

ADVERTIMENT. L'accés als continguts d'aquesta tesi queda condicionat a l'acceptació de les condicions d'ús estableties per la següent llicència Creative Commons:  http://cat.creativecommons.org/?page_id=184

ADVERTENCIA. El acceso a los contenidos de esta tesis queda condicionado a la aceptación de las condiciones de uso establecidas por la siguiente licencia Creative Commons:  <http://es.creativecommons.org/blog/licencias/>

WARNING. The access to the contents of this doctoral thesis it is limited to the acceptance of the use conditions set by the following Creative Commons license:  <https://creativecommons.org/licenses/?lang=en>

UNIVERSITAT AUTONOMA DE BARCELONA

DOCTORAL THESIS

Embedded 3D Reconstruction for Autonomous Driving

Author:

Daniel HERNÁNDEZ JUÁREZ

Supervisors:

Dr. Juan Carlos MOURE and

Dr. David VÁZQUEZ

*A thesis submitted in fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science*

in the

Department of Computer Architecture and Operating Systems
Escola d'Enginyeria

October 19, 2020

UNIVERSITAT AUTONOMA DE BARCELONA

Abstract

Department of Computer Architecture and Operating Systems
Escola d'Enginyeria

Doctor of Philosophy in Computer Science

Embedded 3D Reconstruction for Autonomous Driving

by Daniel HERNÁNDEZ JUÁREZ

The objective of this thesis is to study 3D reconstruction algorithms suitable for autonomous driving. In order to do so, we need fast implementations and representations of the 3D environment that take into account geometric and semantic information. The use of CUDA and GPU parallelization allows to leverage flexible and programmable high performance hardware to fulfill the strong time requirements.

The thesis presents three main contributions. First, we describe the parallelization of the well-known stereo matching algorithm based on Semi-Global Matching (SGM), which estimates depth from two stereo images. We deploy an efficient parallelization design that runs on top of low-energy consumption GPUs and achieves real-time performance.

As our second contribution, we present an improvement of the 3D representation model called the Stixel World that accounts for slanted surfaces. The extension of the model helps representing real scenes that fail under the previous assumptions, and, by an efficient model regularization, keeps the same accuracy of the previous model. We also propose an algorithmic strategy to speed up the process, which reduces the amount of Stixel combinations tested by the dynamic programming approach.

Finally, we explain our parallelization strategies for the Stixel segmentation algorithm. We propose a parallelization strategy that fits the massively parallel architecture of GPUs. We also study the different speed up techniques available for Stixels and how they can be implemented efficiently for this architecture. Additionally, our approach reduces the computational complexity of the algorithm by reformulating the measurement depth model, relying on the confidence of the depth estimation and the identification of invalid values to handle outliers.

Acknowledgements

I would like to express my gratitude to everyone who has participated in the development of this thesis. Especially, my supervisors, Dr. Juan Carlos Moure and Dr. David Vázquez. They guided me well throughout the research work. Their ideas, advice and knowledge were key to develop this thesis and its related publications.

Apart from my supervisors, I would also want to thank Dr. Antonio Espinosa and Dr. Antonio Manuel López, for their help during the PhD and their contributions to this work. They played a major role in polishing my research writing skills.

I also want to thank my colleagues in both CAOS and CVC. Special mention to Gabriel Villalonga and Alejandro Chacón for their very useful technical discussions and help. But also the rest of the PhD students, professors and staff for making the experience enjoyable.

Finally, I want to thank people that has helped me throughout the last years by proofreading, rehearsing my presentations and generally, being there to support me. Especially, Marina and my parents Lola and Pedro.

Contents

Abstract	iii
Acknowledgements	v
1 Introduction	1
1.1 Problem Description	1
1.2 Stereo vision	1
1.3 Stixel Model	2
1.4 GPU Parallelization	4
1.5 Objectives and motivation	4
1.6 Main Contributions	5
1.6.1 Parallelization of SGM for GPU	5
1.6.2 Slanted Stixels	5
1.6.3 Parallelization of Stixels for GPU	5
2 Semi-global Matching Parallelization	7
2.1 Introduction	7
2.2 Disparity Estimation Pipeline	8
2.2.1 Local Matching Cost and Semi-Global Matching (SGM)	8
2.2.2 Related work	9
2.3 Algorithm Description and Massive Parallelization	9
2.3.1 Matching Cost Computation	10
2.3.2 Smoothing Cost Aggregation (SGM) and Disparity Computation	11
2.3.3 Additional Optimizations	13
2.4 Results	14
2.5 Conclusions	15
3 Slanted Stixels	17
3.1 Introduction	17
3.2 Related Work	19
3.3 Stixel Model	20
3.3.1 Data term	22
New depth model	23
3.3.2 Prior term	23
Model complexity prior	24
Segmentation priors	24
Structural priors	24
Transition priors	25
Plane prior	25
3.3.3 Inference	26
3.3.4 Stixel Cut Prior	26
3.4 Generation of the Stixel cut prior	27
3.4.1 Time Series Compression	27

3.4.2	FCN-based method	29
Network architecture	30	
Training data	30	
Training strategies	30	
3.5	Experiments	32
3.5.1	Datasets	32
3.5.2	Experiment details	33
Metrics	33	
Baseline	35	
Input	35	
3.5.3	Results	35
3.6	Conclusions	37
4	Stixels on the GPU	39
4.1	Introduction	39
4.2	Related work	41
4.3	The Stixel Model	42
4.3.1	Mathematical formulation	43
4.3.2	Algorithm based on dynamic programming	44
4.3.3	Reduce the Algorithm's Complexity using SATs	45
4.3.4	Modified measurement model for slanted Stixels	46
4.4	Massive Parallelization	47
4.4.1	Downsampling and transpose	47
4.4.2	Computation of Summed Area Tables	49
4.4.3	Dynamic Programming stage	49
4.4.4	Backtracking and Data compaction	51
4.5	Experiments	52
4.5.1	Accuracy and Compression experiments	52
Datasets	52	
Experiment Details	52	
Results	53	
4.5.2	Performance experiments	53
Results	54	
4.6	Conclusion	56
5	Conclusions	59
5.1	Future work	60
Bibliography		61

List of Figures

1.1	Each pixel of I_{Base} corresponds to one pixel of I_{Match} , and the epipolar geometry of the two cameras limits the search of the matching pixel to the ones contained in a single line. The distance z between the 3D point and the baseline of the camera is computed from the disparity d using triangulation, where f is the focal length and T is the baseline of the camera pair.	2
1.2	Example of the Stixel segmentation and labeling of a column in a typical scene (on the right). The input disparity measurements (black thin lines) and output Stixels encoded with semantic colors (colored thick lines) are shown on the left. Taken from [27].	3
2.1	Each pixel of I_{Base} corresponds to one pixel of I_{Match} , and the epipolar geometry of the two cameras limits the search to a one dimensional line. The distance z between the 3D point and the baseline of the camera is computed from the disparity d using triangulation, where f is the focal length and T is the baseline of the camera pair.	8
2.2	Stages of the GPU-accelerated Disparity Estimation Pipeline	8
2.3	CSCT: 2D-tiled CTA-parallel scheme and computational analysis	10
2.4	Matching cost: 1D-tiled CTA-parallel scheme and computational analysis	11
2.5	Aggregated cost, Top-to-Bottom: CTA-parallel scheme with recurrence in the y-dimension and computational analysis	13
2.6	Performance (fps), performance per Watt and accuracy results for 640×480 px images, 128 disparity levels, and 2, 4 and 8 SGM path directions	15
2.7	Example of disparity computation	15
3.1	The proposed approach: pixel-wise color, semantic and depth information serve as input to our Slanted Stixels model, which is a compact semantic representation of a 3D scene that accurately handles arbitrary scenarios such as San Francisco city. The optional over-segmentation in the top-right yields significant speed gains while nearly retaining the depth and semantic accuracy.	17
3.2	Scene representation obtained by our method of a challenging street environment with a slanted road. Both geometric (top) and semantic (bottom) representations are shown.	18
3.3	Example of input disparity measurements (black lines) and output Stixels encoded with semantic colors (colored lines) for a typical scene column (right). Adapted from [12].	21
3.4	Comparison of original [51] (top) and our Slanted Stixels (bottom): due to the fixed slant in the original formulation, the road surface is not well represented as illustrated on the top-left figure. The novel model is capable of reconstructing the whole scene accurately.	23

3.5	Stixel inference illustrated as shortest path problem on a directed acyclic graph: the Stixel segmentation is computed by finding the shortest path from the source (left gray node) to the sink (right gray node). The vertices represent Stixels with colors encoding their geometric class, <i>i.e.</i> ground, object and sky. Only the incoming edges of ground nodes are shown for simplicity. Adapted from [12].	28
3.6	Generated Stixel cuts (highlighted in red) using the left and right extrema as defined in [31], and also cuts generated from semantic segmentation. Stixel cut density is 30%, equivalent to a $3.3 \times$ reduction in vertical resolution.	29
3.7	Generated Stixel cuts (highlighted in red) for the FCN-based method. Stixel cut density is 31.5%, equivalent to a $3.2 \times$ reduction in vertical resolution.	29
3.8	Definition of the proposed Fully Convolutional Network for generating Stixel cuts.	31
3.9	Sample image from the real-data sequence used for Stixel cut generation. Stixel cut ground-truth is highlighted in red.	32
3.10	The SYNTHIA-SF Dataset. A sample frame (left) with its depth (center) and semantic labels (right).	32
3.11	Frame-rate of our method (measuring only Stixel computation step and corresponding over-segmentation approach) compared to Semantic Stixels [51] for SYNTHIA-SF (image resolution of 1920×1080) on a multi-threaded CPU implementation (Intel i7-6800K) computed with a Stixel width of 8 pixels and same Stixel height. Two over-segmentation approaches are compared: Time Series <i>c.f.</i> section 3.4.1, FCN <i>c.f.</i> section 3.4.2.	33
3.12	Some results on real data: with non-flat roads our model correctly models the scene, while retaining accuracy on objects. We show a failure case in the last row, our approach detects part of the road as sidewalk because of incorrect semantic input. As we highlight with a red circle, the original approach detects a wall, this could lead to an unnecessary emergency break.	36
4.1	Scene representation of a challenging street environment obtained by our method. Geometric (left), semantic (center) and 3D (right) representations are shown. In fig. 4.1a, color encodes the distance from close (red) to far (green). In fig. 4.1b, color encodes the semantic class following [11].	40
4.2	Example of the Stixel segmentation and labeling of a column in a typical scene (on the right). The input disparity measurements (black thin lines) and output Stixels encoded with semantic colors (colored thick lines) are shown on the left. Taken from [27].	42
4.3	Downsampling and Transpose: computational analysis and parallel scheme. The width and height of the input images are W and H . There are z channels corresponding to disparity map, disparity confidence and semantic probabilities. The horizontal and vertical Stixel resolution is defined by s and t , respectively. Accordingly, $h = \frac{H}{s}$ and $w = \frac{W}{t}$	48

4.4	Fused stages for generating SATs, Dynamic Programming (DP) computation, and Backtracking. The computational analysis and parallel scheme are shown for the original proposal [27] and our proposal. w , h , z are defined in fig. 4.3. v_{bottom} , v_{top} , d_{slope} , $d_{intercept}$, and C define the Stixel properties: bottom & top row, depth model and semantic class.	50
4.5	Exemplary outputs on real data (KITTI 2015): RGB Image (left), Slanted Stixels [27] (center) and Our Stixels (right) representations are shown. Color encodes the distance from close (red) to far (green). As we can see, the representation of our proposal is visually similar to the baseline.	54
4.6	Frame-rate of our method compared to Slanted Stixels [27] for 2048×1024 image resolution on the NVIDIA Tegra Xavier and Tegra X2 embedded GPUs.	56
4.7	Breakdown of low-level performance metrics for the two main stages of our method: <i>Down-sampling & Transpose</i> stage (common) and <i>Dynamic Programming</i> stage (ours vs. [27]). <i>IPC</i> is the ratio of machine instructions executed per clock cycle and per SM. Image size is 2048×1024	57

