Hebbiano (CHL). El grafo generado por CHL crea una “triangulación inducida de Delaunay” que es un sub-grafo de la triangulación de Delaunay correspondiente al conjunto de nodos. La triangulación inducida de Delaunay preserva de forma óptima la topología de entrada [Martinetz and Schulten, 1994]. CHL es un componente esencial del algoritmo de aprendizaje de la GNG ya que es utilizado para gestionar la adaptación local de los nodos que conforman la red y la inclusión de nuevos nodos a esta.

La red se especifica como:

- Un conjunto  $N$  de nodos (neuronas). Cada neurona  $c \in N$  tiene su vector de pesos asociado  $w_c \in \mathbb{R}$ . El vector de pesos puede ser considerado como la posición en el espacio de entrada de sus correspondientes neuronas.
- Un conjunto de aristas (conexiones) entre pares de neuronas. Estas conexiones no tienen pesos y su propósito es definir una estructura topológica. Un esquema basado en la antigüedad de las aristas es utilizado para eliminar aquellas conexiones que han dejado de ser validas debido al movimiento de las neuronas en el proceso de adaptación.

Se utilizan parámetros constantes en el tiempo. Además, no es necesario definir el número de nodos utilizados a priori, ya que los nodos son

añadidos de forma incremental durante la ejecución del algoritmo. La inserción de nodos se detiene cuando se alcanza un criterio de rendimiento definido por el usuario o alternativamente el tamaño máximo de la red es alcanzado. El proceso de adaptación de la red a los vectores del espacio de entrada se produce en el paso 6. La inserción de conexiones (paso 4) entre la neurona ganadora y la segunda más cercana a la muestra del espacio de entrada establece la relación topológica entre las neuronas (Figura 2.5). La eliminación de conexiones (paso 8) elimina las aristas que han dejado de pertenecer a la topología. Esto se realiza eliminando las conexiones entre neuronas que no están ya cerca o que tienen otras neuronas más cercanas, de forma que la edad de las aristas entre estas neuronas excede un umbral. La acumulación del error (paso 5) sirve para identificar aquellas áreas del espacio de entrada donde es necesario incrementar el número de neuronas para mejorar la representación.

## B.4 Representación de observaciones 3D: mapas auto-organizativos crecientes

Debido a que en la mayoría de trabajos relacionados con Mapas Auto-Organizativos (SOMs) se centran en la representación de datos estacionarios, que habitualmente han sido procesados previamente y por tanto no presentan ruido, proponemos la utilización del método GNG para la representación de observaciones del mundo real, capturadas utilizando sensores que presentan distintos niveles de ruido. Para ello, hemos estudiado las distintas propiedades por parte del modelo de representación elegido para operar bajo distintos niveles de ruido. Además, debido a que la mayoría de sensores 3D actuales permiten trabajar con un flujo de imágenes en tiempo real y que la mayoría de aplicaciones de visión por computador suelen estar acotadas temporalmente, se ha extendido la versión original del algoritmo GNG para mejorar la adaptación de la red ante observaciones que varían dinámicamente en el tiempo. Finalmente, se ha considerado también información de color sobre el propio modelo así como la capacidad de generar reconstrucciones 3D automáticamente.

### B.4.1 Eliminación de ruido

Con la reciente aparición de sensores 3D de bajo coste capaces de ofrecer información 3D, han aparecido muchas aplicaciones en el área de visión por computador, que utilizan esta información para solucionar problemas hasta ahora sin resolver. Aplicaciones en robótica móvil, reconocimiento de objetos y escenas, reconocimiento de gestos, etcétera. Sin embargo, estos sensores presentan un elevado nivel de ruido, por lo que técnicas habitualmente utilizadas en el procesamiento de nubes de puntos no pueden ser aplicadas directamente, requiriendo un filtrado previo de estos datos. El método propuesto es capaz de crear representaciones 3D del espacio de entrada filtrando el ruido y además creando una representación reducida de los datos. La red GNG, mediante un proceso de aprendizaje, es capaz de adaptar los vectores de referencia de las neuronas así como la interconexión de estas para obtener una representación que preserve la topología del espacio de entrada. En este proceso de adaptación, la red neuronal considera el ruido y los artefactos que puedan existir en los datos originales. Gracias a este proceso, la red crea una triangulación de Delaunay inducida que representa el espacio de entrada. Aprovechando esta triangulación es posible además extraer características útiles como esquinas, bordes o zonas con una elevada curvatura.

El método propuesto se ha comparado con uno de los métodos del estado del arte en técnicas de filtrado, el método Voxel Grid (VG). En la siguiente sección se muestra cómo nuestro método ofrece mejores resultados en cuanto a adaptación al espacio de entrada y además eliminación de ruido.

Hemos realizado varios experimentos sobre distintas escenas para evaluar la efectividad y robustez del método propuesto. Primero, se ha llevado a cabo la estimación de normales con el fin de demostrar cómo características simples como las normales en un punto 3D están afectadas por el ruido presente en los datos de entrada. Segundo, se ha demostrado la capacidad de adaptación de la red neuronal GNG calculando el Mean Squared Error (MSE) sobre nubes de datos obtenidas utilizando un simulador del sensor Kinect. Este simulador nos permite conocer información real sobre datos sin ruido, generados de forma sintética, de manera que posteriormente se

pueden modelar distintos niveles de ruido para el posterior cálculo en el error de adaptación con distintos métodos. Los experimentos, el manejo y la visualización de información 3D han sido desarrollados utilizando la librería PCL<sup>1</sup>. Esta librería ha sido utilizada para implementar, comprobar y visualizar la mayoría de experimentos. Varios algoritmos desarrollados en este trabajo ya han sido incluidos en la librería, una vez defendida y publicada la tesis se espera contribuir a la librería mencionada con varios de los métodos propuestos en este trabajo.

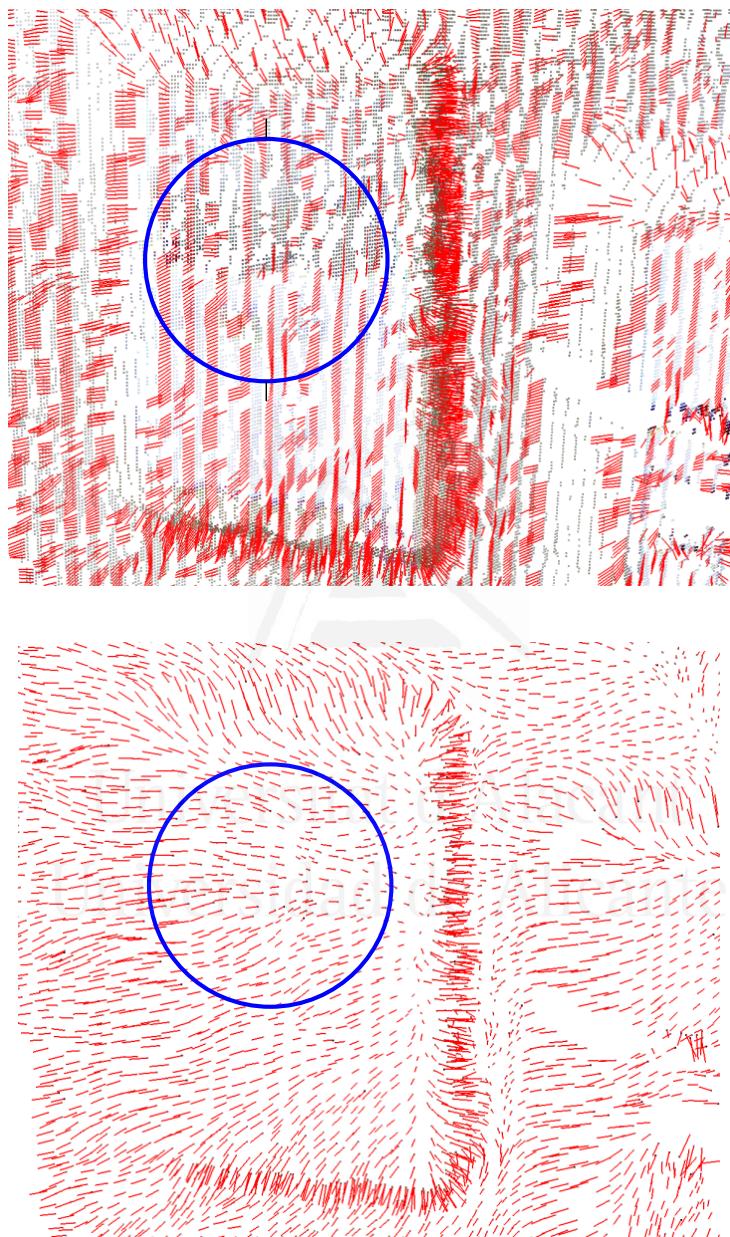
En la Figura B.1 se muestra cómo las normales estimadas sobre la nube filtrada son más estables, ya que sus direcciones a lo largo de superficies planas apenas varían, mientras que las normales estimadas sobre los datos sin filtrar presentan cambios de dirección poco predecibles incluso sobre superficies planas. Se utilizó una superficie plana para llevar a cabo el experimento de manera que se pudiera visualmente apreciar los cambios bruscos en la dirección de las normales estimadas. En la representación filtrada se utilizaron 20,000 neuronas y 1,000 patrones como parámetros de configuración de la red.

#### B.4.2 Filtrado: adaptación al espacio de entrada

Se han llevado a cabo varios experimentos para demostrar la capacidad de adaptación al espacio de entrada del método GNG. Estos experimentos se llevaron a cabo sobre datos sintéticos simulados, de forma que se dispusiera de datos base para el cálculo del error de la representación generada. Para este propósito se ha utilizado el software Blensor [Gschwandtner et al., 2011]. Este software nos permite generar escenas sintéticas y posteriormente extraer vistas parciales de la escena como si un sensor 3D como la cámara Kinect las estuviese capturando. La Figura B.2 muestra una vista parcial capturada utilizando el software Blensor. La escena capturada sin ruido se muestra en la parte superior de la figura, mientras que en la parte inferior mostramos la vista parcial capturada simulando un sensor con cierto nivel de ruido. El ruido se modeló intentando imitar el ruido presente en la mayoría de sensores 3D, para ello, se utilizó una

---

<sup>1</sup>Point Cloud Library (PCL) es un proyecto abierto a gran escala para el procesamiento de imágenes y nubes de puntos [Rusu and Cousins, 2011]



**Figura B.1:** Comparativa estimación normales. Arriba: Estimación de normales sobre la nube de puntos sin filtrar. Abajo: estimación de normales sobre la representación generada por el método GNG.

distribución Gaussiana con media cero y distintos factores de desviación. En la Figura B.3 se muestran además objetos utilizados para llevar a cabo el mismo experimento sobre objetos en lugar de escenas.

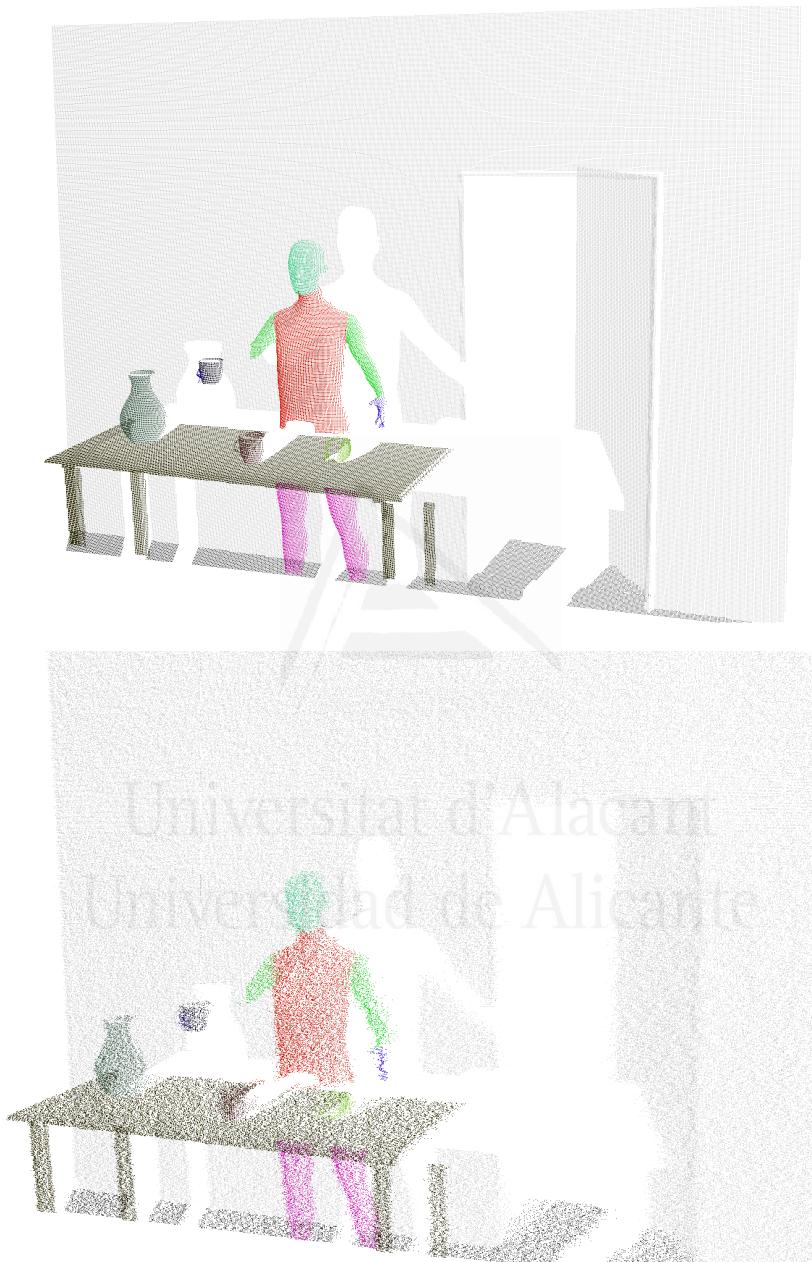
Para realizar este experimento, hemos computado el MSE de las nubes filtradas con los métodos VG y GNG respecto a los datos originales sin ruido. Este error nos permite conocer de forma cuantitativa la precisión de ambos métodos. Para los experimentos, en los que se utiliza la vista parcial de la escena, el MSE viene expresado en metros, mientras que para los modelos CAD el error viene dado en milímetros. VG presenta algunas desventajas como la imposibilidad de establecer un número de puntos fijo, ya que el número de puntos de la nube filtrada viene dado por el tamaño del voxel definido. En este experimento, hemos explorado distintos tamaños de voxel hasta encontrar el número de puntos deseados, de forma que la comparación entre ambos métodos sea equitativa. Sin embargo, la GNG a diferencia de VG nos permite especificar el número de neuronas o puntos que conformarán la representación final.

<b>Escenas simuladas</b>	<b>VG 5000</b>	<b>VG 10000</b>	<b>GNG 5000 250<math>\lambda</math></b>	<b>GNG 10000 500<math>\lambda</math></b>
<b>escena 1</b> $\sigma = 0.15$	0.0067	0.0064	<b>0.0017</b>	0.0022
<b>escena 1</b> $\sigma = 0.25$	0.0197	0.0181	<b>0.0053</b>	0.0065
<b>escena 1</b> $\sigma = 0.40$	0.0475	0.0430	<b>0.0156</b>	0.0185
<b>escena 2</b> $\sigma = 0.15$	0.0053	0.0051	<b>0.0013</b>	0.0017
<b>escena 2</b> $\sigma = 0.25$	0.0148	0.0135	<b>0.0041</b>	0.0051
<b>escena 2</b> $\sigma = 0.40$	0.0372	0.0336	<b>0.0122</b>	0.0143

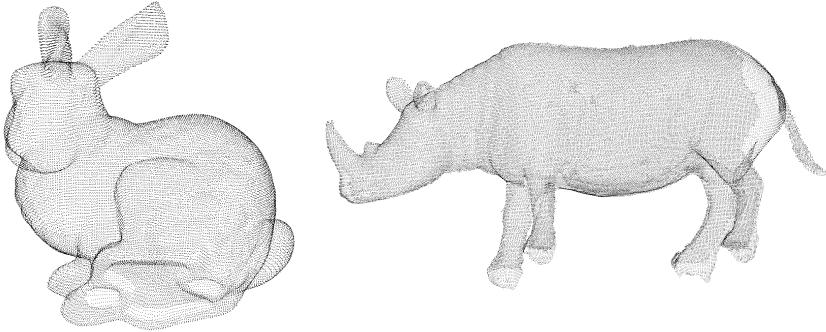
<b>Modelos CAD</b>	<b>VG 5000</b>	<b>VG 10000</b>	<b>GNG 5000 250<math>\lambda</math></b>	<b>GNG 10000 500<math>\lambda</math></b>
<b>modelo 1</b> $\sigma = 0.15$	0.0643	0.0641	0.0684	<b>0.0559</b>
<b>modelo 1</b> $\sigma = 0.25$	0.0981	0.0994	0.0768	<b>0.0642</b>
<b>modelo 1</b> $\sigma = 0.40$	0.2037	0.2276	<b>0.0903</b>	0.0924
<b>modelo 2</b> $\sigma = 0.15$	0.1540	0.1504	0.1756	<b>0.1209</b>
<b>modelo 2</b> $\sigma = 0.25$	0.3055	0.3227	0.1938	<b>0.1430</b>
<b>modelo 2</b> $\sigma = 0.40$	0.8259	0.8895	0.2346	<b>0.2122</b>

**Tabla B.1:** MSE de adaptación al espacio de entrada para distintos modelos. VG vs GNG. Los valores en negrita destacan los mejores resultados obtenidos.

En la Tabla B.1 se muestra el MSE de adaptación al espacio de entrada para distintos modelos y escenas utilizando los métodos GNG y VG. En la parte superior de la tabla se muestran los errores de adaptación al espacio de entrada para distintas vistas parciales adquiridas con un sensor simulado, mientras que en la parte inferior se calcula el error para



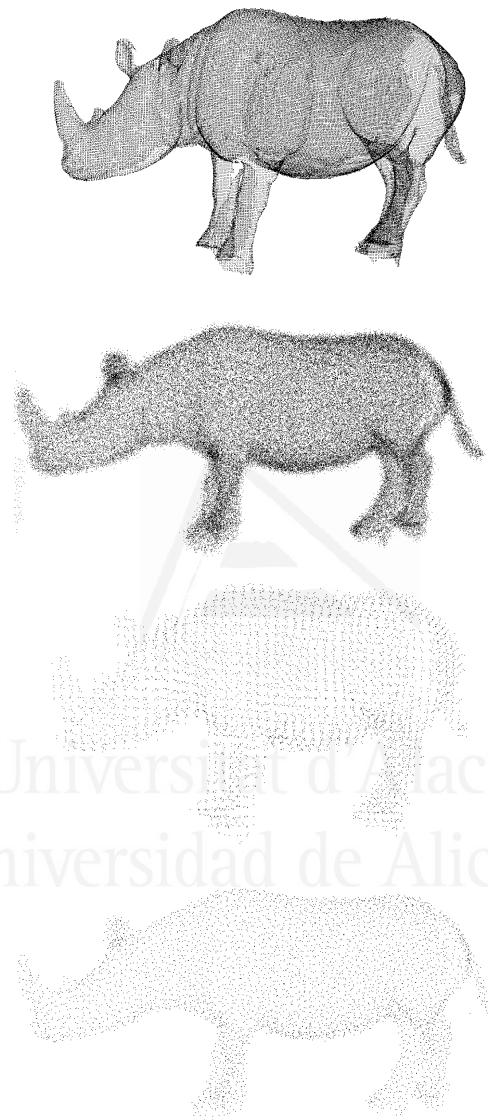
**Figura B.2:** Escena sintética. Arriba: Vista parcial generada sin ruido. Abajo: Vista parcial generada simulando un dispositivo Kinect con ruido Gaussiano  $\sigma = 0.5$  y  $\mu = 0.0$ .



**Figura B.3:** Izquierda: modelo 3D conejo (34,834 puntos). Derecha: modelo 3D rinoceronte (79,934 puntos).

modelos de objetos. Los resultados presentados en la Tabla B.1 muestran cómo el método GNG ofrece un error medio más bajo y por lo tanto una mejor adaptación al espacio de entrada, ofreciendo una mejor calidad de representación en áreas con una curvatura pronunciada y eliminación de ruido. El método VG filtra el espacio de entrada aproximando los puntos internos a un voxel mediante su centroide. Esta forma de filtrar la nube de puntos no permite representar de forma precisa el conjunto original produciendo un mayor error en la adaptación. Los experimentos fueron realizados utilizando distintas configuraciones en cuanto a número de puntos en la representación final, y en el caso concreto de la GNG con distintos valores para el número de entradas  $\lambda$  generadas por iteración. En la Figura B.4 se muestran los valores presentados en la Tabla B.1 en concreto para el modelo CAD 2 (Rinoceronte). La representación producida por VG no se ajusta correctamente al espacio de entrada, especialmente representando de forma imprecisa bordes y puntos esquina.

En la Figura B.5 se muestran escenas de interior y exterior capturadas con distintos sensores 3D: láser 3D montado sobre un brazo robótico, cámara de tiempo de vuelo y una cámara RGB-Depth (RGB-D) basada en luz estructurada. Como se puede apreciar, la red neuronal GNG es capaz de producir representaciones reducidas, manteniendo la topología original del espacio de entrada. Gracias a la interconexión entre las neuronas es posible explotar fácilmente la búsqueda en la vecindad de cada neurona



**Figura B.4:** Calidad de filtrado utilizando 10,000 ppuntos. Comparativa GNG vs Voxel Grid. De arriba a abajo: Modelo original, modelo con ruido  $\sigma = 0.4$ , modelo filtrado utilizando el método GNG y modelo filtrado utilizando VG.

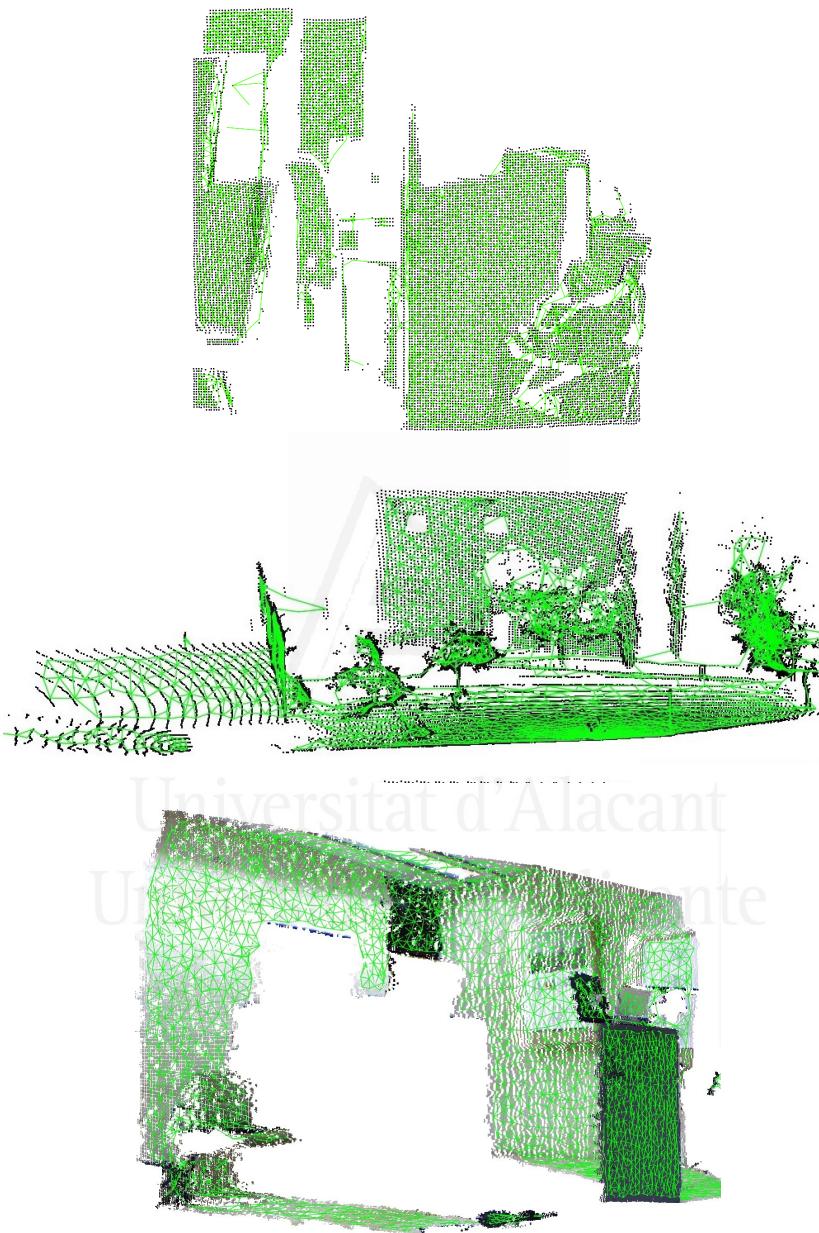
para post-procesamientos como la estimación de normales.

En la Figura B.6 se muestra la representación obtenida mediante el método propuesto para distintas capturas de objetos tomadas utilizando el sensor Kinect de Microsoft.

### B.4.3 Aprendizaje información de color

Debido a que la mayoría de sensores 3D modernos ofrecen información de color, el método de representación propuesto basado en la red neuronal GNG ha sido extendido considerando la información de color del espacio de entrada. La dimensionalidad del espacio de entrada se ha incrementado de 3 a 6 añadiendo información de color en formato RGB. De este modo, los datos de entrada están definidos en  $\mathbb{R}^d$  donde  $d = 6$ . La mayoría de métodos basados en SOM presentados en la revisión del estado del arte solo consideran información espacial dentro de los vectores de referencia de las neuronas  $w_c$ , de esta forma hemos modificado el algoritmo de aprendizaje añadiendo información de color a los vectores de referencia de las neuronas y considerando su ajuste también durante este proceso. Para poder incluir información de color ha sido necesario normalizar esta información de manera que los colores tomen valores entre 0.0 y 1.0. Sin embargo, la información de color no se incluye en el proceso CHL debido a que nuestro principal interés en la utilización de este mapa auto-organizativo es la representación de la topología y la adaptación al espacio de entrada. Por lo tanto, en la etapa de búsqueda de neuronas ganadoras solo se considera la distancia Euclídea. La Figura B.7 muestra cómo la extensión propuesta del método de la GNG es capaz de generar una representación reducida del espacio de entrada capturando también el color original. El aprendizaje provoca un efecto de interpolación de color sobre la representación generada respecto al espacio original, además se sigue conservando la representación de la topología y la adaptación al espacio de entrada. Esta modificación ha sido denominada Colour-GNG.

Además, se ha implementado un método de post-procesado capaz de añadir color a la red neuronal interpolando de forma ponderada el color del espacio de entrada. El objetivo de este método es comparar los resultados obtenidos con la extensión del algoritmo GNG para manejo de color y



**Figura B.5:** Diferentes escenas representadas utilizando la red neuronal GNG. De arriba a abajo: escenas capturada utilizando la cámara de tiempo de vuelo SR4000, un laser LMS-200 Sick montado sobre un brazo robótico y el sensor RGBD Kinect de Microsoft



**Figura B.6:** Distintos objetos representado utilizando la red neuronal GNG. El modelo 3D de Mario pertenece al conjunto de datos presentado en [Tombari et al., 2010a]. El resto de objetos fueron capturados utilizando el dispositivo Kinect de Microsoft.

observar la diferencia de resultados de forma cuantitativa respecto a un método más preciso y que requiere tiempo adicional de cómputo.

De esta forma, el método propuesto como línea base para compararnos computa para cada neurona que conforma la red neuronal el color que debe tomar la misma utilizando los patrones de entrada más cercanos. Para ello se computa un nuevo valor ponderando los valores de los k-vecinos más cercanos utilizando como factor de peso la distancia Euclídea entre los patrones y la neurona en concreto. Los K-vecinos más cercanos se obtienen realizando una búsqueda en un radio definido como parámetro de entrada. Por lo tanto, el cálculo del color para cada neurona queda definido de la siguiente forma:

$$RGB_i = \psi \sum_{\forall j \in N_i} (RGB_j \cdot w(j - i)) \quad (\text{B.1})$$

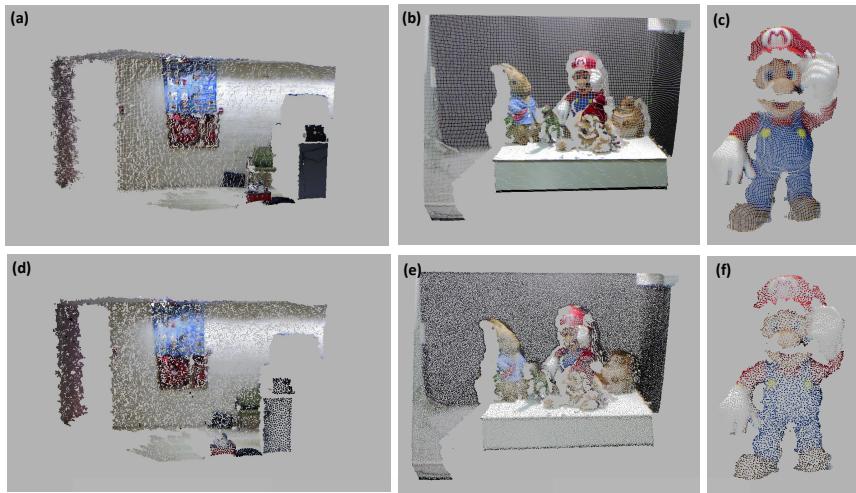
donde  $N_i$  representa el i-ésimo patrón de entrada más cercano a la neurona  $i$ ,  $i$  es la neurona procesada y  $w(j - i)$  es la función de ponderación basada en la distancia Euclídea entre el patrón vecino  $j$  y la propia neurona. La función de ponderación decrece exponencialmente a medida que la distancia hacia el patrón de entrada aumenta.  $\psi$  es un factor de normalización que permite a las componentes RGB tomar valores entre 0.0 y 1.0.

$$w(j - i) = e^{-\|j-i\|} \quad (\text{B.2})$$

La Figura B.8 muestra como se lleva a cabo la interpolación de color, utilizada como método para compararnos, llevando a cabo un post-procesamiento.

Las Figuras B.9 y B.10 muestran distintas observaciones que han sido creadas utilizando ambos métodos. Colour GNG produce un mapa que representa de forma correcta la información de color del espacio de entrada original, produciendo una versión reducida y consistente con información de color. Además, el método propuesto ofrece resultados similares comparado con métodos aplicados en etapas posteriores (interpolación de color), como demostraremos cuantitativamente en la TablaB.2.

Finalmente, se ha analizado cuantitativamente la diferencia entre ambos métodos y se ha calculado el error medio existente entre ambas representaciones, tomando como base los resultados obtenidos por la inter-



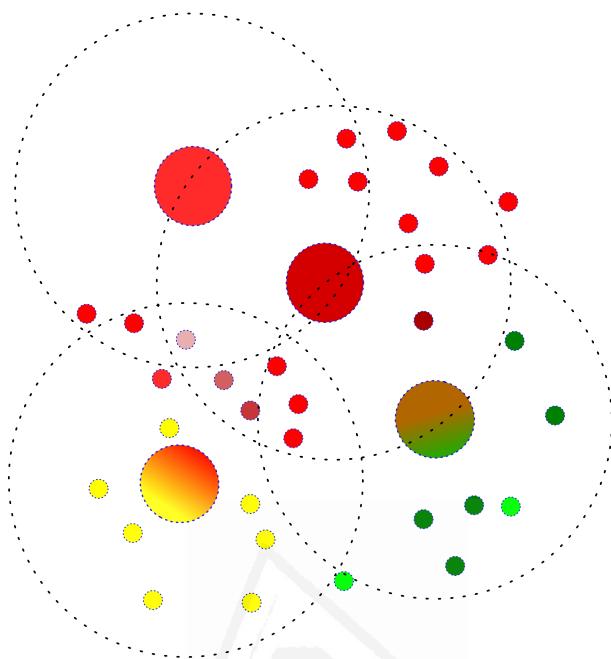
**Figura B.7:** Representación de distintos objetos y escenas utilizando el método propuesto Colour GNG. En (a),(b),(c) se muestran las nubes de puntos originales. En (d),(e),(f) se presentan las versiones reducidas utilizando el método GNG con información de color.

polación ponderada (Tabla B.2). El error se ha calculado sobre las tres componentes de color. Como se puede observar el error máximo obtenido entre la interpolación y el método Color GNG difiere como máximo en 4 unidades, considerando que las componentes toman valores entre 0 y 255.

	Error medio		
	Rojo	Verde	Azul
<b>Escena 1 -</b> $20,000n$ $50\lambda$	1	1	2
<b>Escena 1 -</b> $50,000n$ $100\lambda$	1	1	1
<b>Escena 2 -</b> $20,000n$ $50\lambda$	4	4	4
<b>Escena 2 -</b> $50,000n$ $100\lambda$	2	2	2
<b>Objeto 1 -</b> $3,000n$ $50\lambda$	2	3	3
<b>Objeto 1 -</b> $5,000n$ $100\lambda$	1	2	2
<b>Objeto 2 -</b> $3,000m$ $50\lambda$	3	3	3
<b>Objeto 2 -</b> $5,000n$ $100\lambda$	2	2	2

**Tabla B.2:** Error medio entre el color calculado utilizando la técnica de post-procesamiento (interpolación ponderada) y el método Color GNG.

Con estos resultados podemos concluir que el método propuesto per-

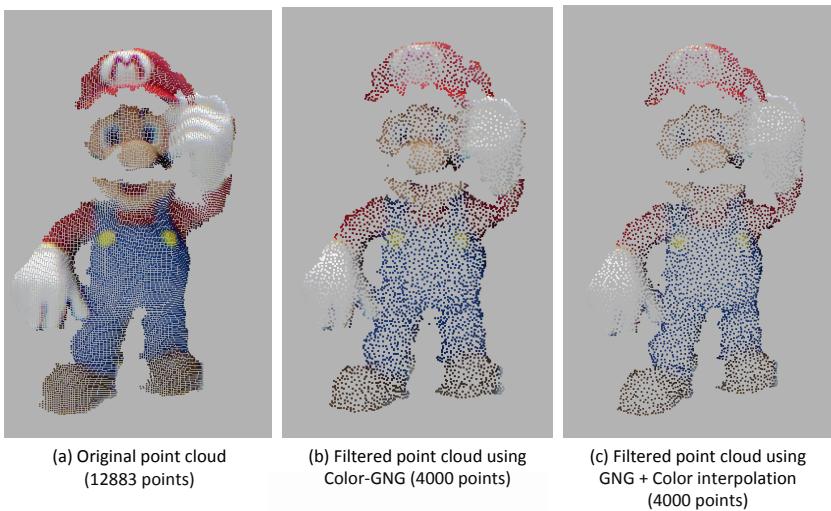


**Figura B.8:** Método de post-procesado para interpolación de color. El color asignado a cada neurona (círculos grandes) es calculado como la media ponderada de los patrones de entrada vecinos (círculos pequeños) en un radio de búsqueda. Radio de búsqueda vecinos más cercanos (circunferencias con línea discontinua).

mite obtener resultados similares comparado con otras técnicas más complejas, como la presentada, y además permite reducir el tiempo de computación.