List of Tables

3.1	We report frame-rate for each stage of our pipeline for a stereo pair of 1242×375 image resolution. OS stands for <i>Over-segmentation</i> . Stixel methods and SGM run-time is obtained using a CPU Intel i7-6800K. A NVidia Maxwell Titan X is used for obtaining <i>Semantic Segmentation</i> frame-rate. Note that Stixel frame-rate is variable when using an over-segmentation method, a representative run-time is provided. The total frame-rates are reported for the whole pipeline.	34
3.2	Accuracy of our approach compared to Semantic Stixels [51], raw SGM and FCN. We provide results on four datasets: Ladicky [34], KITTI 15 [19], Cityscapes [11] and SYNTHIA-SF using metrics: Disparity Error (less is better) and Intersection over Union (more is better) <i>c.f.</i> section 3.5.1 and section 3.5.2. <i>Fast</i> (over-segmentation) methods are presented in section 3.4.1 and section 3.4.2. Significantly best results are highlighted in bold.	34
3.3	Number of Stixels (10^3) obtained by our methods compared to Semantic Stixels [51] and raw input (total number of pixels). We provide results on four datasets: Ladicky [34], KITTI 15 [19], Cityscapes [11] and SYNTHIA-SF <i>c.f.</i> section 3.5.1. <i>Fast</i> (oversegmentation) methods are presented in section 3.4.1 and section 3.4.2.	35
4.1	Data compression measured in pixels per Stixel of our method and [27]. We evaluate on three datasets: KITTI 2015 [19], Cityscapes [11] and SYNTHIA-SF [28], <i>c.f.</i> section 4.5.1.	53
4.2	Accuracy of our method compared to Slanted Stixels [27], input SGM and FCN. We evaluate on three datasets: KITTI 15 [19], Cityscapes [11] and SYNTHIA-SF [28] using these metrics: Disparity Error (lower is better) and Intersection over Union (higher is better), <i>c.f.</i> section 4.5.1 and section 4.5.1. <i>Ours</i> is detailed in section 4.3.4.	53
4.3	Specifications of GPUs employed in our experiments.	55

List of Abbreviations

ASIC	Application-Specific Integrated Circuit
FPGA	Field-Programmable Gate Array
CPU	Central Processing Unit
GPU	Graphics Processing Unit
CTA	Cooperative Thread Array
ADAS	Advanced Driver Assistance Systems
SGM	Semi-Global Matching
MC	Matching Cost
FCN	Fully-Connected Network
RGB	Red Green Blue

Chapter 1

Introduction

1.1 Problem Description

In this dissertation, we will investigate fast 3D reconstruction and scene understanding for autonomous driving through stereo vision. As this problem must be solved in real-time, we focus on fast algorithms and low-energy consumption devices, and also devise GPU-accelerated designs for these algorithms. In this introduction, we present the basic concepts of the topics and discuss the objectives of this work.

To this end, we start with an introduction to stereo vision in section 1.2, and their importance for autonomous driving applications. Next, we talk about the Stixel World algorithm in section 1.3. Then, we introduce the GPU architecture and its massively-parallel programming model in section 1.4. Finally, we present our main objectives in section 1.5 and contributions in section 1.6.

1.2 Stereo vision

The aim of stereo vision is to infer the depth of each pixel in the scene from two camera images. Both images must contain the same objects from a different view point. In the typical binocular stereo configuration, one camera is translated (not rotated) with respect to the other.

An object visible in both cameras will be projected in the image plane in a slightly different location: this difference in position is called *disparity*, and the closer an object is from the stereo rig, the higher the disparity will be. When the object is too far away, the disparity will be zero. Note that one can estimate more distance by increasing the *baseline* (*i.e.* distance between the cameras), with the added noise, computational and memory complexity associated with it.

Stereo vision algorithms are designed to estimate the *disparity* of each individual pixel. Typically, algorithms try to match each pixel in the image obtained from one camera to the corresponding pixel in the image from the other camera, using the neighbor pixels to improve the matching accuracy and reduce the effect of noise in the inputs.

Figure 1.1 describes the epipolar geometry [29] of the process: a point in 3D is projected onto the left and right images. For any point in the left image, *i.e.* I_{Base} , the depth is unknown. But we know that according to epipolar geometry, the corresponding point in the right image, *c.f.* I_{Match} , can be found somewhere on the epipolar line.

The problem is then reduced to a search along the epipolar line, which makes it faster and less prone to noise, since it is less likely to fail when you reduce the amount of matching candidates. Typically, there is a small rotation between the

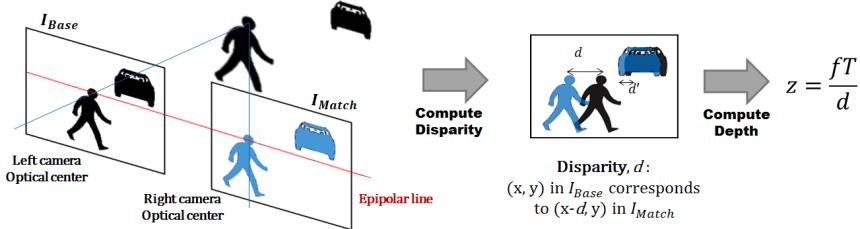


FIGURE 1.1: Each pixel of I_{Base} corresponds to one pixel of I_{Match} , and the epipolar geometry of the two cameras limits the search of the matching pixel to the ones contained in a single line. The distance z between the 3D point and the baseline of the camera is computed from the disparity d using triangulation, where f is the focal length and T is the baseline of the camera pair.

cameras (even when the set up was only a translation) and cameras have to be rectified [29]. After the rectification process, the epipolar line goes through the image row. Once the *disparity* has been recovered, we can derive the distance of each pixel by using the *focal length* and *baseline* (*i.e.* distance between the cameras).

Dense, robust and real-time computation of depth information from stereo-camera systems is a requirement in many industrial applications such as advanced driver assistance systems (ADAS), robotics navigation and autonomous vehicles. An efficient stereo algorithm has been a research topic for decades [2]. It has multiple applications, for example, Alejandro González *et al.* [20] uses stereo information to filter candidate windows for pedestrian detection and provides better accuracy and performance.

Disparity estimation is a difficult task because of the high level of ambiguity that often appears in real situations. For those, a large variety of proposals have been extensively presented [48]. Most of the high-accuracy stereo vision pipelines [60] include the semi-global matching (SGM) consistency-constraining algorithm [29]. The combination of SGM with different kinds of local similarity metrics is insensitive to various types of noise and interferences (like lighting), efficiently deals with large untextured areas and is capable of retaining edges. Recently, deep learning approaches have also been proposed [60, 61], providing state-of-the-art results. However, the neural networks used are still too complex to provide real time results on low-energy consumption embedded devices.

1.3 Stixel Model

The Stixel world is a compressed representation of a 3D scene that preserves its relevant structure. Given that the vertical dimension dominates the structure of street environments, the Stixel world segments the image into independent columns composed of stick-like super-pixels with a 3D planar depth model and semantic labels.

The Stixel model has been successfully used for automotive vision applications either to decrease parsing time, increase accuracy or both. We can find examples of works using the Stixel representation in different topics such as object recognition [6, 36], building a grid map over time [39] and for autonomous driving [63]. Specifically, for motion planning in the context of autonomous driving, the Stixel model has been used *c.f.* [63, 64] to model the geometric constraints of a given scene.

There are three structural classes derived exclusively from depth data: *ground* (Stixels with a slant similar to the expected ground plane), *object* (almost vertical

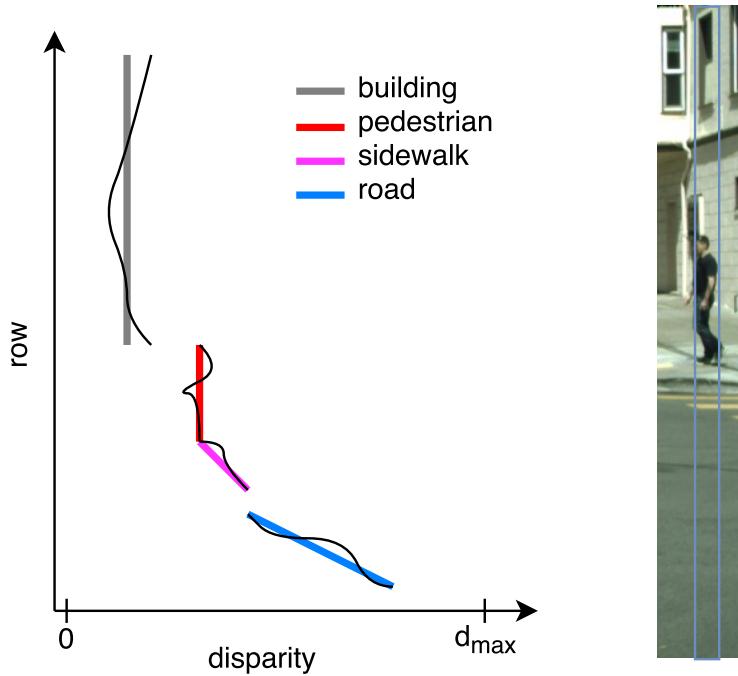


FIGURE 1.2: Example of the Stixel segmentation and labeling of a column in a typical scene (on the right). The input disparity measurements (black thin lines) and output Stixels encoded with semantic colors (colored thick lines) are shown on the left. Taken from [27].

Stixels, usually lying on the ground), and *sky* (Stixels at infinite distance). Semantic classes are refinements to those structural classes (*e.g.* road or sidewalk are ground classes, whereas building and vehicle are object classes). Prior to the segmentation, the per-pixel input images are downsized to the desired vertical and horizontal Stixel resolution.

An example of Stixel segmentation is presented in fig. 1.2. The column highlighted in the image on the right is downsized, and the disparity measurements (inverse of depth) for each Stixel on the column are shown on the left. The resulting Stixel segmentation and labeling are defined by the colored thick lines.

A Stixel column segmentation S consists of an arbitrary number N of Stixels, s_i , each representing four random variables: the Stixel extent via bottom row V_i^b and top row V_i^t , as well as its semantic class C_i and depth model D_i (slope and intercept). Thereby, the number of Stixels itself is a random variable that is optimized jointly during inference. The joint segmentation and labeling problem is carried out independently for each image column via optimization of the posterior distribution $P(S | M)$, a Maximum A Posteriori estimation problem (MAP) defined over a Stixel segmentation S given all measurements M from that particular column.

As described in [42], we can design a recursive definition of the optimization problem in order to solve the problem using a Dynamic Programming scheme. The computational complexity is $\mathcal{O}(w \times h^2)$, where w is the image width and h is the image height.

1.4 GPU Parallelization

GPUs are massively-parallel devices containing tens of throughput-oriented processing units called *streaming multiprocessors* (SMs). Compute and memory operations are executed as vector (SIMD) instructions and are highly pipelined in order to save energy and transistor budget. SMs can execute several vector instructions per cycle, selected from multiple independent execution flows: the higher the available instruction-, vector- and thread-level parallelism, the better the pipeline utilization.

The CUDA programming model merges vector-level and thread-level parallelism, and allows defining a very large number of potentially concurrent execution instances (called *threads*) of the same program code. A unique two-level identifier (*ThrId*, *CTAid*) is used to specialize each thread for a particular data and/or function. A CTA (*Cooperative Thread Array*) comprises all the threads with the same *CTAid*, which run simultaneously and until completion in the same SM, and can share a fast but limited memory space: the so-called *Shared Memory*.

The CUDA 9.0 specification introduces *cooperative groups* to dynamically organize groups of threads to perform collective operations involving communication and synchronization, which enable complex patterns of parallel cooperation. The hardware scheduler maps threads in the same cooperative group to vector instructions, which are executed efficiently, specially when the size of the group matches the hardware warp size.

Each thread has its own private *Local Memory* space, commonly mapped to registers by the compiler. A large space of *Global Memory* is public to all execution instances, and is mapped into a large-capacity but long-latency device memory, which is accelerated using a two-level hierarchy of cache memories.

The parallelization scheme of an algorithm and the data layout determine the available parallelism at the instruction, vector and thread level and the memory access pattern. A large amount of parallelism is required for hiding operation latencies and achieving high resource usage. Additionally, efficient memory performance requires that the set of addresses generated by a group of consecutive threads refer to consecutive positions that can be *coalesced* into a single, wider memory transaction. Since the bandwidth of the device memory is often a performance bottleneck, an efficient CUDA code should promote data reuse on the internal caches, the shared memory, and the registers.

This thesis focuses on embedded GPU-accelerated systems, like the NVIDIA Jetson and DrivePX platforms, that allow low-cost and low-energy consumption, massively parallel computation. Because of these features, they are a better fit for autonomous driving than more powerful high-end GPUs. We have designed the parallelization to extract inherent parallelism, or to augment the parallelism and to maximize temporal (and spacial) locality of the algorithms to use the GPUs efficiently.