## B.5 Implementación GPU algoritmo GNG

Con el fin de acelerar el algoritmo de la GNG se propone la implementación de este sobre arquitecturas masivamente paralelas como las GPUs. Para llevar a cabo este proceso de aceleración se ha rediseñado el algoritmo con el objetivo de conseguir aprovechar al máximo los recursos ofrecidos por la GPU. Muchas de las operaciones realizadas por el algoritmo son idóneas para ser paralelizadas debido a que no existe dependencia directa entre las neuronas a nivel operacional. Sin embargo, sí que existe un nivel de dependencia o sincronización en la etapa de ajuste de



**Figura B.9:** Representación reducida con información de color utilizando el método Colour GNG. Los resultados son visualmente similares a los producidos utilizando una interpolación ponderada sobre el espacio de entrada.

la red durante las distintas iteraciones del algoritmo. Este factor, requiere la sincronización explícita de la ejecución en paralelo al final de cada iteración. En la Figura B.11 se pueden apreciar las etapas del algoritmo que han sido paralelizadas en la GPU.

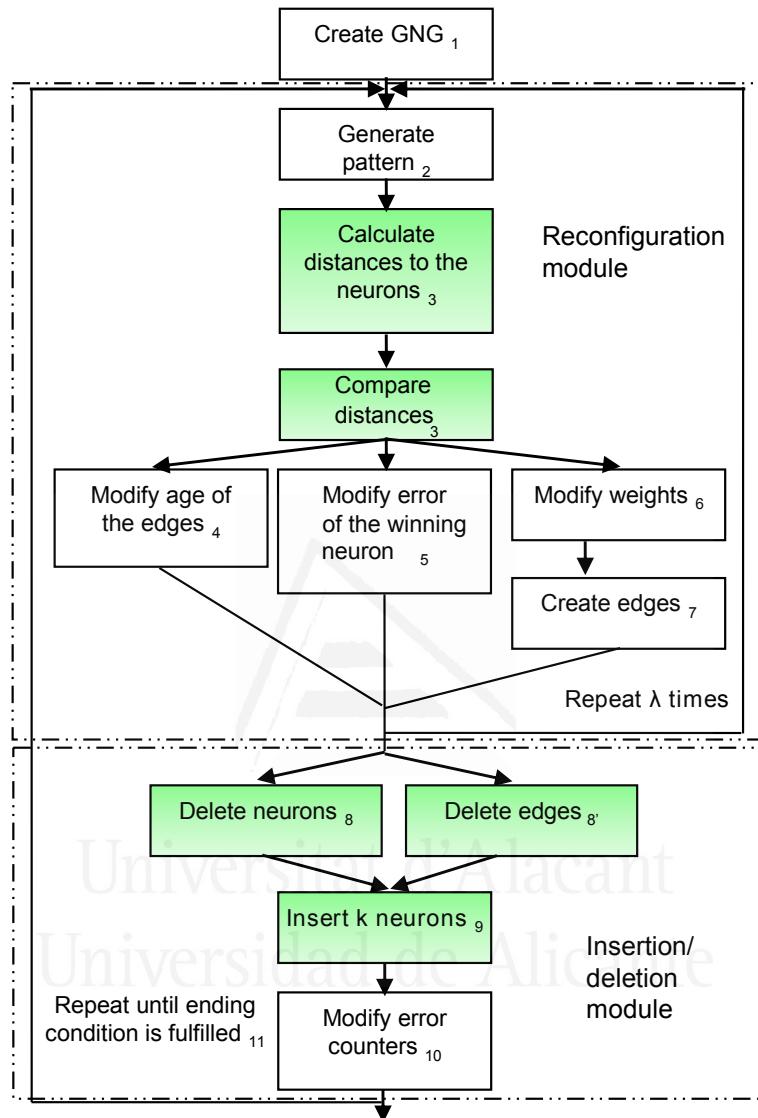
### B.5.1 Cálculo distancias Euclídeas

La primera etapa susceptible de ser acelerada en el algoritmo es el cálculo de distancias Euclídeas realizado en cada iteración. Esta etapa calcula la distancia Euclídea entre un patrón de entrada aleatorio y el resto de neuronas. Esta tarea puede llevarse a cabo en paralelo computando el cálculo de distancias Euclídeas en paralelo utilizando tantos hilos de ejecución en la GPU como neuronas conforman la red neuronal. Para un número de neuronas muy elevado sería posible calcular más de una distancia por hilo de ejecución, pero esto es solo eficiente para vectores muy grandes, donde el número de bloques de hilos ejecutados en la GPU es también elevado. Este número también viene condicionado por la capacidad del modelo de GPU donde se ejecuta el algoritmo, ya que diferentes modelos tienen un



**Figura B.10:** Representación reducida con información de color de dos escenas diferentes.

número de procesadores distinto y por lo tanto la capacidad de cómputo también varía. Esto afecta directamente con la capacidad para ejecutar en paralelo una mayor o menor cantidad de hilos.



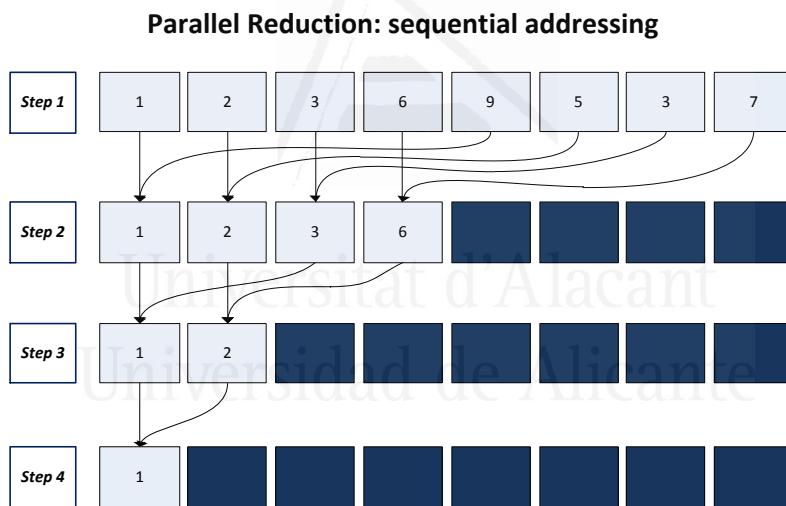
**Figura B.11:** Algoritmo GNG. Se han marcado las etapas paralelizadas.

### B.5.2 Reducción paralela

La segunda etapa del algoritmo que presenta un alto nivel de paralelismo es la búsqueda de la neurona ganadora en cada iteración. En concreto, se deben calcular las dos neuronas con distancia mínima al patrón de entrada. Para acelerar esta búsqueda nos hemos basado en la técnica de reducción paralela originalmente presentada en [Harris, 2008].

Esta técnica acelera la búsqueda de valores máximos, mínimos u otras operaciones binarias en grandes conjuntos de datos.

En concreto, en nuestra implementación GPU del algoritmo GNG hemos modificado la implementación original de la reducción paralela que solo obtiene el valor mínimo. De esta forma, hemos diseñado una aproximación capaz de obtener en el mismo número de pasos los dos valores mínimos del conjunto de datos. En nuestro caso las dos neuronas con menor distancia al patrón de entrada. Esta nueva versión la hemos denominado *2MinParallelReduction*. En la Figura B.12 se muestra la reducción paralela como un árbol binario donde al final de  $\log_2(n)$  pasos obtenemos el resultado final de una operación binaria sobre un conjunto de  $N$  elementos.



**Figura B.12:** Ejemplo de reducción paralela.

Finalmente, otras etapas similares a las presentadas anteriormente han sido también paralelizados. De esta forma la ejecución completa del algoritmo se lleva a cabo en la GPU. En concreto, los pasos 8 y 9 presentan un comportamiento similar al cálculo de distancias Euclídeas y reducción paralela, respectivamente. La diferencia reside en la operación a computar (búsqueda de aristas con una edad superior a un umbral) y búsqueda de

la neurona con mayor error respecto al espacio de entrada (inserción neuronas).

### B.5.3 Factor de aceleración

Para validar nuestra implementación paralela del algoritmo de la red GNG hemos llevado a cabo varios experimentos donde la red es utilizada para la representación de datos 3D. Este problema en concreto requiere de un número elevado de neuronas para obtener una representación de escenas conformadas por muchos puntos. Además para obtener una representación precisa es necesario presentar a la red neuronal un número elevado de patrones de entrada  $\lambda$ . La implementación GPU ha sido ejecutada sobre distintos modelos de GPU con distinta capacidad de computación (Ver Tabla B.3).

Modelo	Arquitectura	Multi-procesadores	Núcleos	Memoria global	Ancho de banda
Quadro 2000	2.1	4	192	1 GB	41.6 GB/s
GeForce GTX 480	2.0	15	480	1.5 GB	177.4 GB/s
Tesla C2070	2.0	14	448	6 GB	144 GB/s

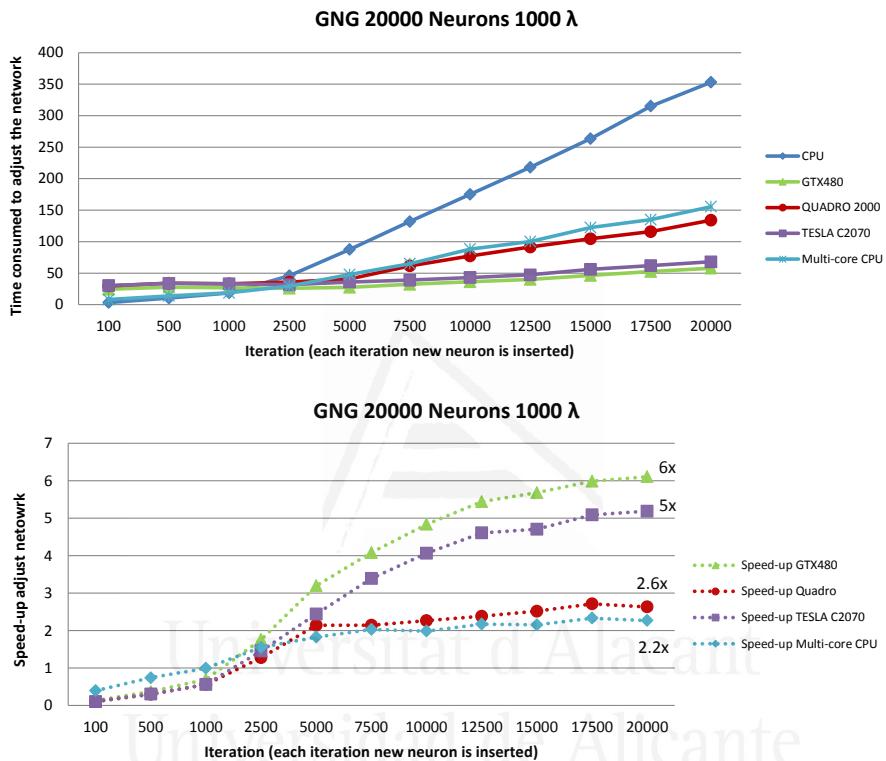
**Tabla B.3:** Dispositivos compatibles con CUDA utilizados en los experimentos.

Basándonos en trabajos previos [García-Rodriguez et al., 2011], la implementación ha sido testeada utilizando distintos parámetros: número de neuronas  $N$  igual a 1000 / 5000 / 10000 / 200000 y patrones de entrada  $\lambda$  igual a 500/1000.  $\epsilon_w = 0.1$ ,  $\epsilon_n = 0.001$ ,  $\alpha = 0.5$ ,  $\gamma = 0.95$ ,  $a_{max} = 250$ .

En la Figura B.13 se muestran los tiempos de ejecución obtenidos utilizando la GNG para representar una distribución de datos 3D con 20,000 neuronas y 1,000 patrones de entrada. En este experimento la implementación CPU incrementa su tiempo de ejecución a medida que el número de neuronas insertadas en la red crece. Sin embargo, la versión paralela ejecutada sobre la GPU permite insertar neuronas sin degradar el rendimiento. Para un número de 20k neuronas se obtiene un factor de aceleración de  $6x$  respecto a la CPU utilizando el modelo de GPU NVIDIA GTX 480.

Sin embargo, en la Figura B.13 se puede apreciar como la versión CPU es más rápida durante las primeras iteraciones. Debido a este efecto, proponemos una versión híbrida capaz de aprovechar toda la capacidad de

cómputo de ambos procesadores y obtener un tiempo de ejecución óptimo. La GPU obtiene un tiempo de ejecución mayor en las primeras iteraciones debido a las latencias asociadas a la gestión de hilos y a la división del problema para su resolución paralela.

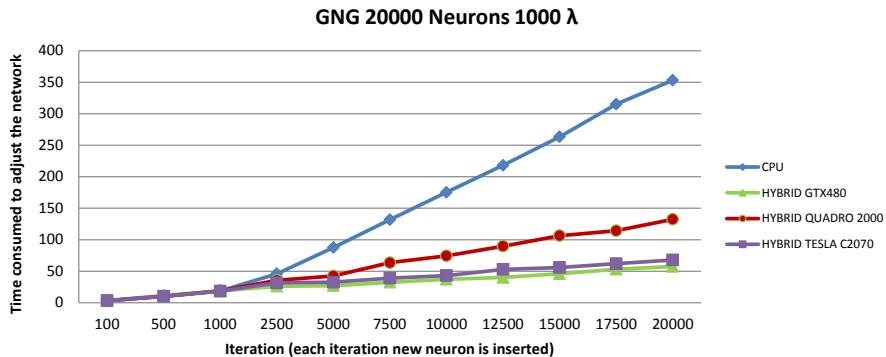


**Figura B.13:** Tiempo de ejecución en la CPU y GPU. Se muestran los factores de aceleración obtenidos sobre distintos modelos de GPU.

#### B.5.4 Versión Híbrida GNG

La versión híbrida se propone debido a que algunas aplicaciones requieren operar bajo restricciones temporales obteniendo una solución con una calidad específica en un período de tiempo. La versión híbrida asegura un rendimiento óptimo en este tipo de aplicaciones al utilizar toda la capacidad de cómputo de ambos procesadores. (Ver Figura B.14).

La aplicación conmuta entre la CPU y la GPU aprovechando la capaci-

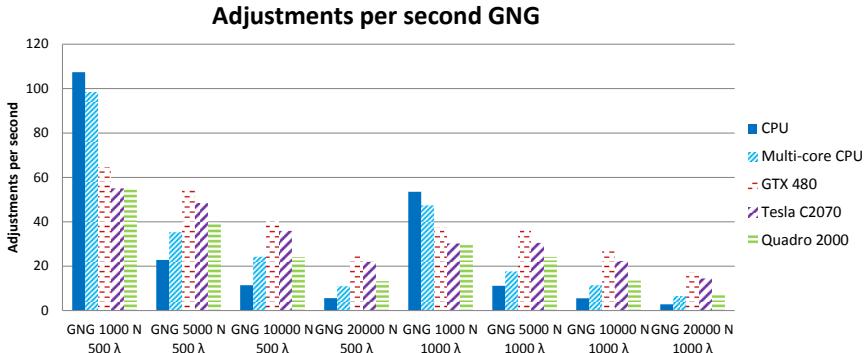


**Figura B.14:** Tiempos de ejecución implementaciones CPU e híbrida sobre distintas GPUs.

dad de cómputo de ambos procesadores. Cuando se detecta que la CPU ofrece un mayor rendimiento (primeras iteraciones) el aprendizaje de la red se lleva a cabo en la CPU, mientras que una vez que se detecta que el tiempo de cómputo de una iteración es inferior en la GPU, se conmuta de forma que el procesamiento se traslada a la GPU. La iteración en la que se debe conmutar depende de la capacidad de cómputo de la GPU utilizada.

### B.5.5 Número de ajustes por segundo

Finalmente, se llevaron a cabo una serie de experimentos que muestran cómo la versión acelerada de la GNG no es solo capaz de acelerar el proceso de aprendizaje de la red neuronal, sino que también nos permite obtener un número mayor de ajustes (actualización) de la red respecto al espacio de entrada. En los experimentos realizados obtuvimos un número de ajustes más elevado en la versión GPU comparado con versiones secuenciales ejecutadas sobre la CPU y sobre implementaciones paralelas ejecutadas también sobre una CPU con varios núcleos. La implementación paralela sobre procesadores multi-núcleo fue desarrollada utilizando Intel Thread Building Blocks y el esquema de parallelización fue similar al ya presentado para la GPU. En la Figura B.15 se muestran las diferentes tasas obtenidas en ajustes por segundo para diferentes tarjetas gráficas con distinto número de núcleos. Al incrementar el número de neuronas en las versiones CPU la tasas de ajustes por segundo disminuye considerablemente.



**Figura B.15:** Tasas de ajustes por segundo obtenidas para distintos procesadores.

## B.6 Extracción de características 3D en tiempo real

El principal objetivo de esta subtarea dentro de la tesis ha sido el estudio de la aplicación de arquitecturas masivamente paralelas en el procesamiento de información 3D y su integración en sistemas de visión 3D con restricciones de tiempo real. La información 3D ha sido obtenida principalmente en forma de mapas RGB-D proporcionados por el sensor Kinect. Para ello se propone la utilización de la GPU como dispositivo principal de cómputo acelerando la extracción de un descriptor característico basado en parches de superficies 3D. Este descriptor es computacionalmente costoso de calcular debido a la información que requiere para su procesamiento: información de las normales de los puntos de la escena y triangulación de la superficie. Además el propio cálculo del descriptor es costoso computacionalmente.