1.5 Objectives and motivation

The objective of this thesis is to study 3D reconstruction algorithms suitable for autonomous driving. In order to do so, we need fast implementations and representations of the 3D environment that take into account geometric and semantic information. The use of CUDA and GPU parallelization provides flexible and programmable high performance hardware that can be used to fulfill those low-latency requirements.

1.6 Main Contributions

The contributions of this thesis are organized in three chapters. The research presented in each chapter has been published in peer-reviewed conferences and journals.

1.6.1 Parallelization of SGM for GPU

Chapter 2 presents the parallelization of the well-known stereo matching algorithm Semi-Global Matching (SGM). We provide an efficient parallelization strategy that achieves real-time performance on low-energy consumption GPUs. This work was published in:

[25] Daniel Hernandez-Juarez et al. "Embedded Real-time Stereo Estimation via Semi-Global Matching on the GPU". In: *International Conference on Computational Science 2016, ICCS 2016, 6-8 June 2016, San Diego, California, USA*. 2016, pp. 143–153. DOI: [10.1016/j.procs.2016.05.305](https://doi.org/10.1016/j.procs.2016.05.305). URL: <http://dx.doi.org/10.1016/j.procs.2016.05.305>

1.6.2 Slanted Stixels

Chapter 3 presents our proposal to improve the Stixel model to account for slanted surfaces. This model modification helps representing real scenes that fail under the previous assumptions while keeping the accuracy by an efficient model regularization. We also propose an algorithmic speed up to reduce the amount of Stixel combinations tested by the dynamic programming approach. This work was published in:

[28] Daniel Hernandez-Juarez et al. "Slanted Stixels: Representing San Francisco's Steepest Streets". In: *British Machine Vision Conference 2017, BMVC 2017, London, UK, September 4-7, 2017*. BMVA Press, 2017. URL: <https://www.dropbox.com/s/m0n8ujh1sxtvge1q/0406.pdf?dl=1>

[27] Daniel Hernandez-Juarez et al. "Slanted Stixels: A Way to Represent Steep Streets". In: *International Journal of Computer Vision* (Sept. 2019), pp. 1–16. ISSN: 0920-5691. DOI: [10.1007/s11263-019-01226-9](https://doi.org/10.1007/s11263-019-01226-9). URL: <https://doi.org/10.1007/s11263-019-01226-9>

1.6.3 Parallelization of Stixels for GPU

Chapter 4 presents our work on parallelization of the Stixel algorithm. We propose a parallelization strategy that fits the massively parallel architecture of GPUs. We also study the different speed up techniques available for Stixels and how they can be implemented efficiently for this architecture. Additionally, our approach reduces the computational complexity of the algorithm by reformulating the measurement depth model, relying on the confidence of the depth estimation and the identification of invalid values to handle outliers. This work was published in:

[26] Daniel Hernandez-Juarez et al. "GPU-Accelerated Real-Time Stixel Computation". In: *2017 IEEE Winter Conference on Applications of Computer Vision, WACV 2017, Santa Rosa, CA, USA, March 24-31, 2017*. 2017, pp. 1054–1062. DOI: [10.1109/WACV.2017.122](https://doi.org/10.1109/WACV.2017.122). URL: <https://doi.org/10.1109/WACV.2017.122>

And the paper under review:

[24] Daniel Hernandez-Juarez et al. "3D Perception with Slanted Stixels on GPU". In: *under review on IEEE Trans. Parallel Distrib. Syst.* (forthcoming)

Chapter 2

Semi-global Matching Parallelization

2.1 Introduction

Dense, robust and real-time computation of depth information from stereo-camera systems is a requirement in many industrial applications such as advanced driver assistance systems (ADAS), robotics navigation and autonomous vehicles. An efficient stereo algorithm has been a research topic for decades [2]. It has multiple applications, for example, [20] uses stereo information to filter candidate windows for pedestrian detection and provides better accuracy and performance.

Fig. 2.1 illustrates how to infer the depth of a given real-world point from its projection points on the left and right images. Assuming a simple translation between the cameras (otherwise, images must be rectified using multiple extrinsic and intrinsic camera parameters), the corresponding points must be in the same row of both images, along the epipolar lines. A similarity measure correlates matching pixels and the *disparity* (d) is the similarity distance between both points.

Disparity estimation is a difficult task because of the high level of ambiguity that often appears in real situations. For those, a large variety of proposals have been extensively presented [48]. Most of the high-accuracy stereo vision pipelines [60] include the semi-global matching (SGM) consistency-constraining algorithm [29]. The combination of SGM with different kinds of local similarity metrics is insensitive to various types of noise and interferences (like lighting), efficiently deals with large untextured areas and is capable of retaining edges.

The high computational load and memory bandwidth requirements of SGM pose hard challenges for fast and low energy-consumption implementations. Dedicated hardware solutions (*e.g.* FPGA or ASIC) [4, 41] achieve these goals, but they are very inflexible regarding changes in the algorithms. Implementations on desktop GPUs can assure real-time constraints [4], but their high power consumption and the need to attach a desktop computer makes them less suitable for embedded systems.

Recently, with the appearance of embedded GPU-accelerated systems like the NVIDIA Jetson TX1 and the DrivePX platforms (incorporating, respectively, one and two Tegra X1 ARM processors), low-cost and low-consumption real-time stereo computation is becoming attainable. The objective of this chapter is to implement and evaluate a complete disparity estimation pipeline on this embedded GPU-accelerated device.

We present simple, but well-designed, baseline massively parallel schemes and data layouts of each of the algorithms required for disparity estimation, and then optimize the baseline code with specific strategies, like vectorization or *CTA-to-Warp* conversion, to boost performance around 3 times. The optimized implementation runs on a single Tegra X1 at 42 frames per second (fps) for an image size of 640×480

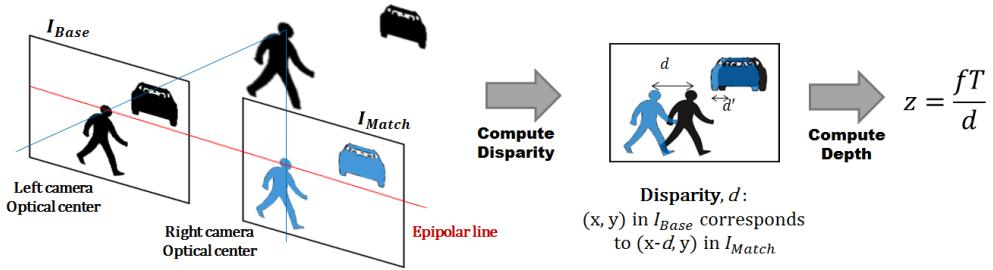


FIGURE 2.1: Each pixel of I_{Base} corresponds to one pixel of I_{Match} , and the epipolar geometry of the two cameras limits the search to a one dimensional line. The distance z between the 3D point and the baseline of the camera is computed from the disparity d using triangulation, where f is the focal length and T is the baseline of the camera pair.

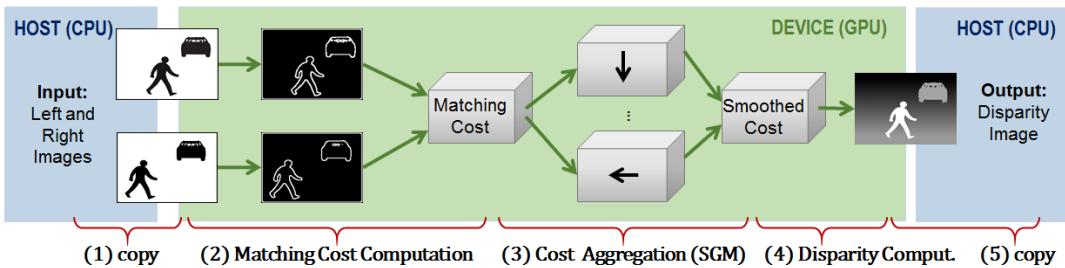


FIGURE 2.2: Stages of the GPU-accelerated Disparity Estimation Pipeline

pixels, 128 disparity levels, and using 4 path directions for the SGM method, providing high-quality real-time operation. While a high-end desktop GPU improves around 10 times the performance of the embedded GPU, the performance per watt ratio is 2.2 times worse.

The rest of the chapter is organized as follows. Section 2.2 presents the algorithms composing the disparity estimation pipeline. In section 2.3 we describe each algorithm and then propose and discuss a parallel scheme and data layout. Finally, section 2.4 provides the obtained results and section 2.5 summarizes the work.

2.2 Disparity Estimation Pipeline

Fig. 2.2 shows the stages of the disparity computation pipeline: (1) the captured images are copied from the Host memory space to the GPU Device; (2) features are extracted from each image and used for similarity comparison to generate a local matching cost for each pixel and potential disparity; (3) a smoothing cost is aggregated to reduce errors (SGM); (4) disparity is computed and a 3×3 median filter is applied to remove outliers; and (5) the resulting disparity image is copied to the Host memory.

2.2.1 Local Matching Cost and Semi-Global Matching (SGM)

Different similarity metrics or cost functions have been proposed in the literature. The less computationally-demanding, and modest quality providers, are Sum of Absolute Differences, ZSAD and Rank Transform. According to [30], Hierarchical Mutual Information and the Census Transform (CT) features [59] provide similar higher quality, being CT substantially less time-consuming. Recently, costs based on neural

networks have outperformed CT [60], but at the expense of a higher computational load.

A CT feature encodes the comparisons between the values of the pixels in a window around a central pixel. After empirically evaluating different variants we selected a Center-Symmetric Census Transform (CSCT) configuration with a 9×7 window, which provides a more compact representation with similar accuracy [53]. The similarity of two pixels is defined as the Hamming distance of their CSCT bit-vector features. Two properties provide robustness for outdoor environments with uncontrolled lighting and in front of calibration errors: the invariance to local intensity changes (neighboring pixels are compared to each other) and the tolerance to outliers (an incorrect value modifies a single bit).

In order to deal with non-unique or wrong correspondences due to low texture and ambiguity, consistency constraints can be included in the form of a global two-dimensional energy minimization problem. Semi-global matching (SGM) approximates the global solution by solving a one-dimensional minimization problem along several (typically 4 or 8) independent paths across the image. For each path direction, image point and disparity, SGM aggregates a cost that considers the cost of neighboring points and disparities. The number of paths affects both the quality and the performance of the results.

2.2.2 Related work

A reference implementation of SGM on CPU [54] reached 5.43 frames per second (fps) with 640×480 image resolution and 128 disparity levels. They applied SGM with 8 path directions ad an additional left-right consistency check and sub-pixel interpolation. A modified version with reduced disparity computation (rSGM) was able to reach 12 fps.

Early GPU implementations [16] and [47] present OpenGL/Cg SGM implementations with very similar performance results peaking at 8 fps on 320×240 resolution images. Versions designed for early CUDA systems and proposed specific modifications of the SGM algorithm. Haller and Nedevschi [21] modified the original cost aggregation formula removing the P1 penalty and using 4 path directions for cost aggregation. In this way, they reduced computation and memory usage, but also reduced accuracy. Their implementation reached 53 fps on a Nvidia GTX 280 with images of 512×383 .

The most recent implementation [4] stated very fast results: 27 fps on 1024×768 images using a NVIDIA Tesla C2050, with 128 disparity levels. By using Rank Transform [59] as matching cost function, their proposal provides lower accuracy [30]. We will notice some differences in the parallel scheme on the following discussion.

As far as we know this is the first evaluation of disparity estimation in a Nvidia GPU-accelerated embedded system, as well as in the last Maxwell architecture. We propose better parallelization schemes to take advantage of the hardware features available in current systems.

2.3 Algorithm Description and Massive Parallelization

This section describes the algorithms used for disparity computation and discusses the alternative parallelization schemes and data layouts. We present the baseline pseudocode for the proposed massively parallel algorithms and explain additional optimizations.

2.3.1 Matching Cost Computation

A 9×7 -window, Center-Symmetric Census Transform (CSCT) concatenates the comparisons of 31 pairs of pixels into a bit-vector feature. Equation 2.1 defines the CSCT, where \otimes is bit-wise concatenation, $I(x, y)$ is the value of pixel (x, y) in the input image, and $s(u, v)$ is 1 if $u \geq v$, or 0 otherwise. The matching cost $MC(x, y, d)$ between a pixel (x, y) in the base image and each potentially corresponding pixel in the match image at disparity d is defined by equation 2.2, where \oplus is bit-wise exclusive-or and *bitcount* counts the number of bits set to 1.

$$CSCT_{9,7}(I, x, y) = \bigotimes \begin{cases} \bigotimes_{i=1}^4 \bigotimes_{j=-3}^3 s(I(x+i, y+j), I(x-i, y-j)) \\ \bigotimes_{j=1}^3 s(I(x, y+j), I(x, y-j)) \end{cases} \quad (2.1)$$

$$MC(x, y, d) = \text{bitcount}(CSCT_{9,7}(I_{\text{base}}, x, y) \oplus CSCT_{9,7}(I_{\text{match}}, x-d, y)) \quad (2.2)$$

The data access patterns inherent in both equations exhibit different data reuse schemes, which prevent both algorithms to be fused. The 2D-tiled parallel scheme shown in Fig. 2.3 matches the 2D-stencil computation pattern of CSCT, and maximizes data reuse: the attached table shows how a tiled scheme using shared memory reduces the total global data accesses by $(62 + 4)/(1.5 + 4) = 12$ times with respect to a straightforward, naïve, embarrassingly parallel design, where each thread reads its input values directly from global memory.

The 1D-tiled parallel scheme for computing matching cost (MC) exploits data reuse on the x-dimension (see Fig. 2.4). As proposed in [4], we can represent matching cost using a single byte without losing accuracy, which reduces 4 times the memory bandwidth requirements in comparison to using 32-bit integers. The attached table shows that the read-cooperative scheme, compared to the naïve design, sacrifices parallelism (divides the number of threads by D , the maximum disparity considered) by higher data reuse (around 8 times less global memory accesses). The low arithmetic intensity of the algorithm (2 main compute operations every 9-Byte memory accesses) advises for this kind of optimization.

Algorithms 1 and 2 show the pseudocode of both parallel algorithms, not including special code for corner cases handling image and CTA boundaries. In both cases, threads in the same CTA cooperate to read an input data tile into shared memory, then synchronize, and finally perform the assigned task reading the input data from shared memory. The first algorithm assumes a CTA size of $WarpSize \times WarpSize$ threads and the second algorithm a CTA of D threads. They are both scalable designs that use a small constant amount of shared memory per thread (1.5 and 12 Bytes, respectively).

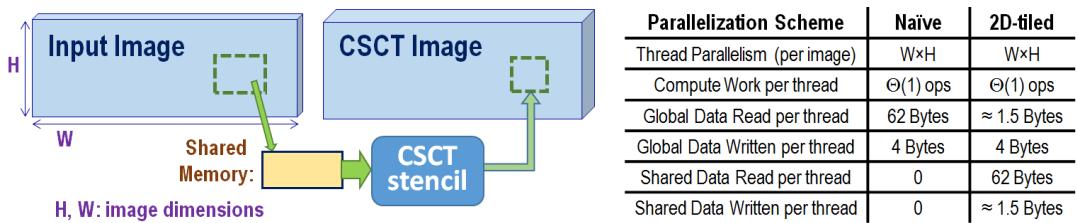


FIGURE 2.3: CSCT: 2D-tiled CTA-parallel scheme and computational analysis

```

input : I[H][W], H, W
output: CSCT[H][W]
parallel for  $y=0$  to H step WarpSize do
    parallel for  $x=0$  to W step WarpSize do
        CTA parallel for  $y_{CTA}, x_{CTA}=(0,0)$  to (WarpSize,WarpSize) do
            copy ( $WarpSize + 8 \times (WarpSize + 6)$ ) tile of I[][] into SharedI[][];
            CTA Barrier Synchronization;
            CSCT[y+yCTA][x+xCTA] = CSCT9,7(SharedI, xCTA, yCTA);
        end
    end
end

```

Algorithm 1: CSCT: 2D-tiled, read-cooperative parallel scheme

There are two memory-efficient layout alternatives for algorithm 2. Each CTA generates a $D \times D$ slice in the y-plane of the MC matrix, and threads can generate together the cost for (1) all the disparity levels for the same pixel or (2) all the pixels in the block for the same disparity level. We chose the first option, and adapt the data layout so that the indexes of disparity levels vary faster on the MC cube and global write instructions are coalesced. The second solution, used in [4], provides similar performance on this algorithm but compromises the available parallelism and the performance of the following SGM algorithm.

2.3.2 Smoothing Cost Aggregation (SGM) and Disparity Computation

The SGM method solves a one-dimensional minimization problem along different paths $r=(r_x, r_y)$ using the recurrence defined by equation 2.3 and a dynamic programming algorithmic pattern. Matrix L_r contains the smoothing aggregated costs for path r . The first term of equation 2.3 is the original matching cost, and the second term adds the minimum cost of the disparities corresponding to the previous pixel $(x - r_x, y - r_y)$, including penalties for small disparity changes (P_1) and for larger disparity discontinuities and (P_2). P_1 is intended to detect slanted and curved surfaces, while P_2 smooths the results and makes abrupt changes difficult. The last term ensures that aggregated costs are bounded. For a detailed discussion refer to [29]. The different L_r matrices must be added together to generate a final cost and then select the disparity corresponding to the minimum (*winner-takes-all* strategy), as shown

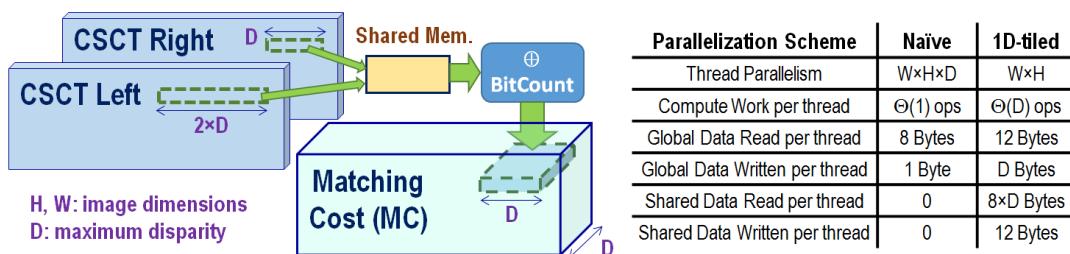


FIGURE 2.4: Matching cost: 1D-tiled CTA-parallel scheme and computational analysis

```

input : CSCTbase[H][W], CSCTmatch[H][W], H, W, D
output: MC[H][W][D]

parallel for  $y=0$  to H do
    parallel for  $x=0$  to W step D do
        CTA parallel for  $ThrId=0$  to D do
            SharedM[ $ThrId$ ] = CSCTmatch[y][ $x+ThrId-D$ ];
            SharedM[ $D+ThrId$ ] = CSCTmatch[y][ $x+ThrId$ ];
            SharedB[ $ThrId$ ] = CSCTbase[y][ $x+ThrId$ ];
            CTA Barrier Synchronization;
            for  $i=0$  to D do
                MC[y][ $x+i$ ][ $ThrId$ ] = BitCount ( SharedB[i]  $\oplus$ 
                    SharedM[ $ThrId+1+i$ ]);
            end
        end
    end
end

```

Algorithm 2: Matching Cost computation: 1D-tiled, read-cooperative parallel scheme; Data layout: MC[y][x][d] (d indexes vary faster)

by equation 2.4.

$$L_r(x, y, d) = MC(x, y, d) + \min \begin{cases} L_r(x - r_x, y - r_y, d) \\ L_r(x - r_x, y - r_y, d - 1) + P_1 \\ L_r(x - r_x, y - r_y, d + 1) + P_1 \\ \min_i L_r(x - r_x, y - r_y, i) + P_2 \end{cases} - \min_k L_r(x - r_x, y - r_y, k) \quad (2.3)$$

$$D(x, y) = \min_d \sum_r L_r(x, y, d) \quad (2.4)$$

Equation 2.3 determines a recurrent dependence that prevents the parallel processing of pixels in the same path direction. Parallelism can be exploited, though, in the direction perpendicular to the path, in the disparity dimension, and for each of the computed path directions. Our proposal exploits all the available parallelism by creating a CTA for each slice in the aggregated cost matrix along each particular path direction.

Fig. 2.5 illustrates the case of the top-to-bottom path direction and algorithm 3 shows the pseudocode. Each of the W slices is computed by a different CTA of D threads, with each thread executing a recurrent loop (line 4) to generate H cost values along the path. Computing the cost for the current pixel and disparity level requires the cost of the previous pixel on neighboring disparity levels: one value can be reused in a private thread register but the neighboring costs must be communicated among threads (lines 7,8 and 12). Finally, all threads in the CTA must collaborate to compute the minimum cost for all disparity levels (line 11).

The case for horizontal paths is very similar, with H slices computed in parallel. Diagonal path directions are a little more complex: W independent CTAs process the diagonal slices moving in a vertical direction (assuming $W \geq H$). When a CTA reaches a boundary, it continues on the other boundary. For example, a top-to-bottom and right-to-left diagonal slice starting at $(x,y) = (100,0)$ will successively

```

input : MC[H][W][D], H, W, D
output: L[H][W][D]

parallel for  $x=0$  to W do
    CTA parallel for  $ThrId=0$  to D do
        Initialize aggr, min and SharedAggr[] with MAX_VALUE;
        for  $y=0$  to H do
            cost = MC[y][x][ThrId];
            CTA Barrier Synchronization;
            left = SharedAggr[ThrId];
            right= SharedAggr[ThrId+2];
            aggr = cost + minimum( aggr, left+P1, right+P1, min+P2 ) - min;
            L[y][x][ThrId] = aggr;
            min = CTA_Minimum_Reduce ( aggr ); *** includes Barrier
            Synchronization SharedAggr[ThrId+1] = aggr;
        end
    end
end

```

Algorithm 3: Aggregated Cost computation: top-to-bottom path direction

process pixels (99,1), (98,2) ... (0, 100), and then will reset the costs corresponding to the previous pixel and continue with pixels (W-1,101), (W-2,102) ...

The cost aggregation and disparity computation defined by equation 2.4 have been fused in Algorithm 4 in order to reduce the amount of memory accesses (avoids writing and then reading the final cost matrix). A CTA-based parallel scheme is proposed so that each CTA produces the disparity of a single pixel (line 7): first, each CTA thread adds the costs corresponding to a given disparity level for all path directions (line 4), and then CTA threads cooperate to find the disparity level with minimum cost (line 5).

2.3.3 Additional Optimizations

We have applied three types of optimizations to the baseline algorithms that provided a combined performance improvement of almost $3\times$. We have vectorized the inner loop of algorithm 3 (lines 4-12) to process a vector of 4 cost values (4 bytes) per instruction (requiring a special byte-wise SIMD instructions for computing the minimum operation). We have also modified the parallel scheme so that a single warp performs the task previously assigned to a CTA, which we call *CTA-to-warp* conversion. It (1) avoids expensive synchronization operations, (2) allows using fast register-to-register communication (using special shuffle instructions) instead

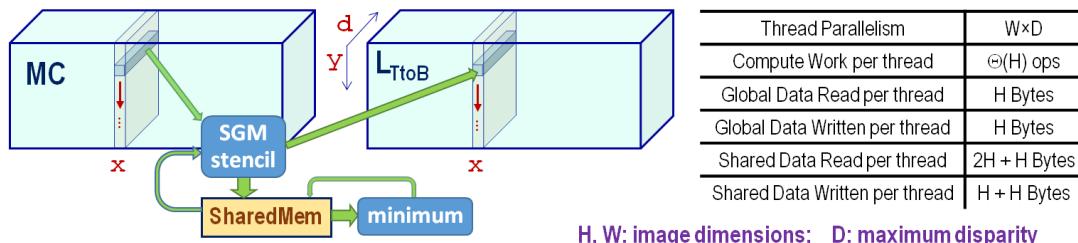


FIGURE 2.5: Aggregated cost, Top-to-Bottom: CTA-parallel scheme with recurrence in the y-dimension and computational analysis

```

input : L0[W][H][D], L1[W][H][D], L2[W][H][D] ... W, H, D
output: Disp[W][H]

parallel for x=0 to W do
    parallel for y=0 to H do
        CTA parallel for ThrId=0 to D do
            cost = L0[x][y][ThrId]+L1[x][y][ThrId]+L2[x][y][ThrId]+...;
            MinIndex = CTA_Minimum_Reduce(cost, ThrId);
            if ThrId == 0 then
                | Disp[x][y] = MinIndex;
            end
        end
    end

```

Algorithm 4: Summation of all path costs and Disparity Computation

of shared-memory communications, and (3) reduces instruction count and increases instruction-level parallelism. A drawback of both strategies is a reduction of thread-level parallelism, as shown in [9]. This is not a severe problem in the embedded Tegra X1 device, with a maximum occupancy of ≈ 4 thousand threads.