### B.6.1 Procesamiento información RGBD en la GPU

El dispositivo Kinect nos proporciona un mapa RGB con información de color  $M_c$  y un mapa de disparidad sin normalizar  $M_d$ . El primer paso a llevar a cabo en la GPU, con el objetivo de acelerar todos los procesos es la proyección de la información de profundidad y color a un espacio tridimensional.

sional, donde ademas esta informacion se encuentre alineada, obteniendo asi una nube de puntos con informacion de color de la escena. La relación entre un mapa de disparidad proporcionado por la Kinect y un mapa normalizado viene dado por  $d = 1/8 \cdot (d_{off} - kd)$ , donde  $d$  es la disparidad normalizada,  $kd$  es la disparidad proporcionada por la Kinect y  $d_{off}$  es el offset particular de un dispositivo Kinect. Los valores de calibración se pueden extraer en la etapa de calibración [Konolige and Mihelich, 2013] del dispositivo. De esta forma la relación entre profundidad y disparidad viene dada por la siguiente fórmula:

$$z = \frac{b \cdot f}{1/8 \cdot (d_{off} - kd)} \quad (\text{B.3})$$

Donde  $b$  es la línea base entre la cámara de infrarojos y la cámara RGB (en metros),  $f$  es la distancia focal de las cámaras (en píxeles). Una vez obtenido el mapa de profundidad  $M_d$ , calculando la profundidad  $z$  para todos los puntos, la proyección de cada uno de los puntos al espacio 3D, junto con su informacion de color, viene dada por:

$$p_x = z \cdot (x - x_c) \cdot 1/f_x$$

$$p_y = z \cdot (y - y_c) \cdot 1/f_y$$

$$p_z = z$$

Donde  $p \in \mathbb{R}^3$ ,  $x$  e  $y$  son la fila y columna del píxel a proyectar,  $x_c$  e  $y_c$  son las distancias (en píxeles) al centro del mapa y  $f_x$  y  $f_y$  son las distancias focales del dispositivo Kinect obtenidas durante la calibración del dispositivo [Konolige and Mihelich, 2013].

Esta transformación se puede computar de forma independiente para cada píxel del mapa, de forma que encaja perfectamente en arquitecturas masivamente paralelas como la GPU, acelerando respecto a la CPU el tiempo de procesamiento. Debido a que esta transformación suele ir seguida de otras operaciones sobre los puntos 3D obtenidos también en la GPU, no es necesario copiar de vuelta los datos a la memoria de la CPU y evitamos así la latencia ocasionada por estas transferencias. En la Sección B.6.6 podemos ver los tiempos de ejecución y aceleración obtenidas.



**Figura B.16:** Izquierda. Mapa de profundidad. Centro. Mapa RGB. Derecha. Nube de puntos proyectada.

### B.6.2 Eliminación de ruido: filtro bilateral

El sensor Kinect utiliza un patrón organizado de luz infrarroja para obtener la profundidad de los objetos observados. Este método para obtener información 3D de la escena presenta problemas cuando las superficies de los objetos presentan un índice alto de especularidad, reflexión, ante la incidencia de la luz, haciendo imposible que el sensor obtenga información de profundidad de los mismos [Zhang, 2012]. Lo mismo ocurre en este caso para objetos que se encuentran muy alejados del sensor. Por ello, si queremos extraer información coherente de la escena observada es necesario previamente corregir en la medida de lo posible el error de la observación. Hasta ahora para corregir el error y seguir manteniendo un elevado número de imágenes por segundo en el procesamiento, se había optado por la utilización de filtros simples como el filtro media o mediana. Gracias a la capacidad de cómputo de la GPU se pueden aplicar filtros más complejos capaces de reducir el error del mapa de profundidad sin eliminar información importante, tal como los bordes de los objetos. Un ejemplo de este tipo es el filtro Bilateral [Tomasi and Manduchi, 1998], el cual permite eliminar ruido de la imagen manteniendo información de los bordes. Este filtro ha sido utilizado originalmente en imágenes de color y escalas de grises para reducir el ruido conservando los bordes, pero se puede aplicar de igual forma para reducir el ruido presente en los mapas de profundidad generados por cámaras 3D como la Kinect. El filtrado bilateral elimina ruido de la imagen promediando el valor de un píxel usando la vecindad de este. Para conseguir preservar los bordes se realiza un promedio de

los píxeles vecinos considerando la distancia espacial entre los píxeles y la similaridad de su valor. El nuevo valor filtrado viene dado por:

$$P_f = \frac{1}{K_p} \sum_{q \in \omega} V_q f(\|p - q\|) g(\|V_p - V_q\|) \quad (\text{B.4})$$

Donde  $K_p$  es un factor de normalización,  $\omega$  es la vecindad del píxel,  $I_q$  es el valor del píxel consultado y  $J_p$  es el valor filtrado del pixel  $p$ . En esta ecuación también reconocemos las funciones kernel  $f(\|p - q\|)$  y  $g(\|I_p - I_q\|)$ . Generalmente,  $f$  y  $g$  son tomadas como dos funciones Gaussianas con desviación estándar  $\sigma_s$  y  $\sigma_r$ .

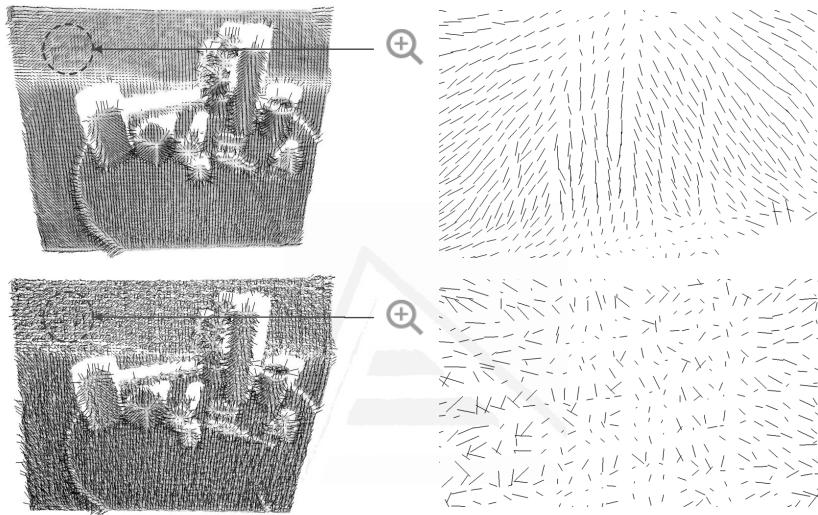
En la Sección 4.5.1.2 se demuestra como la estimación de las normales mejora tras aplicar el filtro bilateral, proporcionando unas normales mas estables al eliminar el ruido original presente en el mapa de profundidad. La mayoría de características 3D extraídas de una escena se basan en la curvatura de la geometría, la cual es calculada usando la información de las normales en cada punto de la escena, por lo que la obtención de unas normales estables nos conduce a una característica mas precisa de la escena. Este proceso puede ser añadido como paso previo al cómputo de la red neuronal GNG como vimos en la sección anterior, generando una representación reducida con un mayor nivel de filtrado de ruido.

El cálculo de los valores filtrados de cada píxel de la imagen se pueden calcular de forma independiente y por tanto paralela sobre arquitecturas como la GPU. En [Chan et al., 2008] y en [Wasza et al., 2011] se proponen implementaciones GPU del filtro para su utilización en aplicaciones en tiempo real. El tiempo de ejecución se mejora considerablemente, permitiendo filtrar los mapas de profundidad generados por el sensor en tiempo real.

### B.6.3 Estimación de normales

Una vez tenemos la nube de puntos organizada previamente almacenada en la memoria de la GPU, la estimación de las normales utilizando Análisis de Componentes Principales (PCA) puede llevarse a cabo en la GPU de forma eficiente. El cálculo de las normales se realiza en la GPU de forma independiente sobre cada punto de la nube, acelerando consider-

ablemente el tiempo de ejecución. Por otro lado, gracias a la eliminación previa de ruido utilizando el filtro bilateral las normales extraídas son mucho más estables que las normales calculadas directamente sobre el mapa original y tienen en cuenta los bordes y puntos esquina de la escena. En la Figura B.17 se puede apreciar este efecto.

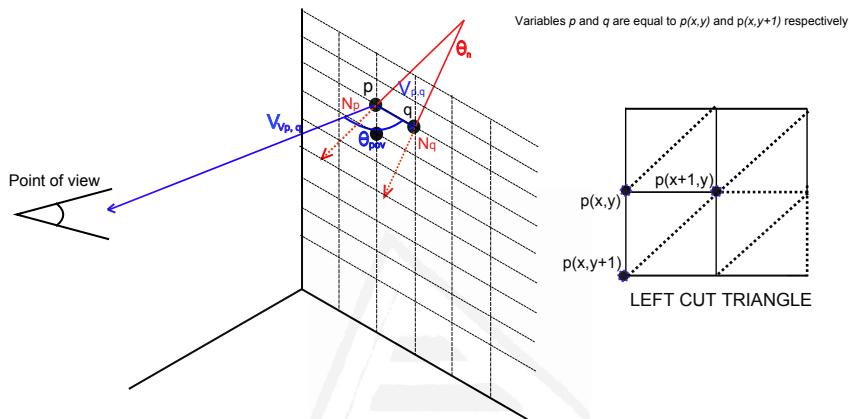


**Figura B.17:** Arriba: Estimación de normales sobre el mapa filtrado (filtro bilateral). Abajo: Estimación de normales sobre el mapa original. Se puede observar como la estimación de las normales mejora, obteniendo normales mas estables, sobre todo en las zonas planas, donde el ruido del mapa original tiene como consecuencia unas normales inestables.

#### B.6.4 Triangulación de superficies en la GPU

En esta sección se propone una versión acelerada y robusta del método propuesto en [Holz and Behnke, 2012] para triangular nubes de puntos organizadas. En este, se propone la triangulación de los puntos 3D obtenidos a partir de cámaras de rango o de luz estructurada, donde los puntos 3D se pueden acceder utilizando su organización en forma de matriz, utilizando  $x$  para especificar la fila e  $y$  la columna, accediendo de esta forma al punto  $p_{x,y}$ . De esta forma se obtiene la superficie de la nube de puntos capturada

por el sensor. El método asume que el punto de vista es conocido y de esta forma calcula la diferencia en ángulo entre el vector formado por el punto de vista  $v_p$  y el punto consultado  $p_{x,y}$ , y los puntos vecinos  $p_{x+1,y}$  y  $p_{x,y+1}$ , de forma que si existe ocultación, se descarta la posible triangulación. En la Figura B.18 podemos observar visualmente esta condición.



**Figura B.18:** Condición para triangular vértices. En esta imagen se puede apreciar cómo la condición de arista válida establece que el ángulo  $\theta_{pov}$  formado por los vectores  $v_{v_p,p}$  y  $v_{p,q}$  debe ser superior a un umbral establecido  $\epsilon_{\theta_{pov}}$ . Este umbral nos asegura que los puntos no producen una occlusion entre ellos. La distancia euclídea entre  $p$  y  $q$  también debe ser inferior a un umbral  $T_d$ , calculado de forma dinámica de acuerdo a la resolución de la nube de puntos y su desviación estándar.

Nuestro método es más robusto debido a que utilizamos la información de las normales previamente calculada en cada punto de la escena como condición adicional de validación. Además, debido a que la triángulación de los puntos se puede hacer de forma independiente, el algoritmo se ha portado a la GPU, donde cada hilo de ejecución de la GPU comprueba si el punto a consultar forma un triángulo válido con sus vecinos. Como resultado, obtenemos un vector con todos los triángulos, aunque para mantener organizada la nube de puntos, se crean triángulos inválidos en aquellos puntos que no satisfacen las restricciones. La condición para crear una arista entre dos puntos queda formulada de la siguiente forma:

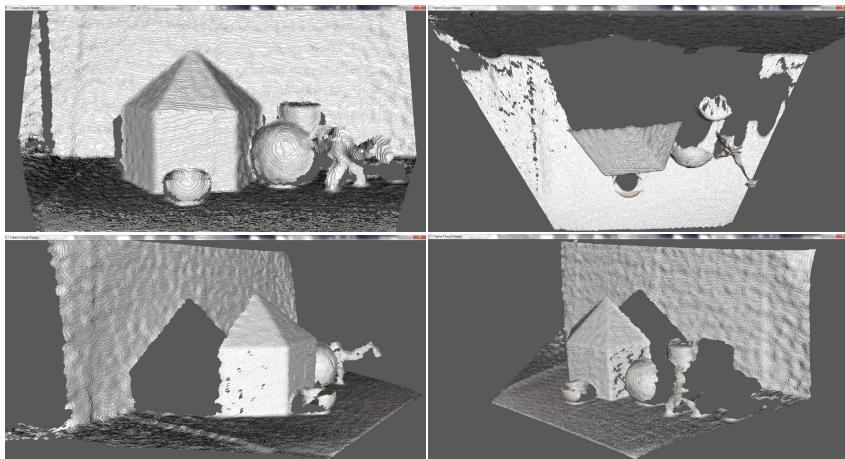
$$edge_{valid} = (|v_{v_p,p} \cdot v_{p,q}| \geq \cos \epsilon_{\theta_{pov}}) \wedge (||p - q||^2 \leq T_d) \wedge (|n_p \cdot n_q| \leq \cos \epsilon_{\theta_n}) \quad (B.5)$$

Dónde  $\epsilon_{\theta_{pov}}$  es el ángulo mínimo existente entre dos puntos y el punto de vista para que no se produzca oclusión. El valor de este ángulo se ha establecido en  $12.5^\circ$  basándose en el análisis visual presentado en la Figura 4.18 y los resultados del trabajo previamente citado [Holz and Behnke, 2012].  $T_d$  representa la distancia máxima entre dos puntos, esta es obtenida en tiempo real basándose en la resolución de la nube de puntos. Para ello se calcula la distancia media en cada punto con su vecindad  $k$ . Basándose en la media y la desviación estándar de todos estos valores se obtiene el umbral  $T_d$  como:  $T_d = \bar{d}_k + \sigma_d$ . Donde  $\bar{d}_k$  es la media de todas las distancias medias calculadas en cada punto de la nube y  $\sigma_d$  es la desviación estándar. Por último,  $\epsilon_{\theta_n}$  es el umbral establecido para el ángulo máximo entre dos normales. Este se calcula siguiendo el mismo procedimiento que para  $T_d$ .

El método propuesto nos permite obtener de forma rápida una triangulación aproximada suficientemente precisa para computar en tiempo real el descriptor propuesto. Nuestro método se aprovecha del conocimiento del punto de vista de la escena y de tener previamente pre-calculado en la memoria de la GPU las normales de todos los puntos de la escena. La implementación GPU consigue tiempos de ejecución considerablemente inferiores a la versión CPU, alcanzando frecuencias de procesamiento alrededor de 30 imágenes por segundo para mapas de profundidad de 640x480 píxeles, mientras la versión CPU obtiene una frecuencia de procesamiento de 6 imágenes por segundo. En la Figura B.19 podemos observar el resultado de la triangulación de una escena.

### B.6.5 Computación del tensor 3D en la GPU

Una vez disponemos de la nube de puntos con información sobre sus normales e información sobre la superficie, hay que extraer el eje base sobre el cual se va a calcular el tensor. Para ello, parejas de puntos son seleccionadas en base a la distancia entre ellos y la orientación de sus



**Figura B.19:** Reconstrucción de la superficie de la escena utilizando el método propuesto. Se puede observar como la reconstrucción presenta algunos huecos debido a que el sensor no ofrece información de esas zonas, bien por encontrarse fuera del área de visión, o por ser una superficie especular.

normales. Para evitar la búsqueda sobre las  $C_2^n$  combinaciones posibles y obtener además parejas de puntos con un elevado carácter de unicidad, se aplica una restricción en la distancia mínima existente entre estos puntos para que formen una pareja válida. Además, esta restricción maximiza la probabilidad de encontrar estos puntos en una región solapada con otras vistas parciales. La distancia permitida entre parejas de vértices se define en base a la resolución de la nube de puntos, utilizando como  $d_{min} = pcl_{res} * 5$  y  $d_{max} = pcl_{res} * 14$ . El cálculo de la resolución de la nube en tiempo real nos permite que estas restricciones se ajusten dinámicamente, sin importar si el sensor está más próximo o más lejano a la escena observada. Además de la restricción en la distancia, también se añade una restricción en el ángulo formado por las normales de esta pareja de puntos, de forma que el ángulo formado por las normales sea superior a  $5^\circ$ , evitando así que el producto cruzado entre estas sea igual a 0 y por tanto se pueda utilizar como eje del sistema de coordenadas del tensor. Cada punto se empareja como máximo con sus 3 vecinos más cercanos, limitando así el número de posibles parejas generadas.

El cálculo de parejas es acelerado respecto a la CPU utilizando tantos hilos en paralelo como puntos en la escena. De esta forma cada hilo de la GPU comprueba si su punto correspondiente forma una pareja con algún punto de su vecindad. Además, para mejorar el rendimiento, debido a que la nube de puntos se encuentra organizada en forma de matriz, y cada posición de esta matriz corresponde de forma aproximada con la resolución media de la nube, la búsqueda en la vecindad se acota dinámicamente para cada nube, reduciendo el espacio de búsqueda a una ventana alrededor del píxel consultado. En la comparación de tiempos con la versión CPU, la misma técnica ha sido aplicada para acelerar también la búsqueda en la vecindad.

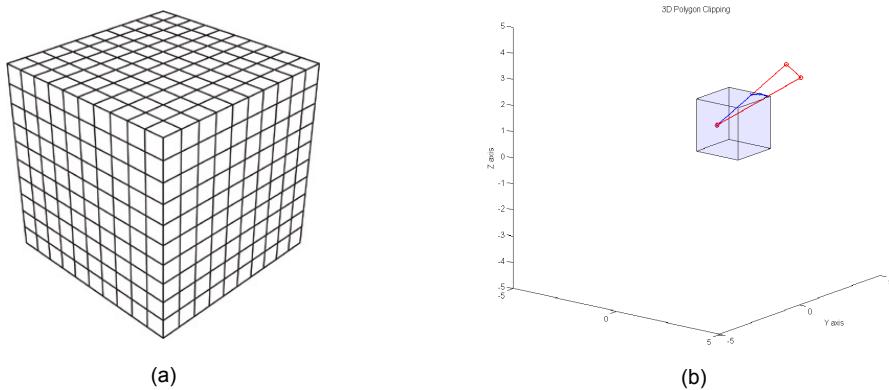
Con la lista de parejas válidas, se define un sistema de coordenadas 3D sobre cada una de las parejas de la siguiente forma: El punto medio de la línea que une ambos puntos define el origen del eje de coordenadas. La media de las normales de ambos puntos definen el eje Z. El producto cruzado de las dos normales define el eje X y finalmente el producto cruzado de los ejes X y Z definen el eje Y. Este nuevo sistema de coordenadas es utilizado como centro para definir el grid 3D. Adicionalmente, esta información también es calculada en paralelo en la GPU, una vez que el hilo de ejecución comprueba que se trata de una pareja válida.

Para el cálculo del grid, que definirá el descriptor característico de la escena, es necesario definir dos parámetros: el número de véxeles que componen el grid  $n_{voxels}$  y el tamaño de cada uno de estos véxeles  $voxel\_size$ . Variando el número de véxeles y por lo tanto el tamaño del grid, hacemos que el descriptor contenga información local, semi-local o global a la escena. En los experimentos realizados en [Mian et al., 2006a] se demuestra que para el problema de reconocimiento de objetos de tamaño medio, un tamaño de  $10 \times 10 \times 10$ , permite extraer descriptores con información semi-local a la escena permitiendo identificar los descriptores bajo un cierto nivel de ocultación. El tamaño del voxel  $voxel\_size$  se define de forma dinámica de acuerdo a la resolución de la nube de puntos. Definido el grid, se calcula el área de la superficie de la escena que intersecta con cada uno de los véxeles, almacenando estos valores en forma de un tensor 3D. Para el cálculo del área intersectada dentro de cada voxel del grid se ha utilizado el al-

goritmo de Sutherland Hodman's [Foley et al., 1990] para calcular el área intersectada por cada polígono que conforma la superficie de la escena. Dado que más de un polígono puede intersectar más de un voxel, el área intersectada se acumula en cada posición del tensor 3D. Para evitar tener que recorrer todos los triángulos de la escena se aplica en ambas versiones, CPU y GPU, un algoritmo creciente de forma que se evita recorrer todos los triángulos de la escena. Comenzando en el triángulo contenido en el centro de coordenadas del grid se consultan todos los triángulos vecinos de forma recursiva, hasta que todos los triángulos consultados quedan fuera del voxel. Esta medida aumenta el rendimiento del algoritmo de forma considerable, sobre todo al trabajar con nubes de puntos que contienen un número elevado. Finalmente, los tensores extraídos pueden ser comprimidos agrupando todas las posiciones con valores distinto de 0. De esta forma se reduce el espacio ocupado.

El cálculo de cada tensor es acelerado considerablemente en la GPU al no existir dependencia entre el cálculo del área intersectada en cada uno de los voxels del grid, ni existir dependencia entre distintos tensores de la escena. De esta forma,  $N \times N \times N$  hilos son ejecutados en la GPU organizados en forma de grid. Cada hilo se encarga de computar el área intersectada por el voxel que representa y almacenar en su posición el área intersectada (Figura B.20). El acceso al voxel correspondiente y el almacenamiento del área son acelerados gracias a la posibilidad de ejecutar los hilos y acceder a ellos utilizando índices 3D. El algoritmo de Sutherland Hodgman's [Foley et al., 1990] es ejecutado por cada uno de los hilos en paralelo. Además, el cálculo de varios tensores se solapa en la GPU de forma que varios tensores son computados al mismo tiempo aprovechando todos los recursos de la GPU.

Todos los tensores extraídos de una vista parcial son almacenados junto con el sistema de coordenadas y el ángulo entre las normales. Esta información permite agrupar todos los tensores con un ángulo entre normales similar dentro de un mismo bin de una tabla hash. De esta forma, el ángulo entre normales puede ser utilizado posteriormente para extraer descriptores similares de forma eficiente. Esta colección de tensores puede ser construida durante la fase de entrenamiento y utilizada posteriormente en la fase



**Figura B.20:** (a) Ejecución de  $Dim_x \times Dim_y \times Dim_z$  hilos en paralelo donde cada hilo de la GPU representa un voxel del grid. (b) Cada hilo con índices  $i, j, k$  computa el área de intersección entre la superficie triangulada y su voxel correspondiente utilizando el algoritmo de Sutherland Hodgman's para cálculo de intersección de polígonos. Aprovechándose de los índices de los hilos proporcionados por CUDA el area es almacenado un vector unidimensional.

de test, obteniendo un alto grado discriminatorio en el reconocimiento de los mismos.

### B.6.6 Rendimiento

Las implementaciones GPU de los métodos presentados en las secciones anteriores han sido testeadas en un computador equipado con un procesador Intel Core i3 540 3.07Ghz y distintos modelos de GPU ya presentados anteriormente en la Tabla B.3. Las implementaciones GPU fueron inicialmente desarrolladas en un ordenador portátil equipado con un procesador Intel i5 3210M 2.5Ghz y una GPU compatible con CUDA. La Tabla B.4 muestra las características de esta GPU adicional utilizada en estos experimentos. Hemos utilizado diferentes modelos que abarcan desde el modelo integrado en un ordenador portátil a modelos con un número elevado de núcleos y que por tanto tienen un mayor consumo energético.

El rendimiento obtenido en las implementaciones GPU permite computar los métodos propuestos en tiempos inferiores a 1seg, habilitando la interacción por parte del usuario y utilización en aplicaciones en tiempo real. En la Tabla B.5 podemos ver los distintos métodos acelerados uti-

Modelo	Multi-procesadores	Núcleos	Mem. Global	Ancho de banda
GeForce GT630M	2	96	1 GB	32 GB/s

**Tabla B.4:** Modelo GPU integrado en un ordenador portátil utilizado en los experimentos

lizando la GPU y los distintos tiempos de ejecución obtenidos, así como la aceleración respecto a implementaciones secuenciales en la CPU. El mejor rendimiento se ha obtenido con el modelo GTX480, el cual posee el mayor número de cores y mayor ancho de banda de memoria, consiguiendo una aceleración considerable respecto a las implementaciones CPU.

Etapa	GT630M	GTX480	Q2k	CPU	GT630M	GTX480	Q2k
Filtrado bilateral mapa de profundidad	11ms	<b>5ms</b>	8ms	1008 ms	91.63x	<b>201.6x</b>	126x
Proyección nube de puntos	2ms	<b>1ms</b>	1ms	50ms	25x	<b>50x</b>	50x
Estimación de normales	9ms	<b>1ms</b>	8ms	190ms	21.11x	<b>190x</b>	23.75x
Computación distancias segmentación	13ms	<b>4ms</b>	11ms	101ms	7.76x	<b>25.25x</b>	9.18x
Triangulación superficie	5ms	<b>2ms</b>	4ms	121ms	24.25x	<b>40.3x</b>	30.25x
Cómputo resolución nube de puntos	7ms	<b>4ms</b>	6ms	330ms	47.14x	<b>82.5x</b>	55x
Búsqueda de parejas válidas	71ms	<b>9ms</b>	35ms	4479ms	63x	<b>497x</b>	127.97x
Cómputo de descriptor (tensor)	6ms	3ms	4ms	130 ms	31.6x	<b>43.33x</b>	32.5x
Tiempo total cómputo 200 tensores	854 ms	<b>490ms</b>	687ms	45887ms	53.72x	<b>93.64x</b>	66.79x

**Tabla B.5:** Comparación tiempos de ejecución y factor de aceleración obtenidos en los distintos métodos implementados en la GPU. Los mejores tiempos de ejecución se obtuvieron para el modelo GTX480, el cual posee el mayor número de núcleos de procesamiento.

## B.7 Conclusiones

En esta tesis doctoral se han presentado varios avances en el ámbito del procesamiento de nubes de puntos adquiridas con sensores 3D de bajo coste. Primero, se ha presentado un método basado en Growing Self-Organizing Maps (GSOM) para la creación de modelos 3D a partir de nubes de puntos 3D no organizadas. Estas nubes de puntos son adquiridas directamente desde sensores 3D sin necesidad de aplicar ningún preprocesamiento sobre ellas. Además, el método presentado no necesita conocer información sobre el sensor desde el cual se adquiere la información, por lo que se podría aplicar a cualquier nube de puntos. Se ha demostrado la capacidad de los GSOM para la representación de distribuciones de datos 3D, junto con su capacidad para preservar la topología de los datos. Esto

se ha demostrado con distintos estudios en los que se ha comparado con otros métodos como VG, mostrando una mejor adaptación al espacio original de entrada. Además de otras ventajas como la capacidad de incorporar color a los modelos 3D y trabajar con secuencias de adquisiciones.

Growing Neural Gas (GNG) se ha considerado uno de los métodos más adecuados para resolver el problema presentado, debido a las características que presenta: flexibilidad, rápida adaptación, preservación de la topología, eliminación de ruido, reducción de la dimensionalidad, etcétera. En esta tesis se ha demostrado la capacidad del método a través de distintos estudios sobre la parametrización de la red y un amplio conjunto de datos, tanto generados sintéticamente emulando ruido, como modelos del mundo real capturados con sensores de bajo coste. Además, el método propuesto se ha comparado con otros métodos similares del estado del arte como pueden ser Voxel Grid (VG) y Unifor Sampling (US), obteniendo una mejor adaptación al espacio de entrada tanto para escenas como para objetos simulados con distintos niveles de ruido.

Además, el algoritmo original de la GNG ha sido modificado para el manejo de secuencias de nubes de puntos. Esto acelera considerablemente el proceso de aprendizaje y permite al método obtener un mayor rendimiento. Esta modificación es posible debido a que el algoritmo no tiene que reiniciar el mapa creado para cada nueva adquisición. En su lugar, se ajusta el mapa creado en la adquisición anterior para adaptarlo al nuevo espacio de entrada. También se ha mejorado el método original, añadiendo información de color al proceso de aprendizaje de la red. De esta forma, los vectores de referencia de las neuronas, no solo almacenan la posición en el espacio Euclídeo, sino que también se considera el color. Esta modificación ha permitido eliminar el post-procesamiento de la representación final para añadir información de color. Finalmente, debido a que el algoritmo original de la GNG no produce triángulos, generando una representación alámbrica, se ha modificado la etapa Competitive Hebbian Learning (CHL) de forma que durante el proceso de aprendizaje además de interconectar las neuronas se crea una malla de triángulos 3D que representa de forma precisa el espacio de entrada, produciendo de esta forma un modelo 3D.

La versión mejorada de la GNG ha sido también utilizada de forma efectiva para mejorar el proceso de detección de puntos de interés en nubes de puntos. Este proceso es comúnmente utilizado en aplicaciones del área de visión por computador como el registro o el reconocimiento de formas. Las nubes de puntos capturadas con el sensor 3D Kinect, que contienen cierto nivel de ruido, han sido filtradas y reducidas usando el algoritmo propuesto. La estructura proporcionada por el método GNG nos ha proporcionado un modelo de representación compacto y reducido, eliminando cierta información, pero manteniendo la topología original del espacio de entrada. Se ha demostrado cómo la mayoría de algoritmos tradicionales utilizados para la detección de puntos de interés mejoran, requiriendo menos iteraciones para rechazar falsos positivos en el emparejamiento de los puntos de interés y mejorando por tanto la precisión del método. Además, este incremento en precisión y por tanto eliminación de ruido, ha sido validado en un proceso de registro de vistas parciales de una escena 3D. En el registro de estas vistas parciales se ha obtenido un error más bajo en la transformación final aplicada para alinear las vistas parciales. En la mayoría de combinaciones validadas de detectores de puntos de interés y descriptores se ha obtenido un error más bajo en la transformación calculada, cuando la estructura generada por la red GNG fue utilizada como información de entrada. Además el método propuesto obtuvo mejores resultados que otras técnicas, comúnmente utilizadas para la reducción y filtrado de nubes de puntos, como son VG y US.

Otro de los objetivos inicialmente planteados en esta tesis, la aceleración del algoritmo de aprendizaje de la GNG, ha sido llevado a cabo con éxito, obteniendo resultados positivos y novedosos en esta línea de investigación. El algoritmo de la GNG ha sido rediseñado y extendido con el fin de obtener una versión más eficiente, capaz de operar bajo restricciones temporales. Esta versión acelerada ha sido implementada utilizando la GPU como procesador principal para llevar a cabo la computación del proceso de aprendizaje. Como se ha demostrado en los experimentos, el tiempo de ejecución de la versión original del algoritmo ha sido acelerado de forma considerable. La implementación paralela desarrollada es capaz de aprovechar la capacidad de cómputo de las GPUs actuales bajo

el paradigma GPGPU, permitiendo aumentar el número de neuronas que conforma la red, sin afectar considerablemente al tiempo global de cómputo. Mientras que el tiempo de ejecución de la versión original ejecutada en la CPU crece de forma exponencial al incrementar el número de neuronas. Además, se ha demostrado que la versión paralelizada y rediseñada para la GPU no solo mejora respecto a la versión secuencial en la CPU, sino también respecto a una versión paralela también ejecutada en CPUs multi-núcleo. Finalmente, con el objetivo de obtener una versión que obtuviese el mayor rendimiento posible. Se ha presentado una versión híbrida capaz de aprovechar la capacidad de cómputo de ambos procesadores para obtener el máximo rendimiento y mejorando así el tiempo de ejecución global.

La versión extendida y mejorada del algoritmo GNG para representación de nubes de puntos ha sido validada en distintos casos de estudio. Demonstrando su capacidad para mejorar la resolución de problemas de distintas áreas: robótica, visión por computador y Computer-Aided Design (CAD). Ligeras modificaciones en el algoritmo han sido suficientes para aplicar el método en los distintos casos de estudio presentados. Todos los casos presentaban un problema en común, la necesidad de encontrar un modelo de representación capaz de eliminar ruido, representar de forma precisa y preservar la topología del espacio de entrada. Además, en muchos de los casos, la utilización de la representación generada ha reducido el tiempo de cómputo global de las aplicaciones, dotándolas además de una representación capaz de aprender la topología original de los datos representados.

La vasta capacidad de cómputo del novedoso paradigma de computación GPGPU ha sido también utilizada para acelerar el cómputo de un descriptor 3D. Este descriptor basado en la geometría local del espacio de entrada, ha sido implementado sobre la GPU, obteniendo tiempos de procesamiento muy bajos, que podrían ser considerados “tiempo real”. Además, el proceso completo para la extracción de los descriptores ha sido implementado sobre la GPU, acelerando también otros algoritmos necesarios para la extracción del descriptor 3D propuesto. Estos otros algoritmos son también comunes a la mayoría de descriptores 3D. Entre estos pasos encontramos la

adquisición de datos desde el dispositivo 3D a memoria GPU, la estimación de normales y la triangulación de las nubes de puntos.

Finalmente, el proceso para la extracción y emparejamiento de descriptores 3D sobre escenas reales ha sido acelerado obteniendo tiempos de ejecución por debajo de 1 segundo. De hecho, el prototipo implementado ha sido aplicado a un caso de estudio para reconocimiento de objetos en escenas 3D, permitiendo extraer 200 descriptores en aproximadamente 0.5 segundos y por tanto permitiendo reconocer los objetos presentes en la escena en tiempo real.