Finally, to reduce the amount of data accessed from memory, the computation of the aggregated cost for the last path direction (Alg. 3 Bottom-to-Top) is fused with the final cost summation and disparity computation (Alg. 4), providing a 1.35x performance speedup on the Tegra X1. Also, fusing the computation of the initial matching cost (Alg. 2) with the aggregate cost computation for the horizontal path directions (Alg. 3) improves performance by 1.13x.

2.4 Results

We have measured execution time and disparity estimation accuracy for multiple images, 128 disparity levels, and 2, 4 and 8 path directions. Apart from executing on a NVIDIA Tegra X1, which integrates 8 ARM cores and 2 Maxwell SMs with a TDP of 10W, and for comparison purposes, we have also executed on a high-end NVIDIA Titan X, with 24 Maxwell SMs and a TDP of 250W. We ignore the time for CPU-GPU data transfers (less than 0.5% of the total elapsed time) since it can be overlapped with computation. Since performance scales proportional to the number of image pixels, we will restrict our explanation to 640×480 images.

The legend in Fig. 2.6 indicates the disparity estimation accuracy, measured using the KITTI benchmark-suite [19], when using different SGM configurations, and not considering occluded pixels and treating more than 3 pixel differences as errors. Using 4 path directions (excluding diagonals) reduces accuracy very slightly, while using only the left-to-right and top-to-bottom directions reduces accuracy more noticeably.

The left and right charts in Fig. 2.6 show, respectively, the performance throughput (frames per second, or fps) and the performance per watt (fps/W) on both GPU systems and also for different SGM configurations. The high-end GPU always provides more than 10 times the performance of the embedded GPU (as expected by the difference in number of SMs), but the latter offers around 2 times more performance per Watt. It is remarkable that real-time rates (42 fps) with high accuracy are achieved by the Tegra X1 when using 4 path directions.

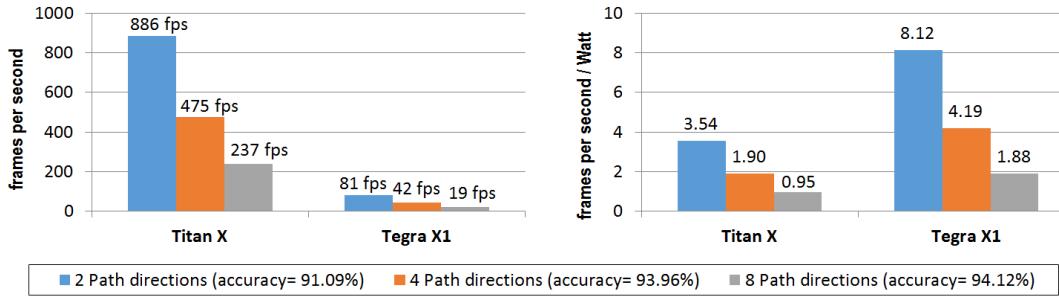


FIGURE 2.6: Performance (fps), performance per Watt and accuracy results for 640×480 px images, 128 disparity levels, and 2, 4 and 8 SGM path directions

Finally, an example of the disparity computed by our proposed algorithm can be seen in Fig. 2.7b.

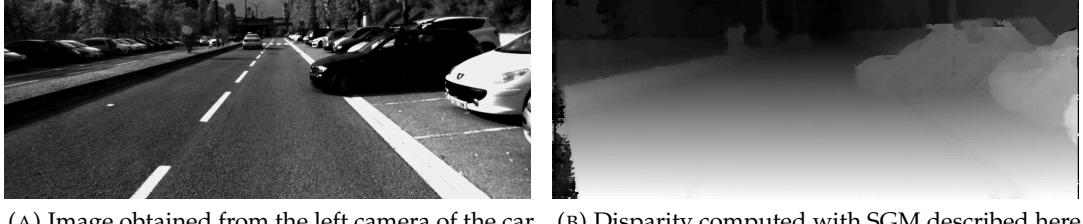


FIGURE 2.7: Example of disparity computation

2.5 Conclusions

The results obtained show that our implementation of depth computation for stereo-camera systems is able to reach real-time performance on a Tegra X1. This fact indicates that low-consumption embedded GPU systems, like the Tegra X1, are well capable of attaining real-time processing demands. Hence, their low-power envelope and remarkable performance make them good target platforms for real-time video processing, paving the way for more complex algorithms and applications.

We have proposed baseline parallel schemes and data layouts for the disparity estimation algorithms that follow general optimization rules based on a simple GPU performance model. They are designed to gracefully scale on the forthcoming GPU architectures, like NVIDIA Pascal. Then, we have optimized the baseline code and improved performance around 3 times with different specific strategies, like vectorization or CTA-to-Warp conversion, that are also expected to be valid for forthcoming architectures.

We plan to prove the higher performance potential of the new embedded NVIDIA Pascal GPUs to enable real-time implementations with larger images and a higher number of disparity levels, and more complex algorithms that provide better estimation results. In this sense, we are going to include post-filtering steps such as Left-Right Consistency Check, subpixel calculation, and adaptive P2, which are well-known methods of increasing accuracy.

Chapter 3

Slanted Stixels

3.1 Introduction

Autonomous vehicles, advanced driver assistance systems, robots and other intelligent devices need to understand their environment. For this purpose, both geometric (distance) and semantic (classification) sources of information are useful. We want to represent these inputs in a very compact model and compute them in real-time to serve as a building block of higher-level modules, such as localization and planning.

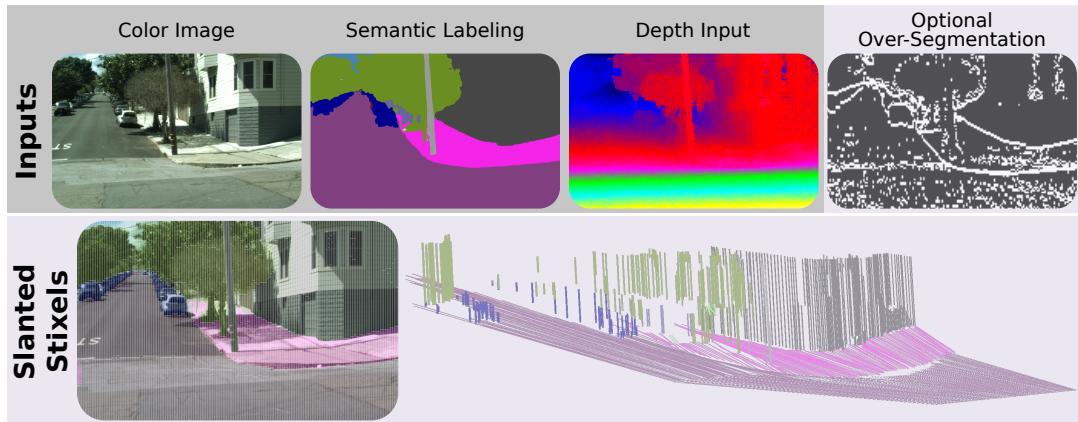
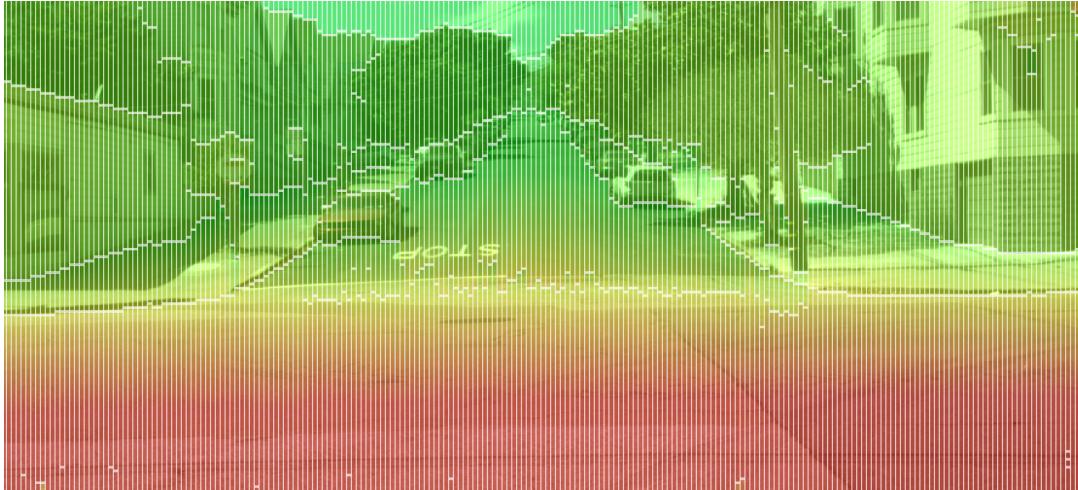


FIGURE 3.1: The proposed approach: pixel-wise color, semantic and depth information serve as input to our Slanted Stixels model, which is a compact semantic representation of a 3D scene that accurately handles arbitrary scenarios such as San Francisco city. The optional over-segmentation in the top-right yields significant speed gains while nearly retaining the depth and semantic accuracy.

This success has led to increased interest in the model from the intelligent vehicles community over the past years. The Stixel world has been successfully used for representing traffic scenes, as introduced in [44]. It has shown its potential particularly in the Bertha-Benz drive [63], where it has been successfully applied for visual scene understanding in autonomous driving. This success has led to increased interest in the model from the intelligent vehicles community over the past years [5, 8, 10, 12, 26, 28, 31, 35, 51].

The Stixel world defines a compact medium-level representation of dense 3D disparity data obtained from stereo vision using rectangles, the so called *Stixels*, as elements. Stixels are classified either as *ground-like* planes, *upright objects* or *sky*, which are important geometric elements found in man-made environments. This representation transforms millions of disparity pixels to hundreds or thousands of Stixels. At the same time, most scene structures, such as free space and obstacles, which are relevant for autonomous driving tasks, are adequately represented.



(A) Disparity representation of Stixels. The coloring encodes the distance from close (red) to far (green)



(B) Semantic representation of Stixels. The coloring encodes the semantic class following [11]

FIGURE 3.2: Scene representation obtained by our method of a challenging street environment with a slanted road. Both geometric (top) and semantic (bottom) representations are shown.

The idea behind the Stixel model is that planar surfaces are dominant in man-made environments and they can be modeled using this assumption. Scene structure found in urban environments can be modeled with certain constraints, *e.g.* the sky is above the horizon line and objects usually lie on the ground. Generally, the geometric constraints of a scene are tied to the vertical direction. Hence, the environment can be modeled as a column-wise segmentation of the image with a 3D stick-like shape, *i.e.* a set of Stixels, *c.f.* fig. 3.1. The segmentation of the image is estimated by solving a column-wise energy minimization problem, taking depth and semantic cues as inputs as well as *a priori* information that is used to regularize the solution *c.f.* fig. 3.1.

The Stixel model has been successfully used for automotive vision applications either to decrease parsing time, increase accuracy or both. We can find examples of works using the Stixel representation in different topics such as object recognition [6, 36], building a grid map over time [39] and for autonomous driving [63]. Specifically, for motion planning in the context of autonomous driving, the Stixel model has been used *c.f.* [63, 64] to model the geometric constraints of a given scene.

We propose a new depth model that is able to accurately represent arbitrary kinds of slanted objects and non-flat roads. The improved Stixel representation outperforms the original Stixel model in scenarios with non-flat roads, while keeping the same accuracy on flat road scenes. The induced extra computational complexity is reduced by incorporating an over-segmentation strategy that can be applied to any Stixel model proposed so far. An earlier version of our work [28] proposed a simple over-segmentation strategy that provided faster execution at the expense of decreasing the accuracy of the model. This chapter introduces a novel over-segmentation approach based on a Fully Convolutional Network (FCN) that outperforms the previous strategy, and achieves similar speedup results but retaining most of the accuracy of the original version. An overview of our method is shown in fig. 3.1.

Our main contributions are: (1) a novel depth model to accurately represent arbitrary kinds of slanted surfaces into the Stixel representation; (2) a novel over-segmentation prior designed to reduce the run-time of the method; (3) an effective over-segmentation strategy based on a shallow Fully Convolutional Network; (4) a new synthetic dataset with non-flat roads that includes pixel-level semantic and depth ground-truth, which is publicly available¹; and (5) an in-depth evaluation in terms of run-time as well as semantic and depth accuracy carried out on this novel dataset and several real-world benchmarks. Compared to the existing state-of-the-art approaches, our method substantially improves the depth accuracy in non-flat road scenarios.

3.2 Related Work

Our proposed method introduces a novel Stixel-based scene representation that is able to account for non-flat roads, *c.f.* fig. 3.2. We also devise an approximation to reduce the computational complexity of the underlying Dynamic Programming algorithm.

First, we will comment on works proposing different road scene models. Occupancy grid maps are models used to represent the surroundings of the vehicle [13, 39, 40, 55]. Typically, a grid in bird's eye perspective is defined and used to detect occupied grid cells and then, from this information, to extract the obstacles, drivable area, and unobservable areas from range data. These grids and the Stixel world both represent the 2D image in terms of column-wise stripes allowing to capture the camera data in a polar fashion. Also, the Stixel data model is similar to the forward step usually found in occupancy grid maps [12]. However, the Stixel inference method in the image domain presents important differences compared to classical grid-based approaches.

Our work builds upon the proposal from [51]: they use semantic cues in addition to depth to extract a Stixel representation, which is able to provide a rich yet compact representation of the traffic scene. However, their model assumes a constant road slant and is therefore limited to flat road scenarios. In contrast, our proposal overcomes this drawback by incorporating a novel plane model together with effective priors on the plane parameters.

Our proposal of using Stixels cuts is related to [10]: they use fast object detectors for different object classes, *e.g.* Viola-Jones cascade detector [56], to produce top and bottom Stixel cuts that are used as prior information, which is then integrated into the Stixel algorithm. They prove that using object-level knowledge provides significant accuracy improvements. Instead, we leverage semantic information as

¹<http://synthia-dataset.net>

pixel-level knowledge in our model for the same purpose of improving accuracy. Semantic segmentation identifies the objects and other elements of the image, *e.g.* walls or sidewalks, providing pixel-level information, instead of boxes around the objects. Also, semantic segmentation requires a single predictor, while the method proposed by [10] needs a detector trained for each object class. In contrast, we define a Stixel cut prior to generate an over-segmentation of the optimal Stixel cuts in order to speed up the execution of the algorithm.

There are some methods [5, 31, 35], that represent simplified scene models with a single Stixel per column. The advantage of these approaches is that the computational complexity of the underlying algorithms is linear, but they cannot represent some complex scenarios found in the real world, *e.g.* a pedestrian and a building in the same column.

Recent work by [8] uses edge-based disparity maps to compute Stixels. Their method is fast but they show that it gives inferior accuracy compared to the original Stixel model [45].

[35] firstly introduced the use of an FCN in Stixel-based methods. A single RGB image feeds the FCN to estimate the bottom of the first non-road Stixel, *i.e.* closest obstacle. We use an FCN for a entirely different objective: to extract a Stixel cut over-segmentation that accelerates the execution of the algorithm. Moreover, the input of our FCN is a disparity map obtained from a stereo camera.

Finally, there are some works proposing fast implementations for Stixel computation. The FPGA implementation from [39] runs at 25 Hz with a Stixel width of 5 pixels, but the authors do not indicate the image resolution. [26] present a GPU-accelerated implementation that runs at 26 Hz for a Stixel width of 5 pixels and image resolution of 1024×440 pixels, computed using a Semi-Global Matching (SGM) [29] stereo algorithm. We propose a novel approximation that accelerates the computation by reducing the algorithmic complexity. Accordingly, our proposal could benefit from the aforementioned FPGA- or GPU-accelerated implementations.

3.3 Stixel Model

The Stixel world is a compressed representation of a 3D scene that preserves its relevant structure. Since the structure in street environments is dominant in the vertical domain, the Stixel world leverages this idea to model a scene without taking into account the horizontal neighborhood. This assumption leads to an efficient inference method and also allows the inference to be performed on all columns in parallel.

The Stixel world is defined as a segmentation of image columns into stick-like super-pixels with class labels and a 3D planar depth model *c.f.* fig. 3.3. We consider three structural classes: *object*, *ground* and *sky*. These classes have properties that are derived from an underlying 3D model: for *object* Stixels the distance is roughly constant and usually lie on the ground, for *sky* Stixels the distance is infinite and for *ground* Stixels we favor planes with accordance to the expected ground.

The Stixel world has several properties that are useful for higher-level processing stages: (1) it is a medium-level scene representation that significantly reduces the quantity of elements, *e.g.* from millions of pixels to hundreds of Stixels, while keeping an abstract representation of physical extent, depth and semantics; (2) the representation is based upon a street model; (3) the representation is not high-level because an object is represented by more than one Stixel horizontally and it can be split in more than one Stixel vertically too, *e.g.* occlusions and slanted objects such as cars viewed from behind.

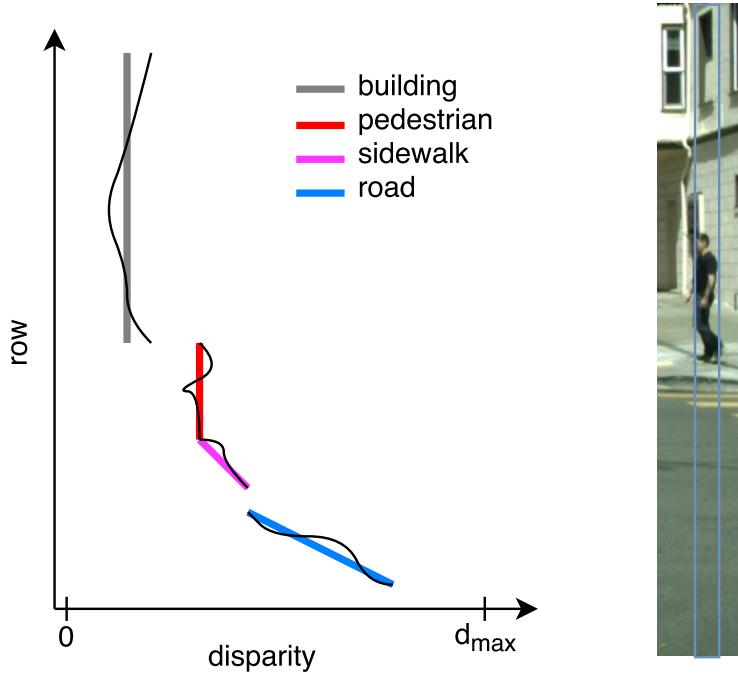


FIGURE 3.3: Example of input disparity measurements (black lines) and output Stixels encoded with semantic colors (colored lines) for a typical scene column (right). Adapted from [12].

The joint Stixel segmentation and labeling problem is carried out via optimization of the column-wise posterior distribution $P(S_i | M_i)$ defined over a Stixel segmentation S_i given all measurements M_i from that particular image column. In the following, we drop the column indexes for ease of notation. We obtain Stixel width > 1 as illustrated *e.g.* in fig. 3.1 by down-sampling of the inputs, this width is fixed and is chosen to reduce the computational complexity during inference, however heavy down-sampling leads to degradation in accuracy [12].

A Stixel column segmentation consists of an arbitrary number N of Stixels S_i , each representing four random variables: the Stixel extent via bottom V_i^b and top V_i^t row, as well as its class C_i and geometric depth model G_i . Thereby, the number of Stixels itself is a random variable that is optimized jointly during inference. To this end, the posterior probability is defined by means of the unnormalized prior and likelihood distributions

$$P(S | M) = \frac{1}{Z} \tilde{P}(M | S) \tilde{P}(S) \quad (3.1)$$

where Z is the normalizing partition function. Transformed to log-likelihoods via

$$P(S = s | M = m) = -\log(e^{-E(s, m)}) \quad (3.2)$$

where $E(\cdot)$ is the energy function, $E_{data}(\cdot)$ is the **likelihood** term and $E_{prior}(\cdot)$ is the **prior** term.

$$E(s, m) = E_{data}(s, m) + E_{prior}(s) \quad (3.3)$$

3.3.1 Data term

The **likelihood** term $E_{data}(\cdot)$ thereby rates how well the measurements \mathbf{m}_v at pixel v fit to the overlapping Stixel \mathbf{s}_i

$$\begin{aligned} E_{data}(\mathbf{s}, \mathbf{m}) &= \sum_{i=1}^N E_{stixel}(\mathbf{s}_i, \mathbf{m}) \\ &= \sum_{i=1}^N \sum_{v=v_i^b}^{v_i^t} E_{pixel}(\mathbf{s}_i, \mathbf{m}_v) . \end{aligned} \quad (3.4)$$

This pixel-wise energy is further split in a semantic and a depth term

$$E_{pixel}(\mathbf{s}_i, \mathbf{m}_v) = E_{disp}(\mathbf{s}_i, d_v) + w_l \cdot E_{sem}(\mathbf{s}_i, l_v) . \quad (3.5)$$

The parameter w_l controls the influence of the semantic data term. The input is provided by an FCN that delivers normalized semantic scores $l_v(c_i)$ with $\sum_{c_i} l_v(c_i) = 1$ for all classes c_i at pixels v . The semantic energy favors semantic classes of the Stixel that fit to the observed pixel-level semantic input [51]. The semantic likelihood term is

$$E_{sem}(\mathbf{s}_i, l_v) = -\log(l_v(c_i)) . \quad (3.6)$$

The depth model is designed to represent the different characteristics of the different geometric classes, *i.e.* *object*, *ground* and *sky* Stixels. Furthermore, the model enforces multiple stacked Stixels in cases of objects with the same class but different depths.

Our depth input is a dense disparity map, each pixel is assigned a disparity value or is masked as invalid *i.e.* $d_v \in \{0 \dots d_{max}, d_{invalid}\}$. The depth term is defined by means of a probabilistic and generative sensor model $P_v(\cdot)$ that considers the accordance of the depth measurement d_v at row v to the Stixel \mathbf{s}_i

$$E_{disp}(\mathbf{s}_i, d_v) = -\log(P_v(D_v = d_v | \mathbf{s}_i = \mathbf{s}_i)) . \quad (3.7)$$

Invalid d_{inv} disparity measurements have to be handled, therefore, a prior probability of a valid disparity value is defined as p_{val}

$$P_v(D_v | \mathbf{s}_i) = \begin{cases} p_{val} P_{v,val}(D_v | \mathbf{s}_i) & \text{if } d_v \neq d_{inv} \\ (1 - p_{val}) & \text{otherwise} \end{cases} \quad (3.8)$$

where $P_{v,val}(D_v | \mathbf{s}_i)$ is the measurement model of valid disparities only. It is comprised of a constant outlier probability p_{out} and a Gaussian sensor noise model for valid measurements with confidence c_v

$$P_{v,val}(D_v | \mathbf{s}_i) = \frac{p_{out}}{Z_U} + \frac{1 - p_{out}}{Z_G(\mathbf{s}_i)} e^{-\left(\frac{c_v(d_v - \mu(\mathbf{s}_i, v))}{\sigma(\mathbf{s}_i)}\right)^2} \quad (3.9)$$

that is centered at the expected disparity $\mu(\mathbf{s}_i, v)$ given by the depth model of the Stixel, where Z_U and $Z_G(\mathbf{s}_i)$ normalize the distributions. Similarly to [45], we use the confidence of the depth estimates c_v to influence the shape of the distribution. The Gaussian models the typical disparity noise and the uniform distribution makes the model more robust to outliers, which is weighted by p_{out} . The standard deviation $\sigma(\mathbf{s}_i)$ models the noise of the stereo matching algorithm and depends on the class c_i .

New depth model

The depth model defines the 3D outline of a Stixel using very few parameters per Stixel and reflects our assumptions on the surrounding scene. Both, data term (*c.f.* eq. (3.9)) and priors (*c.f.* section 3.3.2) have a significant impact on the inferred depth model. In previous formulations, the three different geometric classes were designed using restrictive constant height (ground Stixels) and constant depth (object and sky Stixels), assumptions per Stixel, *e.g.* for object Stixels: $\mu(s_i, v) = \text{constant}$.

This chapter introduces a new plane depth model that relaxes the previous assumptions in favor of a more accurate depth representation. The new model is formulated such that it nicely interacts with this well founded and experimentally validated depth sensor model. To this end, we formulate the depth model $\mu(s_i, v)$ using two random variables defining a plane in the disparity space that evaluates to the disparity in row v via

$$\mu(s_i, v) = b_i \cdot v + a_i . \quad (3.10)$$

Note that we assume narrow Stixels and thus can neglect one plane parameter, *i.e.* the roll.

This model is a generalization of the previous class-specific depth models used in previous works, allowing for a more flexible representation of the scene because of the extra free parameter *c.f.* fig. 3.4. The way of modeling the different Stixel classes *i.e. object, ground and sky* is through priors, as explained in section 3.3.2. Also, to completely understand the details about the inference, we suggest to read section 3.3.3.