Universitat d'Alacant  
Universidad de Alicante

# Bibliography

---

- [Algorri and Schmitt, 1996] Algorri, M.-E. and Schmitt, F. (1996). Surface Reconstruction from Unstructured 3D Data. *Computer Graphics Forum*, 15(1):47–60. 14
- [Alonso-Montes and González Penedo, 2004] Alonso-Montes, C. and González Penedo, M. (2004). 3d object surface reconstruction using growing self-organised networks. In Sanfeliu, A., Martínez Trinidad, J. F., and Carrasco Ochoa, J. A., editors, *Progress in Pattern Recognition, Image Analysis and Applications*, volume 3287 of *Lecture Notes in Computer Science*, pages 163–170. Springer Berlin Heidelberg. 217
- [Amdahl, 1967] Amdahl, G. M. (1967). Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, AFIPS '67 (Spring), pages 483–485, New York, NY, USA. ACM. 110
- [Amenta et al., 2001] Amenta, N., Choi, S., and Kolluri, R. K. (2001). The power crust. In *Proceedings of the sixth ACM symposium on Solid modeling and applications*, SMA '01, pages 249–266, New York, NY, USA. ACM. 15, 27
- [Angelopoulou et al., 2013] Angelopoulou, A., García-Rodríguez, J., Psarrou, A., Mentzelopoulos, M., Reddy, B., Orts-Escalano, S., Serra, J. A., and Lewis, A. (2013). Natural user interfaces in volume visualisation using microsoft Kinect. In *ICIP Workshops*, pages 11–19. 26
- [Angelopoulou et al., 2005] Angelopoulou, A., Psarrou, A., García-Rodríguez, J., and Revett, K. (2005). Automatic landmarking of 2D

medical shapes using the growing neural gas network. In Liu, Y., Jiang, T., and Zhang, C., editors, *Computer Vision for Biomedical Image Applications*, volume 3765 of *Lecture Notes in Computer Science*, pages 210–219. Springer. 19

[Angelopoulou et al., 2007] Angelopoulou, A., Psarrou, A., and García-Rodriguez, J. (2007). Robust modelling and tracking of nonrigid objects using active-gng. In *Proc. IEEE 11th Int. Conf. Computer Vision ICCV 2007*, pages 1–7. 26

[Baader and Hirzinger, 1993] Baader, A. and Hirzinger, G. (1993). Three-dimensional surface reconstruction based on a self-organizing feature map. In *ICAR '93 Sixth Intern. Conf. on Advanced Robotics Tokyo, Japan, Nov. 1-2*, pages 480–487. 17

[Baader and Hirzinger, 1994] Baader, A. and Hirzinger, G. (1994). A self-organizing algorithm for multisensory surface reconstruction. In *IROS '94. Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems. Advanced Robotic Systems and the Real World*, volume 1, pages 81–8, New York, NY, USA. IEEE. 17

[Baena et al., 2013] Baena, R. M. L., López-Rubio, E., Domínguez, E., Palomo, E. J., and Jerez, J. M. (2013). A self-organizing map for traffic flow monitoring. In *IWANN (2)*, pages 458–466. 19, 26

[Barhak, 2002] Barhak, J. (2002). *Freeform objects with arbitrary topology from multirange images*. PhD thesis, Israel Institute of Technology, Haifa, Israel. 28, 68, 69, 70

[Bell and Hoberock, 2011] Bell, N. and Hoberock, J. (2011). *Thrust: A Productivity-Oriented Library for CUDA*, chapter ch. 26, pages pp. 359–371. Elsevier. 127

[Besl and McKay, 1992] Besl, P. and McKay, N. (1992). A method for registration of 3-d shapes. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 14(2):239–256. 7, 155, 157

## Bibliography

---

- [Bhattacharjee et al., 2011] Bhattacharjee, A., Contreras, G., and Martonosi, M. (2011). Parallelization libraries: Characterizing and reducing overheads. *ACM Trans. Archit. Code Optim.*, 8(1):5:1–5:29. 119
- [Bloomenthal, 1985] Bloomenthal, J. (1985). Modeling the mighty maple. *ACM SIGGRAPH Computer Graphics*, 19(3):305–311. 8
- [Blum et al., 2012] Blum, M., Springenberg, J. T., Wülfing, J., and Riedmiller, M. (2012). A learned feature descriptor for object recognition in RGB-D data. In *ICRA*, pages 1298–1303. 132
- [Boeres et al., 2004] Boeres, M. C., Ribeiro, C. C., and Bloch, I. (2004). A randomized heuristic for scene recognition by graph matching. In *WEA*, volume 3059, pages 100–113. Springer. 216
- [Boissonnat, 1984] Boissonnat, J.-D. (1984). Geometric structures for three-dimensional shape representation. *ACM Trans. Graph.*, 3(4):266–286. 15, 18
- [Brand and Kettnaker, 2000] Brand, M. and Kettnaker, V. (2000). Discovery and segmentation of activities in video. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8):844–851. 172
- [Buck et al., 2004] Buck, I., Foley, T., Horn, D., Sugerman, J., Fatahalian, K., Houston, M., and Hanrahan, P. (2004). Brook for GPUs: stream computing on graphics hardware. *ACM Trans. Graph.*, 23(3):777–786. 104
- [Buhmann, 2003] Buhmann, M. D. (2003). *Radial Basis Functions*. Cambridge University Press, New York, NY, USA. 30
- [Campbell and Flynn, 2001] Campbell, R. J. and Flynn, P. J. (2001). A survey of free-form object representation and recognition techniques. *Comput. Vis. Image Underst.*, 81(2):166–210. 12, 191
- [Cao and Suganthan, 2002] Cao, X. and Suganthan, P. N. (2002). Hierarchical overlapped growing neural gas networks with applications to video shot detection and motion characterization. In *Proc. Int. Joint Conf. Neural Networks IJCNN '02*, volume 2, pages 1069–1074. 26

- [Carr et al., 2001] Carr, J. C., Beatson, R. K., Cherrie, J. B., Mitchell, T. J., Fright, W. R., McCallum, B. C., and Evans, T. R. (2001). Reconstruction and representation of 3D objects with radial basis functions. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 67–76, New York, NY, USA. ACM. 16, 30
- [Chan et al., 2008] Chan, D., Buisman, H., Theobalt, C., and Thrun, S. (2008). A Noise-Aware Filter for Real-Time Depth Upsampling. In *Workshop on Multi-camera and Multi-modal Sensor Fusion Algorithms and Applications - M2SFA2 2008*, Marseille, France. Andrea Cavallaro and Hamid Aghajan. 138, 245
- [Che et al., 2008] Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J. W., and Skadron, K. (2008). A performance study of general-purpose applications on graphics processors using CUDA. *J. Parallel Distrib. Comput.*, 68(10):1370–1380. 217
- [Chen and Bhanu, 2007] Chen, H. and Bhanu, B. (2007). 3D free-form object recognition in range images using local surface patches. *Pattern Recognition Letters*, 28(10):1252 – 1262. 11
- [Cheng, 1997] Cheng, Y. (1997). Convergence and ordering of Kohonen’s batch map. *Neural Comput.*, 9(8):1667–1676. 108
- [Civera et al., 2010] Civera, J., Grasa, O. G., Davison, A. J., and Montiel, J. M. M. (2010). 1-point ransac for extended Kalman filtering: Application to real-time structure from motion and visual odometry. *J. Field Robot.*, 27(5):609–631. 93
- [Collins et al., 2000] Collins, R. T., Lipton, A. J., and Kanade, T. (2000). Introduction to the special section on video surveillance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):745–746. 172
- [Connolly, 1984] Connolly, C. I. (1984). Cumulative generation of octree models from range data. In *Proceedings of the First International Conference on Robotics and Automation*, page 25. IEEE. 15, 47

- [Cootes and Taylor, 2001] Cootes, T. F. and Taylor, C. J. (2001). Statistical models of appearance for medical image analysis and computer vision. In *Medical Imaging 2001*, pages 236–248. International Society for Optics and Photonics. 216
- [Cramer et al., 2012] Cramer, T., Schmidl, D., Klemm, M., and an Mey, D. (2012). OpenMP Programming on Intel Xeon Phi coprocessors: An early performance comparison. In *Proceedings of the Many-core Applications Research Community (MARC) Symposium at RWTH Aachen University*, pages 38–44. 204
- [Cretu et al., 2008] Cretu, A.-M., Petriu, E. M., and Payeur, P. (2008). Evaluation of growing neural gas networks for selective 3D scanning. In *Proc. Int. Workshop Robotic and Sensors Environments ROSE 2008*, pages 108–113. 26, 29, 217
- [Cretu et al., 2009] Cretu, A.-M., Petriu, E. M., and Payeur, P. (2009). Data acquisition and modeling of 3d deformable objects using neural networks. In *SMC*, pages 3383–3388. IEEE. 217
- [Cselenyi, 2005] Cselenyi, Z. (2005). Mapping the dimensionality, density and topology of data: the growing adaptive neural gas. *Computer methods and programs in biomedicine*, 78:141–56. 217
- [Delgado et al., 2009] Delgado, S., Gonzalo, C., Martinez, E., and Arquero, A. (2009). Topology preserving visualization methods for growing self-organizing maps. In *Bio-Inspired Systems: Computational and Ambient Intelligence*, volume 5517 of *Lecture Notes in Computer Science*, pages 196–203. Springer Berlin Heidelberg. 24
- [Dey and Goswami, 2003] Dey, T. K. and Goswami, S. (2003). Tight cocone: a water-tight surface reconstructor. In *Proceedings of the eighth ACM symposium on Solid modeling and applications*, SM '03, pages 127–134, New York, NY, USA. ACM. 15, 16
- [Dittenbach et al., 2001] Dittenbach, M., Merkl, D., and Rauber, A. (2001). Hierarchical clustering of document archives with the growing hierarchical self-organizing map. In Dorffner, G., Bischof, H., and Hornik,

- K., editors, *Artificial Neural Networks - ICANN 2001, International Conference Vienna, Austria, August 21-25, 2001 Proceedings*, volume 2130 of *Lecture Notes in Computer Science*, pages 500–508. Springer. 24
- [do Rego et al., 2007] do Rego, R. L. M., Araujo, A. F. R., and de Lima Neto, F. B. (2007). Growing self-organizing maps for surface reconstruction from unstructured point clouds. In *Proc. Int. Joint Conf. Neural Networks IJCNN 2007*, pages 1900–1905. 18, 26
- [Do Rego et al., 2010] Do Rego, R. L. M. E., Araujo, A. F. R., and De Lima Neto, F. B. (2010). Growing self-reconstruction maps. *Trans. Neur. Netw.*, 21(2):211–223. 29, 68, 69, 70
- [Doherty et al., 2005] Doherty, K. A. J., Adams, R. G., Davey, N., and Pensuwon, W. (2005). Hierarchical topological clustering learns stock market sectors. In *Proc. ICSC Congress Computational Intelligence Methods and Applications*. 217
- [Doucette et al., 2001] Doucette, P., Agouris, P., Stefanidis, A., and Musavi, M. (2001). Self-organised clustering for road extraction in classified imagery. *ISPRS Journal of Photogrammetry and Remote Sensing*, 55(5–6):347 – 358. 19
- [Edelsbrunner and Mücke, 1994] Edelsbrunner, H. and Mücke, E. P. (1994). Three-dimensional alpha shapes. *ACM Trans. Graph.*, 13(1):43–72. 15
- [Fanany and Kumazawa, 2004] Fanany, M. I. and Kumazawa, I. (2004). A neural network for recovering 3D shape from erroneous and few depth maps of shaded images. *Pattern Recogn. Lett.*, 25:377–389. 18
- [Fang et al., 2013a] Fang, J., Varbanescu, A. L., and Sips, H. (2013a). Identifying the key features of Intel Xeon Phi: A comparative approach. Technical report. 204
- [Fang et al., 2013b] Fang, J., Varbanescu, A. L., Sips, H., Zhang, L., Che, Y., and Xu, C. (2013b). Benchmarking Intel Xeon Phi to guide kernel design. Technical report. 204

## Bibliography

---

- [Fischler and Bolles, 1981] Fischler, M. A. and Bolles, R. C. (1981). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395. 93
- [Fisher, 2004] Fisher, R. B. (May 2004). Pets04 surveillance ground truth data set. In *Proc. Sixth IEEE Int. Work. on Performance Evaluation of Tracking and Surveillance (PETS04)*. 176
- [Fleishman et al., 2005] Fleishman, S., Cohen-Or, D., and Silva, C. T. (2005). Robust moving least-squares fitting with sharp features. *ACM Trans. Graph.*, 24(3):544–552. 16, 17
- [Flexer, 2001] Flexer, A. (2001). On the use of self-organizing maps for clustering and visualization. *Intell. Data Anal.*, 5(5):373–384. 24
- [Florez et al., 2002] Florez, F., García-Rodríguez, J. M., García-Chamizo, J., and Hernandez, A. (2002). Hand gesture recognition following the dynamics of a topology-preserving network. In *Proc. Fifth IEEE Int Automatic Face and Gesture Recognition Conf*, pages 318–323. 26
- [Florez-Revuelta, 2002] Florez-Revuelta, F. (2002). *Modelo de representación y procesamiento de movimiento para diseño de arquitecturas de tiempo real especializadas*. PhD thesis, University of Alicante, Alicante, Spain. 30, 46
- [Florez-Revuelta et al., 2002] Florez-Revuelta, F., García-Chamizo, J. M., García-Rodríguez, J., and Hernandez-Saez, A. (2002). Representation of 2D objects with a topology preserving network. In Quereda, J. M. I. and Micó, L., editors, *PRIS*, pages 267–276. ICEIS Press. 26, 216, 217
- [Flynn and Jain, 1991] Flynn, P. J. and Jain, A. K. (1991). CAD-based computer vision: From CAD models to relational graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(2):114–132. 191
- [Foley et al., 1990] Foley, J. D., van Dam, A., Feiner, S. K., and Hughes, J. F. (1990). *Computer graphics: principles and practice (2nd ed.)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. 145, 251

- [Freeman, 1990] Freeman, H. (1990). *Machine Vision for Three-Dimensional Scenes*. Academic Press. 191
- [Frezza-Buet, 2008] Frezza-Buet, H. (2008). Following non-stationary distributions by controlling the vector quantization accuracy of a growing neural gas network. *Neurocomput.*, 71:1191–1202. 26
- [Fritzke, 1993] Fritzke, B. (1993). Growing cell structures - a self-organizing network for unsupervised and supervised learning. *Neural Networks*, 7:1441–1460. 26, 28, 46
- [Fritzke, 1995] Fritzke, B. (1995). *A Growing Neural Gas Network Learns Topologies*, volume 7, pages 625–632. MIT Press. 21, 26, 29, 34, 217
- [Furao and Hasegawa, 2006] Furao, S. and Hasegawa, O. (2006). An incremental network for on-line unsupervised classification and topology learning. *Neural Netw.*, 19:90–106. 25
- [García-Rodríguez, 2009] García-Rodríguez, J. (2009). *Self-organizing neural network model to represent objects and their movement in realistic scenes*. PhD thesis, University of Alicante, Alicante, Spain. 19
- [García-Rodríguez et al., 2010] García-Rodríguez, J., Angelopoulou, A., García-Chamizo, J. M., and Psarrou, A. (2010). GNG based surveillance system. In *Proc. Int Neural Networks (IJCNN) Joint Conf*, pages 1–8. 19
- [García-Rodríguez et al., 2006a] García-Rodríguez, J., Angelopoulou, A., and Psarrou, A. (2006a). Growing neural gas (GNG): A soft competitive learning method for 2D hand modelling. *IEICE - Trans. Inf. Syst.*, E89-D:2124–2131. 26
- [García-Rodríguez et al., 2006b] García-Rodríguez, J., Angelopoulou, A., Psarrou, A., and Revett, K. (2006b). Automatically building 2d statistical shapes using the topology preservation model gng. volume 3851 of *Lecture Notes in Computer Science*, pages 519–528. Springer Berlin Heidelberg. 216

- [García-Rodríguez et al., 2007] García-Rodríguez, J., Florez-Revuelta, F., and García-Chamizo, J. M. (2007). Image compression using growing neural gas. In *Proc. Int. Joint Conf. Neural Networks IJCNN 2007*, pages 366–370. 26
- [García-Rodríguez and García-Chamizo, 2011] García-Rodríguez, J. and García-Chamizo, J. M. (2011). Surveillance and human-computer interaction applications of self-growing models. *Appl. Soft Comput.*, 11(7):4413–4431. 176, 217
- [García-Rodriguez et al., 2011] García-Rodriguez, J., Angelopoulou, A., García-Chamizo, J., Psarrou, A., Orts-Escalano, S., and Morell-Gimenez, V. (2011). Fast autonomous growing neural gas. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 725 –732. 122, 173, 239
- [Gokturk et al., 2004] Gokturk, S. B., Yalcin, H., and Bamji, C. (2004). A time-of-flight depth sensor - system description, issues and solutions. In *Proceedings of the 2004 Conference on Computer Vision and Pattern Recognition Workshop (CVPRW'04) Volume 3 - Volume 03*, CVPRW '04, pages 35–. IEEE Computer Society. 40
- [Gotardo et al., 2004] Gotardo, P., Boyer, K., Bellon, O., and Silva, L. (2004). Robust extraction of planar and quadric surfaces from range images. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 2, pages 216–219Vol.2. 4
- [Gregorski et al., 2000] Gregorski, B., Hamann, B., and Joy, K. (2000). Reconstruction of B-spline surfaces from scattered data points. *Proceedings Computer Graphics International 2000*, pages 163–170. 4
- [Griffin et al., 2012] Griffin, W., Wang, Y., Berrios, D., and Olano, M. (2012). Real-time GPU surface curvature estimation on deforming meshes and volumetric data sets. *IEEE Transactions on Visualization and Computer Graphics*, 18(10):1603–1613. 132
- [Grisetti et al., 2007] Grisetti, G., Grzonka, S., Stachniss, C., Pfaff, P., and Burgard, W. (2007). Efficient estimation of accurate maximum