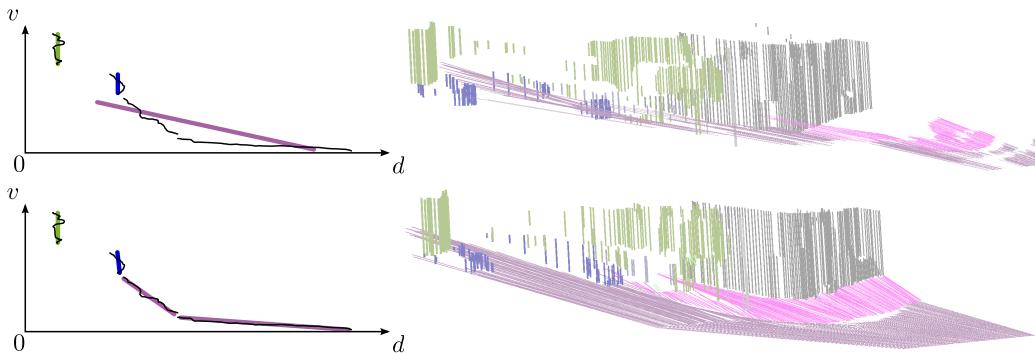


FIGURE 3.4: Comparison of original [51] (top) and our Slanted Stixels (bottom): due to the fixed slant in the original formulation, the road surface is not well represented as illustrated on the top-left figure. The novel model is capable of reconstructing the whole scene accurately.

3.3.2 Prior term

The **prior** captures knowledge about the segmentation independent from measurements, in this section we define the priors used for this model, they are based on [12]. The Markov property is used so that the prior reduces to pair-wise relations between subsequent Stixels. Accordingly, the prior is computed as

$$E_{\text{prior}}(\mathbf{s}) = \sum_{i=2}^N E_{\text{pair}}(\mathbf{s}_i, \mathbf{s}_{i-1}) + E_{\text{first}}(\mathbf{s}_1) . \quad (3.11)$$

In the next sections, where different priors are introduced, $E_{pair}(\mathbf{s}_i, \mathbf{s}_{i-1})$ is the summation of all these priors. However, $E_{first}(\mathbf{s}_1)$ does not include pairwise terms, *i.e.*

$$\begin{aligned} E_{first}(\mathbf{s}_1) = & E_{mc}(\mathbf{s}_1) + E_{segfirst}(\mathbf{s}_1) + E_{seglast}(\mathbf{s}_1) \\ & + E_{top \geq bottom}(\mathbf{s}_1) + E_{plane}(\mathbf{s}_1) \end{aligned} \quad (3.12)$$

Model complexity prior

A model complexity term favors solutions composed of fewer Stixels and thus invokes costs for each Stixel in the column segmentation S :

$$E_{mc}(\mathbf{s}_i) = C_{mc}. \quad (3.13)$$

There is a trade-off between compactness and accuracy. A high C_{mc} parameter would lead to a very compact segmentation *i.e.* few Stixels. However, a representation with few Stixels is more likely to have lower accuracy, *e.g.* a solution comprised of one Stixel the size of the whole column would result in a huge disparity and semantic error.

Segmentation priors

The model has to enforce that all pixels are assigned to exactly one Stixel, *i.e.* non-overlapping Stixels, Stixels extend over all the column and Stixels are connected. Therefore, the first priors are defined to comply with the following rules: The first Stixel must begin in row 1 and the last Stixel must end in row h , *i.e.*

$$E_{segfirst}(\mathbf{s}_i) = \begin{cases} \infty & \text{if } v_i^b \neq 1, i = 1 \\ 0 & \text{otherwise} \end{cases} \quad (3.14)$$

$$E_{seglast}(\mathbf{s}_i) = \begin{cases} \infty & \text{if } v_i^t \neq h, i = n \\ 0 & \text{otherwise} \end{cases}. \quad (3.15)$$

Furthermore, every Stixel must be connected to the next one and the Stixel top row must be greater than the bottom row, *i.e.*

$$E_{connection}(\mathbf{s}_i, \mathbf{s}_{i-1}) = \begin{cases} 0 & \text{if } v_i^b = v_{i-1}^t + 1 \\ \infty & \text{otherwise} \end{cases} \quad (3.16)$$

$$E_{top \geq bottom}(\mathbf{s}_i) = \begin{cases} 0 & \text{if } v_i^b \leq v_i^t \\ \infty & \text{otherwise} \end{cases}. \quad (3.17)$$

Structural priors

The gravity prior penalizes a flying object *i.e.* an *object* Stixel not lying on top of the previous *ground* Stixel,

$$E_{gravity}(\mathbf{s}_i, \mathbf{s}_{i-1}) = \begin{cases} \alpha_{gravity}^- + \beta_{gravity}^- \Delta_g & \text{if } \Delta_g < 0 \\ \alpha_{gravity}^+ + \beta_{gravity}^+ \Delta_g & \text{if } \Delta_g > 0 \\ 0 & \text{otherwise} \end{cases} \quad (3.18)$$

where $\Delta_g = \mu_s(v_i^b, g_i) - \mu_s(v_{i-1}^t, g_{i-1})$ is the difference between the *object* Stixel disparity $\mu_s(v_i^b, g_i)$ at it's bottom pixel v_i^b and the disparity of the *ground* Stixel $\mu_s(v_{i-1}^t, g_{i-1})$ at the top row v_i^t . It only applies for s_i being an object and s_{i-1} being a ground Stixel.

The depth ordering prior penalizes a combination of two staggered *object* Stixels when the upper of the two is closer (in distance to the car) than the lower one.

$$E_{ord}(s_i, s_{i-1}) = \begin{cases} \alpha_{ord} + \beta_{ord}(g_i - g_{i-1}) & \text{if } g_i > g_{i-1} \\ 0 & \text{otherwise} \end{cases}. \quad (3.19)$$

A novel prior is introduced in this chapter: the ground gap prior penalizes two consecutive *ground* Stixels when the bottom disparity of the upper Stixel *i.e.* disparity at row v_i^b and the disparity of the lower Stixel at row v_i^b do not match.

$$E_{gap}(s_i, s_{i-1}) = \begin{cases} \alpha_{gap}^- + \beta_{gap}^-\Delta_{gap} & \text{if } \Delta_{gap} < 0 \\ \alpha_{gap}^+ + \beta_{gap}^+\Delta_{gap} & \text{if } \Delta_{gap} > 0 \\ 0 & \text{otherwise} \end{cases} \quad (3.20)$$

where $\Delta_{gap} = g_s(v_i^b, g_i) - g_s(v_{i-1}^b, g_{i-1})$. These structural priors do not enforce their assumptions. Instead, they penalize unusual combinations, *e.g.* a flying object (gravity prior), traffic signs (ordering prior).

Transition priors

These priors define the knowledge regarding the transition between a pair of Stixels.

$$E_{transition}(s_i, s_{i-1}) = \gamma_{c_i, c_{i-1}} \quad (3.21)$$

where $\gamma_{c_i, c_{i-1}}$ is the transition cost between previous Stixel class c_{i-1} to current Stixel class c_i . This is defined via a two-dimensional transition matrix for all combinations of classes $\gamma_{c_i, c_{i-1}}$. Only first order relations are modeled in order to infer efficiently.

Plane prior

In this chapter, we propose a new additional prior term that uses the specific properties of the three geometric classes. We expect the two random variables A, B representing the plane parameters of a Stixel to be Gaussian distributed, *i.e.*

$$E_{plane}(s_i) = \left(\frac{a_i - \mu_{c_i}^a}{\sigma_{c_i}^a} \right)^2 + \left(\frac{b_i - \mu_{c_i}^b}{\sigma_{c_i}^b} \right)^2 - \log(Z) . \quad (3.22)$$

This prior favors planes in accordance to the expected 3D layout corresponding to the geometric class. For instance, *object* Stixels are expected to have an approximately constant disparity, *i.e.* $\mu_{object}^b = 0$. The expected road slant μ_{ground}^b can be set using prior knowledge or a preceding road surface detection. For *sky* Stixels we expect infinite distance *i.e.* 0 disparity, therefore, we set $\mu_{sky}^a = \mu_{sky}^b = 0$.

The standard deviations $\sigma_{c_i}^a$ and $\sigma_{c_i}^b$ are used in order to enforce the assumptions for each Stixel class, *i.e.* the more confident we are that *object* Stixels have constant distance, the closer to 0 we would set σ_{object}^b . The same applies for *ground* Stixels: if we know the road is not slanted, we can rely on the expected previous road model

and set $\sigma_{ground}^b \rightarrow 0$. For *sky* Stixels, it does not make sense to have a disparity different to 0. Therefore, we set $\sigma_{sky}^a \rightarrow 0$ and $\sigma_{sky}^b \rightarrow 0$.

Note that the novel formulation is a strict generalization of the original method, since they are equivalent, *e.g.* if the slant is fixed, *i.e.* $\sigma_{object}^b \rightarrow 0$, $\mu_{object}^b = 0$.

3.3.3 Inference

The sophisticated energy function defined in section 3.3 is optimized via Dynamic Programming as in [44]. However, we must also optimize jointly for the novel depth model. When optimizing for the plane parameters a_i, b_i of a certain Stixel s_i , it becomes apparent that all other optimization parameters are independent of the actual choice of the plane parameters. We can thus simplify

$$\operatorname{argmin}_{a_i, b_i} E(s, m) = \operatorname{argmin}_{a_i, b_i} E_{stixel}(s_i, m) + E_{plane}(s_i) . \quad (3.23)$$

Thus, we minimize the global energy function with respect to the plane parameters of all Stixels and all geometric classes independently. We can find an optimal solution of the resulting weighted least squares problem in closed form. However, we still need to compare the Stixel measurements to our new plane depth model. Therefore, the complexity added to the original formulation is another quadratic term in the image height.

3.3.4 Stixel Cut Prior

The Stixel inference process described so far requires the estimation of the cost for each possible Stixel in an image. However, many Stixels can be trivially discarded, *e.g.* in image regions with homogeneous depth and semantic input, making it possible to avoid the computation steps associated to the calculation of these.

We propose a novel prior that exploits hypothesis generation to significantly reduce the computational burden of the inference task. To this end, we formulate a new prior similar to [10]; however, instead of Stixel bottom and top probabilities, we incorporate generic likelihoods for pixels being the cut between two Stixels.

We leverage this additional information adding a novel prior term for a Stixel s_i

$$E_{cut}(s_i) = -\log(c_{v_i}(cut)) \quad (3.24)$$

where $c_{v_i}(cut)$ is the confidence for a cut at v_i , thus $c_{v_i}(cut) = 0$ implies that there is no cut between two Stixels at row v .

As described in [42], we can design a recursive definition of the optimization problem in order to solve the problem using a Dynamic Programming scheme. In order to simplify our description, we use a special notation to refer to Stixels: $ob_b^t = \{v^b, v^t, object\}$. Similarly, OB^k is defined as the minimum aggregated cost of the best segmentation from position 0 to k . The Stixel at the end of the segmentation associated with each minimum cost is denoted as ob^k . We next show a recursive

definition of the problem:

$$OB^k = \min \left\{ \begin{array}{ll} E_{data}(ob_0^k) & +E_{prior}(ob_0^k) \\ E_{data}(ob_x^k) & +E_{prior}(ob_x^k, ob^{x-1}) \\ & +OB^{x-1} \forall x \in cuts, x \leq k \\ E_{data}(ob_x^k) & +E_{prior}(ob_x^k, gr^{x-1}) \\ & +GR^{x-1} \forall x \in cuts, x \leq k \\ E_{data}(ob_x^k) & +E_{prior}(ob_x^k, sk^{x-1}) \\ & +SK^{x-1} \forall x \in cuts, x \leq k \end{array} \right. . \quad (3.25)$$

We only show the case for *object* Stixels, but the other cases are solved similarly. Also, GR^k and SK^k stand for *ground* and *sky* respectively. The base case problem, *i.e.* segmenting a column of the single pixel at the bottom, is defined: $OB^0 = E_{data}(ob_0^0) + E_{prior}(ob_0^0)$. Our method trusts that all the optimal cuts will be included in our over-segmentation (*cuts* in eq. (3.25)), therefore, only those positions are checked as Stixel bottom and top. This reduces the complexity of the Stixel estimation problem for a single column to $\mathcal{O}(h' \times h')$, where h' is the number of over-segmentation cuts computed for this column, h is image height and $h' \ll h$.