- likelihood maps in 3D. In *In Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*. 156
- [Gschwandtner et al., 2011] Gschwandtner, M., Kwitt, R., Uhl, A., and Pree, W. (2011). BlenSor: Blender Sensor Simulation Toolbox Advances in Visual Computing. volume 6939 of *Lecture Notes in Computer Science*, chapter 20, pages 199–208. Springer Berlin / Heidelberg, Berlin, Heidelberg. 52, 223
- [Gustafson, 1988] Gustafson, J. L. (1988). Reevaluating amdahl’s law. *Communications of the ACM*, 31:532–533. 110
- [Hähnel et al., 2002] Hähnel, D., Burgard, W., and Thrun, S. (2002). Learning compact 3D models of indoor and outdoor environments with a mobile robot. *Robotics and Autonomous Systems*. to appear. 156
- [Hämäläinen, 2002] Hämäläinen, T. D. (2002). Self-organizing neural networks. chapter Parallel implementation of self-organizing maps, pages 245–278. Springer-Verlag New York, Inc., New York, NY, USA. 108
- [Harris, 2008] Harris, M. (2008). Optimizing parallel reduction in CUDA. *NVIDIA Dev. Technology*. 113, 162, 237
- [Hetzler et al., 2001] Hetzler, G., Leibe, B., Levi, P., and Schiele, B. (2001). 3D object recognition from range images using local feature histograms. In *CVPR (2)*, pages 394–399. IEEE Computer Society. 133
- [Hill and Marty, 2008] Hill, M. D. and Marty, M. R. (2008). Amdahl’s law in the multicore era. *IEEE COMPUTER*. 110
- [Himmelsbach et al., 2009] Himmelsbach, M., Luettel, T., and Wuensche, H.-J. (2009). Real-time object classification in 3D point clouds using point feature histograms. In *Proceedings of the 2009 IEEE/RSJ international conference on Intelligent robots and systems*, IROS’09, pages 994–1000, Piscataway, NJ, USA. IEEE Press. 132
- [Holdstein and Fischer, 2008] Holdstein, Y. and Fischer, A. (2008). Three-dimensional surface reconstruction using meshing growing neural gas (MGNG). *Vis. Comput.*, 24:295–302. 18, 26, 29, 68, 69, 217

## Bibliography

---

- [Holz and Behnke, 2012] Holz, D. and Behnke, S. (2012). Fast range image segmentation and smoothing using approximate surface reconstruction and region growing. In *Proceedings of the 12th International Conference on Intelligent Autonomous Systems (IAS)*, Jeju Island, Korea. 140, 141, 246, 248
- [Hoppe et al., 1992] Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., and Stuetzle, W. (1992). Surface reconstruction from unorganized points. *SIGGRAPH Comput. Graph.*, 26(2):71–78. 16, 27
- [Horn et al., 2007] Horn, D. R., Sugerman, J., Houston, M., and Hanrahan, P. (2007). Interactive k-d tree GPU raytracing. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, I3D ’07, pages 167–174. 105
- [Howarth and Buxton, 2000] Howarth, R. and Buxton, H. (2000). Conceptual descriptions from monitoring and watching image sequences. *Image and Vision Computing*, 18(2):105 – 135. 172
- [Howarth and Buxton, 1992] Howarth, R. J. and Buxton, H. (1992). Analogical representation of spatial events for understanding traffic behaviour. In *Proc. 10 th European Conference on AI*, pages 785–789. John Wiley. 172
- [Hu et al., 2004a] Hu, W., Tan, T., Wang, L., and Maybank, S. (2004a). A survey on visual surveillance of object motion and behaviors. *IEEE Transactions on Systems, Man and Cybernetics*, 34:334–352. 172
- [Hu et al., 2004b] Hu, W., Xie, D., and Tan, T. (2004b). A hierarchical self-organizing approach for learning the patterns of motion trajectories. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 15(1):135–144. 172
- [Hwu, 2011] Hwu, W.-m. W. (2011). *GPU Computing Gems Emerald Edition*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition. 105

- [Igarashi et al., 2011] Igarashi, J., Shouno, O., Fukai, T., and Tsujino, H. (2011). 2011 special issue: Real-time simulation of a spiking neural network model of the basal ganglia circuitry using general purpose computing on graphics processing units. *Neural Netw.*, 24:950–960. 108
- [Intel, 2012] Intel (2012). *Intel Threading Building Blocks 4.0 Open Source*. <http://threadingbuildingblocks.org/>. Intel. 119
- [Ivrissimtzis et al., 2003] Ivrissimtzis, I., Jeong, W.-K., and Seidel, H.-P. (2003). Using growing cell structures for surface reconstruction. In *Shape Modeling International*, pages 78 – 86. 28
- [Izadi et al., 2011] Izadi, S., Newcombe, R. A., Kim, D., Hilliges, O., Molyneaux, D., Hodges, S., Kohli, P., Shotton, J., Davison, A. J., and Fitzgibbon, A. W. (2011). KinectFusion: real-time dynamic 3D surface reconstruction and interaction. In *SIGGRAPH Talks*, pages 23:1–23:1. 132, 216
- [Jain et al., 1995] Jain, R., Kasturi, R., and Schunck, B. G. (1995). *Machine vision*. McGraw-Hill, Inc., New York, NY, USA. 6
- [Jang et al., 2008] Jang, H., Park, A., and Jung, K. (2008). Neural network implementation using CUDA and openmp. In *Proc. DICTA '08.Digital Image Computing: Techniques and Applications*, pages 155–161. 108
- [Jenkinson, 2003] Jenkinson, M. (2003). Measuring transformation error by RMS derivation. Technical report, Oxford Centre for Functional Magnetic Resonance Imaging of the Brain (FMRIB), Departament of Clinical Neurology, University of Oxford, John Radcliffe Hospital, Headley Way, Headington, Oxford, UK. 96
- [Jimeno-Morenilla et al., 2011] Jimeno-Morenilla, A., López, V., Espí, R., and Cuenca, S. (2011). A morphological-based method for inverse offset generation. An application for surface reconstruction using mechanical digitizers. *International Journal of Advanced Manufacturing Technology*, 54:1067–1076. 178

## Bibliography

---

- [Johnson and Hebert, 1999] Johnson, A. E. and Hebert, M. (1999). Using spin images for efficient object recognition in cluttered 3D scenes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 21(5):433–449. 11, 133, 165
- [Jolliffe, 1986] Jolliffe, I. (1986). *Principal Component Analysis*. Springer Verlag. 49
- [Juang et al., 2011] Juang, C.-F., Chen, T.-C., and Cheng, W.-Y. (2011). Speedup of implementing fuzzy neural networks with high-dimensional inputs through parallel processing on graphic processing units. *IEEE T. Fuzzy Systems*, 19(4):717–728. 108
- [Junior et al., 2004] Junior, A., Neto, A. D. D., and de Melo, J. (2004). Surface reconstruction using neural networks and adaptive geometry meshes. In *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*, volume 1, pages –807. 28
- [Kass et al., 1987] Kass, M., Witkin, A., and Terzopoulos, D. (1987). Snakes: Active contour models. *Int. Journal of Computer Vision*, 1. 216
- [Katz et al., 2010] Katz, R., Nieto, J., and Nebot, E. (2010). Unsupervised classification of dynamic obstacles in urban environments. *J. Field Robot.*, 27:450–472. 156
- [Kazhdan et al., 2006] Kazhdan, M., Bolitho, M., and Hoppe, H. (2006). Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, SGP ’06, pages 61–70, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association. 16
- [Kerl et al., 2013] Kerl, C., Sturm, J., and Cremers, D. (2013). Dense visual slam for rgb-d cameras. In *Proc. of the Int. Conf. on Intelligent Robot Systems (IROS)*. 216
- [Khan and Cheng, 2011] Khan, W.A., A.-R. A. and Cheng, K. (2011). *Virtual Manufacturing*. Springer-Verlag, London. 191

- [Khoshelham and Elberink, 2012] Khoshelham, K. and Elberink, S. O. (2012). Accuracy and resolution of Kinect depth data for indoor mapping applications. *Sensors*, 12(2):1437–1454. 41, 82, 216
- [Kirk and Hwu, 2010] Kirk, D. B. and Hwu, W.-m. W. (2010). *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition. 106, 217
- [Kitson, 1989] Kitson, F. (1989). An algorithm for curve and surface fitting using B-splines. In *Acoustics, Speech, and Signal Processing, 1989. ICASSP-89., 1989 International Conference on*, pages 1207–1210 vol.2. 7
- [Kobbelt and Botsch, 2004] Kobbelt, L. and Botsch, M. (2004). A survey of point-based techniques in computer graphics. *Computers & Graphics*, 28:801–814. 15, 47, 81
- [Kohonen, 1982] Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43(1):59–69. 24, 25, 26
- [Kohonen, 1995] Kohonen, T. (1995). *Self-Organising Maps*. Springer-Verlag. 17, 46, 217
- [Konolige and Mihelich, 2013] Konolige, K. and Mihelich, P. (2013). Technical description of Kinect calibration. 243
- [Kramer et al., 2012] Kramer, J., Burrus, N., Herrera, D., Echtler, F., and Parker, M. (2012). *Hacking the Kinect*. Apress, Berkely, CA, USA, 1st edition. 192
- [Lee et al., 2011] Lee, W., Park, N., and Woo, W. (2011). Depth-assisted real-time 3D object detection for augmented reality. *ICAT'11*, 2:126–132. 132
- [Lejdfors and Ohlsson, 2006] Lejdfors, C. and Ohlsson, L. (2006). Implementing an embedded GPU language by combining translation and gen-

- eration. In *Proceedings of the 2006 ACM symposium on Applied computing*, SAC '06, pages 1610–1614, New York, NY, USA. ACM. 104
- [Leonardis et al., 1997] Leonardis, A., Jaklič, A., and Solina, F. (1997). Superquadrics for segmenting and modeling range data. *IEEE Trans. Pattern Anal. Mach. Intell.*, 19(11):1289–1295. 9
- [Liu et al., 2006] Liu, H.-b., Wang, X., Wu, X.-j., and Qiang, W.-y. (2006). Surface reconstruction based on radial basis functions network. In *Proceedings of the Third international conference on Advances in Neural Networks - Volume Part III*, ISNN'06, pages 1242–1247, Berlin, Heidelberg. Springer-Verlag. 30
- [Lorensen and Cline, 1987] Lorensen, W. E. and Cline, H. E. (1987). Marching cubes: A high resolution 3D surface construction algorithm. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, volume 21, pages 163–169, New York, NY, USA. ACM Press. 16
- [Lowe, 2004] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–110. 82
- [Luebke et al., 2004] Luebke, D., Harris, M., Krüger, J., Purcell, T., Govindaraju, N., Buck, I., Woolley, C., and Lefohn, A. (2004). GPGPU: general purpose computation on graphics hardware. In *Annual Conference on Computer Graphics*. 104
- [Mann and Haykin, 1990] Mann, R. and Haykin, S. (1990). A parallel implementation of Kohonen’s feature maps on the warp systolic computer. In *Proc. IJCNN-90-WASH-DC, Int. Joint Conf. on Neural Networks*, volume II, pages 84–87, Hillsdale, NJ. Lawrence Erlbaum. 108
- [Mark et al., 2003] Mark, W. R., Glanville, R. S., Akeley, K., and Kilgard, M. J. (2003). Cg: a system for programming graphics hardware in a C-like language. *ACM Transactions on Graphics*, 22:896–907. 104

- [Martinetz and Schulten, 1994] Martinetz, T. and Schulten, K. (1994). Topology representing networks. *Neural Networks*, 7(3). 25, 26, 28, 45, 46, 220
- [Martinetz et al., 1993] Martinetz, T. M., Berkovich, S. G., and Schulten, K. J. (1993). ‘neural-gas’ network for vector quantization and its application to time-series prediction. 4(4):558–569. 26, 33, 34
- [Mathew and Joy, 2010] Mathew, S. and Joy, P. (2010). Ultra fast SOM using CUDA. Technical report, NeST-NVIDIA Center for GPU Computing. 108
- [May et al., 2009] May, S., Droeuschel, D., Holz, D., Fuchs, S., Malis, E., Nüchter, A., and Hertzberg, J. (2009). Three-dimensional mapping with time-of-flight cameras. *J. Field Robot.*, 26:934–965. 156
- [Mederos et al., 2005] Mederos, B., Amenta, N., Velho, L., and de Figueiredo, L. H. (2005). Surface reconstruction from noisy point clouds. In *Proceedings of the third Eurographics symposium on Geometry processing*, SGP ’05, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association. 16
- [Mencl and Müller, 1997] Mencl, R. and Müller, H. (1997). Interpolation and approximation of surfaces from three-dimensional scattered data points. In *Proceedings of the Conference on Scientific Visualization*, DAGSTUHL ’97, pages 223–, Washington, DC, USA. IEEE Computer Society. 14
- [Merrill and Grimshaw, 2010] Merrill, D. G. and Grimshaw, A. S. (2010). Revisiting sorting for GPGPU stream architectures. In *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, PACT ’10, pages 545–546, New York, NY, USA. ACM. 127
- [Mian et al., 2006a] Mian, A. S., Bennamoun, M., and Owens, R. (2006a). Three-dimensional model-based object recognition and segmentation in cluttered scenes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(10):1584–1601. 11, 133, 145, 159, 160, 162, 168, 250

## Bibliography

---

- [Mian et al., 2006b] Mian, A. S., Bennamoun, M., and Owens, R. A. (2006b). A novel representation and feature matching algorithm for automatic pairwise registration of range images. *Int. J. Comput. Vision*, 66(1):19–40. 133
- [Mikolajczyk and Schmid, 2002] Mikolajczyk, K. and Schmid, C. (2002). An affine invariant interest point detector. In Heyden, A., Sparr, G., Nielsen, M., and Johansen, P., editors, *Computer Vision — ECCV 2002*, volume 2350 of *Lecture Notes in Computer Science*, pages 128–142. Springer Berlin Heidelberg. 83
- [Miller et al., 1991] Miller, J. V., Breen, D. E., Lorensen, W. E., O’Bara, R. M., and Wozny, M. J. (1991). Geometrically deformed models: a method for extracting closed geometric models from volume data. *SIGGRAPH Comput. Graph.*, 25(4):217–226. 17
- [Mole and Araújo, 2010] Mole, V. L. D. and Araújo, A. F. R. (2010). Growing self-organizing surface map: Learning a surface topology from a point cloud. *Neural Comput.*, 22(3):689–729. 70
- [Moraes et al., 2012] Moraes, F. C., Botelho, S. C., Filho, N. D., and Gaya, J. F. O. (2012). Parallel high dimensional self organizing maps using CUDA. *Brazilian Robotics Symposium and Latin American Robotics Symposium (SBR-LARS)*, 0:302–306. 108
- [Muñoz-Salinas et al., 2008] Muñoz-Salinas, R., Aguirre, E., García-Silvente, M., and Gonzalez, A. (2008). A multiple object tracking approach that combines colour and depth information using a confidence measure. *Pattern Recognition Letters*, 29(10):1504 – 1514. 156
- [Muraki, 1991] Muraki, S. (1991). Volumetric shape description of range data using blobby model. *SIGGRAPH Comput. Graph.*, 25(4):227–235. 17
- [Nageswaran et al., 2009] Nageswaran, J. M., Dutt, N., Krichmar, J. L., Nicolau, A., and Veidenbaum, A. (2009). Efficient simulation of large-scale spiking neural networks using CUDA graphics processors. In *Proc. Int. Joint Conf. Neural Networks IJCNN 2009*, pages 2145–2152. 108

- [Nasse et al., 2009] Nasse, F., Thurau, C., and Fink, G. A. (2009). Face detection using GPU-based convolutional neural networks. In *Proceedings of the 13th International Conference on Computer Analysis of Images and Patterns*, CAIP '09, pages 83–90, Berlin, Heidelberg. Springer-Verlag. 108
- [Nickolls and Dally, 2010] Nickolls, J. and Dally, W. J. (2010). The GPU computing era. *IEEE Micro*, 30:56–69. 105
- [Noble, 1988] Noble, J. (1988). Finding corners. *Image and Vision Computing*, 6:121–128. 84
- [Nordström, 1991] Nordström, T. (1991). *Designing Parallel Computers for Self Organizing Maps*. Forskningsrapport / Tekniska högskolan i Luleå. 108
- [Nuchter et al., 2004] Nuchter, A., Surmann, H., Lingemann, K., Hertzberg, J., and Thrun, S. (2004). 6d slam with an application in autonomous mine mapping. In *In Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1998–2003. 81
- [NVIDIA, 2013] NVIDIA (2013). *NVIDIA CUDA Programming Guide 5.0*. 104, 105
- [NVIDIA, 2013] NVIDIA (2013). Visual profiler 5.0. <https://developer.nvidia.com/nvidia-visual-profiler>. 148
- [Oh, 2004] Oh, K. (2004). GPU implementation of neural networks. *Pattern Recognition*, 37(6):1311–1314. 108
- [Oh and Jung, 1999] Oh, S. and Jung, K. (1999). View-point insensitive human pose recognition using neural network and CUDA. *World Academy of Science*, 60. 108
- [Oikonomidis et al., 2011] Oikonomidis, I., Kyriazis, N., and Argyros, A. A. (2011). Full dof tracking of a hand interacting with an object by modeling occlusions and physical constraints. In *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, pages 2088–2095, Washington, DC, USA. IEEE Computer Society. 216

- [Olesen et al., 2012] Olesen, S., Lyder, S., Kraft, D., Krüger, N., and Jessen, J. (2012). Real-time extraction of surface patches with associated uncertainties by means of Kinect cameras. *Journal of Real-Time Image Processing*, pages 1–14. 132
- [O'Rourke, 1998] O'Rourke, J. (1998). *Computational Geometry in C*. Cambridge University Press, New York, NY, USA, 2nd edition. 15
- [Palomo et al., 2013] Palomo, E. J., Domínguez, E., Luque-Baena, R. M., and Muñoz, J. (2013). Image compression and video segmentation using hierarchical self-organization. *Neural Processing Letters*, 37(1):69–87. 26
- [Pentland, 1986] Pentland, A. P. (1986). Perceptual organization and the representation of natural form. *Artif. Intell.*, 28(3):293–331. 9
- [Pharr and Fernando, 2005] Pharr, M. and Fernando, R. (2005). *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (GPU Gems)*. Addison-Wesley Professional. 10
- [Platos and Gajdos, 2010] Platos, J. and Gajdos, P. (2010). Large data real-time classification with non-negative matrix factorization and self-organizing maps on GPU. In *Computer Information Systems and Industrial Management Applications (CISIM), 2010 International Conference on*, pages 176–181. 108
- [Pomerleau et al., 2012] Pomerleau, F., Breitenmoser, A., Liu, M., Colas, F., and Siegwart, R. (2012). Noise characterization of depth sensors for surface inspections. In *2012 2nd International Conference on Applied Robotics for the Power Industry (CARPI)*, pages 16–21. IEEE. 216
- [Prabhu, 2008] Prabhu, R. (2008). SOMGPU: an unsupervised pattern classifier on graphical processing unit. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, page 1011–1018. IEEE. 108
- [Prisacariu and Reid, 2009] Prisacariu, V. and Reid, I. (2009). fastHoG - a real-time GPU implementation of hog. Technical Report 2310/09, Department of Engineering Science, Oxford University. 132

- [Qin and Suganthan, 2004] Qin, A. K. and Suganthan, P. N. (2004). Robust growing neural gas algorithm with application in cluster analysis. *Neural Netw.*, 17:1135–1148. 217
- [Raja and Jain, 1992] Raja, N. S. and Jain, A. K. (1992). Recognizing geons from superquadrics fitted to range data. *Image Vision Comput.*, 10(3):179–190. 9
- [Rivera-Rovelo et al., 2006] Rivera-Rovelo, J., Herold, S., and Bayro-Corrochano, E. (2006). Object segmentation using growing neural gas and generalized gradient vector flow in the geometric algebra framework. In *Progress in Pattern Recognition, Image Analysis and Applications*, volume 4225 of *Lecture Notes in Computer Science*, pages 306–315. Springer Berlin / Heidelberg. 10.1007/11892755-31. 26
- [Roberts, 1965] Roberts, L. (1965). Machine perception of three-dimensional solids. In *Optical and Electro-Optical Information Processing*. MIT Press, Cambridge, MA. 5
- [Roth and Wibowoo, 1997] Roth, G. and Wibowoo, E. (1997). An Efficient Volumetric Method for Building Closed Triangular Meshes from 3-D Image and Point Data. In *IN GRAPHICS INTERFACE 97*, pages 173–180. 16
- [Ruprecht et al., 1995] Ruprecht, D., Nagel, R., and Müller, H. (1995). Spatial free-form deformation with scattered data interpolation methods. *Computers and Graphics*, 19(1):63–71. 17
- [Rusu et al., 2009] Rusu, R. B., Blodow, N., and Beetz, M. (2009). Fast point feature histograms (FPFH) for 3D registration. In *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pages 3212 –3217. 91
- [Rusu et al., 2008a] Rusu, R. B., Blodow, N., Marton, Z. C., and Beetz, M. (2008a). Aligning Point Cloud Views using Persistent Feature Histograms. In *Proceedings of the 21st IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Nice, France, September 22-26*. 88, 91

## Bibliography

---

- [Rusu and Cousins, 2011] Rusu, R. B. and Cousins, S. (2011). 3D is here: Point Cloud Library (PCL). In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China. 48, 223
- [Rusu et al., 2008b] Rusu, R. B., Marton, Z. C., Blodow, N., and Beetz, M. (2008b). Learning informative point classes for the acquisition of object model maps. pages 643–650. 156
- [Satish et al., 2009] Satish, N., Harris, M., and Garland, M. (2009). Designing efficient sorting algorithms for manycore GPUs. In *Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*, IPDPS '09, pages 1–10. 105, 127
- [Schnabel and Klein, 2006] Schnabel, R. and Klein, R. (2006). Octree-based point-cloud compression. In Botsch, M. and Chen, B., editors, *Symposium on Point-Based Graphics 2006*. Eurographics. 10
- [Schreiber and Rauter, 2009] Schreiber, D. and Rauter, M. (2009). GPU-based non-parametric background subtraction for a practical surveillance system. In *In proceedings of ICCV Workshops*. 174
- [Söderkvist, 1999] Söderkvist, I. (1999). Introductory overview of surface reconstruction methods. Research report / Department of Mathematics, Luleå University of Technology. 12
- [Shen et al., 2004] Shen, C., O'Brien, J. F., and Shewchuk, J. R. (2004). Interpolating and approximating implicit surfaces from polygon soup. *ACM Trans. Graph.*, 23(3):896–904. 16
- [Shi and Tomasi, 1994] Shi, J. and Tomasi, C. (1994). Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, pages 593–600. IEEE. 84
- [Sipiran and Bustos, 2011] Sipiran, I. and Bustos, B. (2011). Harris 3D: a robust extension of the harris operator for interest point detection on 3D meshes. *Vis. Comput.*, 27(11):963–976. 83

- [Skrypnyk and Lowe, 2004] Skrypnyk, I. and Lowe, D. G. (2004). Scene modelling, recognition and tracking with invariant image features. In *ISMAR*, pages 110–119. IEEE Computer Society. 215
- [Sledge and Keller, 2008] Sledge, I. J. and Keller, J. M. (2008). Growing neural gas for temporal clustering. In *Proc. 19th Int. Conf. Pattern Recognition ICPR 2008*, pages 1–4. 217
- [Stanford Scanning Repository, 2013] Stanford Scanning Repository (2013). The stanford 3D scanning repository. <http://graphics.stanford.edu/data/3Dscanrep/>. 42
- [Steder et al., 2010] Steder, B., Grisetti, G., and Burgard, W. (2010). Robust place recognition for 3D range data based on point features. In *ICRA*, pages 1400–1405. IEEE. 156
- [Steder et al., 2009] Steder, B., Grisetti, G., Loock, M. V., and Burgard, W. (2009). Robust on-line model-based object detection from range images. In *IROS*, pages 4739–4744. IEEE. 156
- [Stergiopoulou and Papamarkos, 2006] Stergiopoulou, E. and Papamarkos, N. (2006). A new technique for hand gesture recognition. In *Image Processing, 2006 IEEE International Conference on*, pages 2657–2660. 19
- [Stockman and Shapiro, 2001] Stockman, G. and Shapiro, L. G. (2001). *Computer Vision*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition. 12
- [Stone et al., 2010] Stone, J. E., Gohara, D., and Shi, G. (2010). OpenCL: A parallel programming standard for heterogeneous computing systems. *IEEE Des. Test*, 12(3):66–73. 104
- [Sun and Gustafson, 1993] Sun, X.-H. and Gustafson, J. L. (1993). Computer benchmarks. chapter Toward a better parallel performance metric, pages 23–39. Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands. 110

## Bibliography

---

- [Szummer and Picard, 1998] Szummer, M. and Picard, R. (1998). Indoor-outdoor image classification. In *Content-Based Access of Image and Video Database, 1998. Proceedings., 1998 IEEE International Workshop*, pages 42 –51. 80
- [Tamimi et al., 2006] Tamimi, H., Andreasson, H., Treptow, A., Duckett, T., and Zell, A. (2006). Localization of mobile with omnidirectional vision using particle filter and iterative sift. *Robotics and Autonomous Systems*, 54(9):758 – 765. 80
- [Temizel et al., 2011] Temizel, A., Halıcı, T., Loğoglu, B., Temizel, T. T., Ömrüuzun, F., and Karaman, E. (2011). Experiences on image and video processing with CUDA and OpenCL. *GPU Computing Gems 1, NVIDIA (Elsevier)*. 174
- [Tian et al., 2002] Tian, Y., Tan, T., and Sun, H. (2002). A novel robust algorithm for real-time object tracking. *Chinese J. Automation*, pages 851–853. 172
- [Tomasi and Manduchi, 1998] Tomasi, C. and Manduchi, R. (1998). Bilateral filtering for gray and color images. In *Proceedings of the Sixth International Conference on Computer Vision, ICCV '98*, pages 839–846, Washington, DC, USA. IEEE Computer Society. 81, 137, 244
- [Tombari and Salti, 2011] Tombari, F. and Salti (2011). A combined texture-shape descriptor for enhanced 3D feature matching. In *Image Processing (ICIP), 2011 18th IEEE International Conference on*, pages 809 –812. 92
- [Tombari et al., 2010a] Tombari, F., Salti, S., and Di Stefano, L. (2010a). Unique signatures of histograms for local surface description. In *Proceedings of the 11th European Conference on Computer Vision Conference on Computer Vision: Part III, ECCV'10*, pages 356–369, Berlin, Heidelberg. Springer-Verlag. xxiv, xxxiii, 42, 44, 231
- [Tombari et al., 2010b] Tombari, F., Salti, S., and Di Stefano, L. (2010b). Unique signatures of histograms for local surface description. In *Proceedings of the 11th European conference on computer vision conference*

- [on Computer vision: Part III, ECCV'10, pages 356–369, Berlin, Heidelberg. Springer-Verlag. 91
- [Treisman and Gelade, 1980] Treisman, A. M. and Gelade, G. (1980). A feature-integration theory of attention. *Cognitive Psychology*, 12(1):97–136. 80
- [Uetz and Behnke, 2009] Uetz, R. and Behnke, S. (2009). Large-scale object recognition with cuda-accelerated hierarchical neural networks. In *In Proceedings of the 1st IEEE International Conference on Intelligent Computing and Intelligent Systems*. 108
- [Velastin and Remagnino, 2006] Velastin, S. A. and Remagnino, P. (2006). *Intelligent Distributed Video Surveillance Systems*. IET Digital Library. 172
- [Vesanto and Alhoniemi, 2000] Vesanto, J. and Alhoniemi, E. (2000). Clustering of the self-organizing map. *IEEE Transactions on Neural Networks*, 11(3):586–600. 24
- [Viejo and Cazorla, 2013] Viejo, D. and Cazorla, M. (2013). A robust and fast method for 6DoF motion estimation from generalized 3D data. *Autonomous Robots*, pages 1–14. 155, 156, 158
- [Viejo et al., 2012a] Viejo, D., García-Rodríguez, J., and Cazorla, M. (2012a). A study of a soft computing based method for 3D scenario reconstruction. *Appl. Soft Comput.*, 12(10):3158–3164. 26
- [Viejo et al., 2012b] Viejo, D., García-Rodríguez, J., Cazorla, M., Mendez, D. G., and Johnsson, M. (2012b). Using GNG to improve 3D feature extraction - application to 6DoF egomotion. *Neural Networks*, 32:138–146. 26
- [Villaverde and Graña, 2009] Villaverde, I. and Graña, M. (2009). An improved evolutionary approach for egomotion estimation with a 3D tof camera. In *Bioinspired Applications in Artificial and Natural Computation*, volume 5602 of *Lecture Notes in Computer Science*, pages 390–398. Springer Berlin / Heidelberg. 156

- [Vinayak et al., 2013] Vinayak, Murugappan, S., Liu, H., and Ramani, K. (2013). Shape-it-up: Hand gesture based creative expression of 3D shapes using intelligent generalized cylinders. *Comput. Aided Des.*, 45(2):277–287. 8, 9
- [Vojacek and Dvorsky, 2013] Vojacek, L. and Dvorsky, J. (2013). Growing neural gas – a parallel approach. In *Computer Information Systems and Industrial Management*, volume 8104 of *Lecture Notes in Computer Science*, pages 408–419. Springer Berlin Heidelberg. 109
- [Wagner et al., 2013] Wagner, R., Frese, U., and Bäuml, B. (2013). 3d modeling, distance and gradient computation for motion planning: A direct gpgpu approach. In *ICRA*, pages 3586–3592. 217
- [Walder et al., 2006] Walder, C., Schölkopf, B., and Chapelle, O. (2006). Implicit surface modelling with a globally regularised basis of compact support. *Comput. Graph. Forum*, 25(3):635–644. 16
- [Wang, 2006] Wang, C. C. L. (2006). Incremental reconstruction of sharp edges on mesh surfaces. *Comput. Aided Des.*, 38(6):689–702. 17
- [Wang et al., 2006] Wang, G., Wong, T.-T., and Heng, P.-A. (2006). GPU-friendly warped display for scope-maintained video surveillance. *Multi-media Syst.*, 12(3):169–178. 174
- [Wang et al., 2011] Wang, L., Huang, M., and El-Ghazawi, T. A. (2011). Exploiting concurrent kernel execution on graphic processing units. In Smari, W. W. and McIntire, J. P., editors, *HPCS*, pages 24–32. IEEE. 148
- [Wang et al., 2000] Wang, Y., Peterson, B. S., and Staib, L. H. (2000). Shape-based 3D surface correspondence using geodesics and local geometry. In *CVPR*, pages 2644–2651. IEEE Computer Society. 11
- [Wasza et al., 2011] Wasza, J., Bauer, S., and Hornegger, J. (2011). Real-time preprocessing for dense 3-d range imaging on the GPU: Defect interpolation, bilateral temporal averaging and guided filtering. In *ICCV Workshops*, pages 1221–1227. 81, 138, 245

- [Weingarten et al., 2004] Weingarten, J. W., Gruener, G., and Siegwart, R. (2004). Probabilistic plane fitting in 3D and an application to robotic mapping. In *ICRA*, pages 927–932. IEEE. 156
- [Wiskott et al., 1997] Wiskott, L., Fellous, J.-M., Krüger, N., and Malsburg, C. V. D. (1997). Face recognition by elastic bunch graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19:775–779. 216
- [Wu et al., 1991] Wu, C.-H. J., Hodges, R. E., and Wang, C.-J. (1991). Parallelizing the self-organizing feature map on multiprocessor systems. *Parallel Computing*, 17(6-7):821–832. 108
- [Wu et al., 2000] Wu, Y., Liu, Q., and Huang, T. S. (2000). An adaptive self-organizing color segmentation algorithm with application to robust real-time human hand localization. In *in Proc. of Asian Conference on Computer Vision*, pages 1106–1111. 26, 172
- [Wu and Takatsuka, 2006] Wu, Y. and Takatsuka, M. (2006). Spherical self-organizing map using efficient indexed geodesic data structure. *Neural Networks*, 19(6-7):900–910. 108
- [Xiao et al., 2011] Xiao, Y., Leung, C. S., Ho, T.-Y., and Lam, P.-M. (2011). A GPU implementation for LBG and SOM training. *Neural Comput. Appl.*, 20(7):1035–1042. 108
- [Yu, 1999] Yu, Y. (1999). Surface reconstruction from unorganized points using self-organizing neural networks yizhou yu. In *In IEEE Visualization 99, Conference Proceedings*, pages 61–64. 27, 28
- [Zhang, 2012] Zhang, Z. (2012). Microsoft Kinect sensor and its effect. *MultiMedia*, IEEE, 19(2):4 –10. 137, 244
- [Zhong, 2009] Zhong, Y. (2009). Intrinsic shape signatures: A shape descriptor for 3D object recognition. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, pages 689 –696. 87

# List of Acronyms

---

**6DoF** 6 Degrees Of Freedom

**ALU** Arithmetic Logic Unit

**ANN** Artificial Neural Network

**API** Application Programming Interface

**CAD** Computer-Aided Design

**CAM** Computer-Aided Manufacturing

**CAVIAR** Context Aware Vision using Image-based Active Recognition

**CHL** Competitive Hebbian Learning

**CI** Computational Intelligence

**CMOS** Complementary Metal–Oxide–Semiconductor

**CNC** Computer Numerical Control

**CPU** Central Processing Unit

**CSHOT** Color Signature of Histograms of Orientations

**CUDA** Compute Unified Device Architecture

**DC** Distributed Computing

**ECHL** Extended Competitive Hebbian Learning

**EVD** Eigen Value Decomposition

---

**FPFH** Fast Point Feature Histogram

**FPGA** Field Programmable Gate Array

**GC** Generalized Cylinder

**GCS** Growing Cell Structures

**GNG** Growing Neural Gas

**GPU** Graphics Processing Unit

**GPGPU** General-Purpose computing on Graphics Processing Units

**GSRM** Growing Self-Reconstruction Maps

**GSOM** Growing Self-organising Map

**HLSL** High Level Shader Language

**HPC** High Performance Computing

**ICP** Iterative Closest Point

**IEEE** Institute of Electrical and Electronics Engineers

**ILP** Instruction Level Parallelism

**IR** Infrared

**ISS** Intrinsic Shape Signature

**LVQ** Linear Vector Quantization

**LSD** Local Shape Descriptor

**MSE** Mean Square Error

**MGNG** Modified Growing Neural Gas

**NG** Neural Gas

**NURBS** Non-Uniform Rational B-Spline

**OpenCL** Open Computing Language

**OpenGL** Open Graphics Language

**PCA** Principal Component Analysis

**PCI** Peripheral Component Interconnect

**PCL** PointCloud Library

**PFH** Point Feature Histogram

**PFHRGB** Point Feature Fast Histogram RGB

**RAM** Random-Access Memory

**RANSAC** RANdom Sample Consensus

**RBF** Radial Basis Function

**RGB** Red Green Blue

**RGB-D** RGB-Depth

**RMSE** Root-Mean-Squared Error

**SHOT** Signature of Histograms of Orientations

**SIFT** Scale Invariant Feature Transform

**SIMD** Single Instruction, Multiple Data

**SIMT** Single Instruction, Multiple Threads

**SL** Supervised Learning

**SLAM** Simultaneous Localization And Mapping

**SM** Streaming Multi-processor

**SOM** Self-Organising Maps

**SoC** System on Chip

**SRAM** Static Random Access Memory

**TBB** Threading Building Blocks

---

**TPN** Topology Preserving Networks

**TRN** Topology Representing Networks

**UL** Unsupervised Learning

**US** Uniform Sampling

**VG** Voxel Grid

**VLSI** Very Large Scale Integration