The computational complexity reduction becomes apparent in fig. 3.5. As stated in [12], the inference problem can be interpreted as finding the shortest path in a directed acyclic graph. Our approach prunes all the vertices associated with the Stixel's top row not included according to the Stixel cut prior, *c.f.* fig. 3.5b.

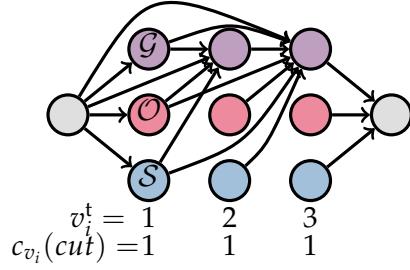
3.4 Generation of the Stixel cut prior

The previous section explained how to use a Stixel cut prior to reduce the computational complexity of the Stixel inference. The idea is that many Stixel cuts could be trivially discarded, *e.g.* in image regions with homogeneous depth and semantic input. We can save a lot of computation by not processing those unlikely Stixel cuts. The goal is to devise a fast method to generate an over-segmentation of the optimal Stixel cuts. And, if those optimal cuts are included in the generated hypothesis, then the Stixel algorithm will provide the same output as in the original case, but doing much fewer computation steps.

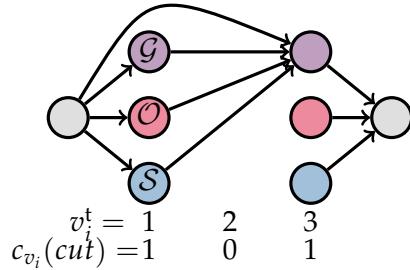
We propose two methods to generate Stixel cuts. The first method is a simple strategy that uses some mathematical concepts to identify points of interest *c.f.* section 3.4.1. It is a very fast approach, but misses some of the optimal Stixel cuts and, therefore, the final accuracy of the Stixel inference is reduced. The second method uses a shallow Fully Convolutional Network (FCN) that is trained on the disparity map to infer likely Stixel cuts *c.f.* section 3.4.2. This strategy is also very fast, since the FCN is small, and is able to provide almost all of the optimal Stixel cuts. For both methods, we leverage semantic segmentation information by including the edges of the semantic image into the set of the generated Stixel cuts.

3.4.1 Time Series Compression

The first method to generate Stixel cuts is based on the work of [31], and has linear time complexity and linear memory requirements. In their work, each column of the disparity map is treated independently as a time series, *i.e.* a signal with measurements on equal intervals of time. They first perform an *extreme points detection*



(A) Stixel graph representation



(B) Pruned graph using Stixel cut prior

FIGURE 3.5: Stixel inference illustrated as shortest path problem on a directed acyclic graph: the Stixel segmentation is computed by finding the shortest path from the source (left gray node) to the sink (right gray node). The vertices represent Stixels with colors encoding their geometric class, *i.e.* ground, object and sky. Only the incoming edges of ground nodes are shown for simplicity. Adapted from [12].

step that generates a list of possible Stixel cuts, and then apply subsequent filters to this list in order to generate the final Stixel segmentation. As we want to obtain an over-segmentation containing all the optimal Stixel cuts, we only use the first step of their proposal.

The detection of extreme points is based on techniques for time series compression [18]. A time series can be compressed by selecting local extreme points, *i.e.* maxima and minima of a function within a range. The assumption is that local extreme points are enough to find the important parts of the signal, and the rest would be unimportant points or noise.

In [31] only left and right extrema are selected, while other kinds of extrema are discarded. Given a time series $\{t_1, t_2, \dots, t_i, \dots, t_{n-1}, t_n\}$ and point t_i with $1 < i < n$, the definition of left and right minimum is as follows (the definition of maxima is symmetric):

- t_i is left minimum if $t_i < t_{i-1}$ and there is t_j such that $j > i$ and $t_i = \dots = t_j < t_{j+1}$.
- t_i is right minimum if $t_i < t_{i+1}$ and there is t_j such that $j < i$ and $t_{j-1} > t_j = \dots = t_i$.

Similarly, we generate Stixel cuts by finding left and right extrema and the first and last points of the sequence of pixels in the column. The example in fig. 3.6 illustrates the method. The predicted Stixel cuts are indicated in red color. In the

example the vertical resolution is reduced around 3.3 times, which implies reduced computational work for the Stixel inference task.



FIGURE 3.6: Generated Stixel cuts (highlighted in red) using the left and right extrema as defined in [31], and also cuts generated from semantic segmentation. Stixel cut density is 30%, equivalent to a $3.3\times$ reduction in vertical resolution.

3.4.2 FCN-based method

We propose a novel shallow deep neural network *c.f.* fig. 3.8 that generates a set of promising Stixel cuts from depth images *c.f.* fig. 3.7. We follow the proposal in [32]: we use disparities instead of depth. We have experimentally found that adding the RGB image to the input of the neural network does not improve the accuracy of the method, compared to the simpler and faster strategy of directly adding the edges of the semantic image into the set of the generated Stixel cuts.



FIGURE 3.7: Generated Stixel cuts (highlighted in red) for the FCN-based method. Stixel cut density is 31.5%, equivalent to a $3.2\times$ reduction in vertical resolution.

We design the network to provide an over-segmentation of the optimal Stixel cuts that should be significantly smaller than the total number of potential Stixel cuts (which is the height of the image). Also, the computational work required for the network inference must be small, ideally similar to the Time Series method proposed

in section 3.4.1. In the remainder of this section, we will first discuss the proposed network architecture, and then describe the data and training strategy.

Network architecture

Our proposal is based on the architecture described by [50]. They present a multi-modal FCN designed for semantic segmentation with a mid-level fusion architecture that exploits complementary input cues, *i.e.* RGB and disparity data. Their design includes the Network in Network (NiN) method proposed by [37]. Our proposal inherits the network branch that processes the disparity data and discards the branch on the RGB data, which is described in detail in fig. 3.8. The proposed FCN is a very shallow network with three consecutive NiNs, and a final deconvolution that recovers the desired resolution of the Stixel cuts. The output of the FCN is a binary image indicating whether or not there is a Stixel cut for that pixel.

Training data

We trained the proposed FCN using disparity maps generated from images in the Synthia synthetic dataset [46] and from images in a real-data sequence (6757 images) recorded in San Francisco, *c.f.* fig. 3.9. In both cases, the disparity maps are generated from the left and right RGB images using a stereo matching algorithm [29]. This is the expected situation in a realistic scenario, where the SGM algorithm in the perception pipeline generates the disparity map and feeds the FCN that produces the Stixel over-segmentation.

The ground-truth for the training data (the expected Stixel cuts) is generated as a combination of methods. In the case of the annotated synthetic dataset, which contains both pixel-level semantic and instance-level annotations, the ground-truth includes, as desired Stixel cuts, the boundaries of the instances and the semantic classes in the image (as in [12]). Finally, the Stixel cuts associated to disparity changes are obtained by running the Stixel inference method. In the real-data sequence, we only perform this last step because we lack ground-truth.

As discussed previously *c.f.* section 3.3.2, the definition of the parameters of the Stixel model represent a trade-off between compactness and accuracy. Since we need an over-segmentation of the optimal Stixel cuts, we adjust the parameters of the model to be conservative and to favor accuracy versus compactness.

The idea of using the Stixel model as a way to train a fast and simple neural network to approximate the optimal Stixel segmentation is inspired by model distillation techniques [7]. The comparatively slow Dynamic Programming method to solve the probabilistic model is used to transfer the knowledge inside the complex model to a fast and compact FCN that approximates the optimal Stixel cuts.

Training strategies

Since our problem is to classify each pixel of our input disparity map as *cut* or *not-cut*, we use cross-entropy as the loss function that must be minimized. The distribution of *cut/not-cut* is strongly biased in our input and, accordingly, we introduce a class-balancing weight in the loss function, similarly to [57]. These weights cause the FCN to generate wider edges *c.f.* fig. 3.7. This is useful, since the FCN roughly detects the Stixel cut positions, and the precise detection is left to the Stixel inference.

We set the learning rate to 10^{-8} and the batch size to five: four of those inputs are Synthia images and one of them is a real-data image. The missing disparities are

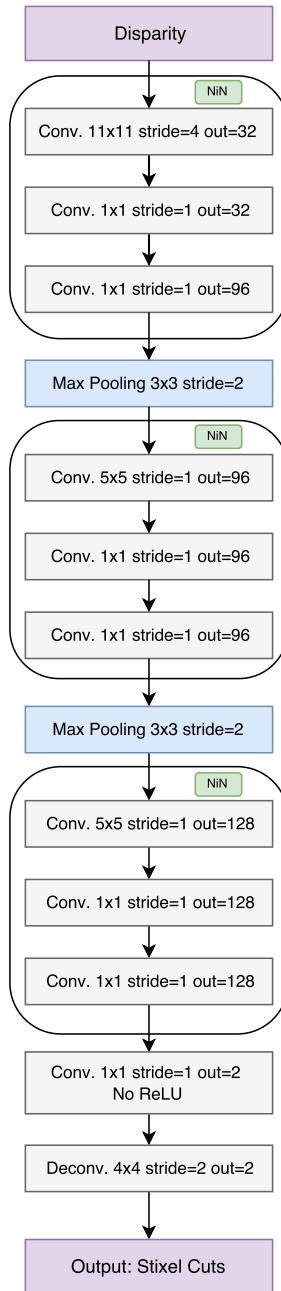


FIGURE 3.8: Definition of the proposed Fully Convolutional Network for generating Stixel cuts.



FIGURE 3.9: Sample image from the real-data sequence used for Stixel cut generation. Stixel cut ground-truth is highlighted in red.

encoded as -1 . Input normalization is done by subtracting the mean value from the disparity map. We initialize the FCN with the weights used in [50], since semantic segmentation is a similar problem.

3.5 Experiments

We will assess the accuracy and run-time of our method. We need to verify that our proposal improves the accuracy for scenes with non-flat roads, as well as maintaining the accuracy for flat road scenes. To accomplish this, we present synthetic and real-data datasets to evaluate our method in section 3.5.1. The experimental details such as inputs, metrics and baselines are presented in section 3.5.2. Our quantitative and qualitative results are presented in section 3.5.3.

3.5.1 Datasets

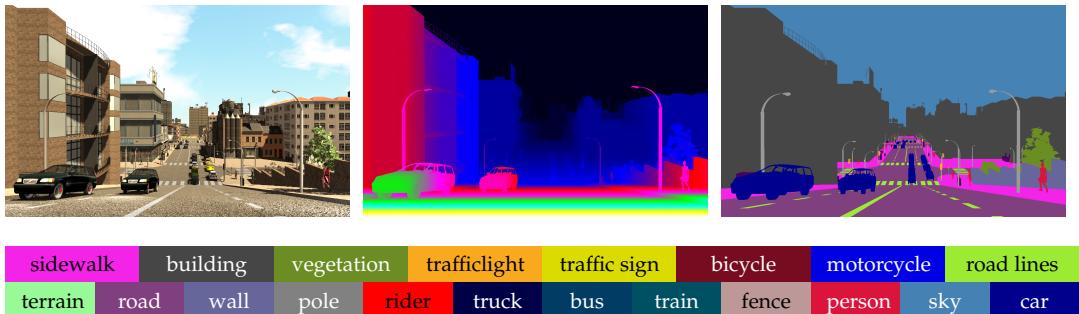


FIGURE 3.10: The SYNTHIA-SF Dataset. A sample frame (left) with its depth (center) and semantic labels (right).

We must evaluate accuracy for both geometric and semantic information. In order to do that, we use *Ladicky* [34], an annotated subset of *KITTI* [19], which contains both dense semantic labels and depth ground-truth. It is a set of 60 images with 0.5

mega-pixel resolution, we evaluate Stixel semantic and depth accuracy. We ignore the three rarest object classes (as suggested in [34]), which leaves us with 8 classes.

The training data used for our semantic segmentation FCN, is collected from publicly available semantic annotations on other parts of *KITTY* [23, 33, 52, 58, 62]. Our amalgamated training set contains 676 images, the labels are harmonized as suggested by [34]. The harmonization and data processing is consistent with previous work [51] to allow for fair comparison.

To further evaluate disparity accuracy, we use the training data of *KITTI 2015* [19] stereo challenge. The dataset contains 200 images with associated sparse disparity ground-truth generated by a laser scanner. No semantic segmentation ground-truth is available for this dataset.

To evaluate our semantic accuracy, we use *Cityscapes* [11], a complex dataset with dense semantic segmentation annotations of 19 classes on ~ 3000 images for training and 500 images for validation, we use this validation set for testing our method.

All the datasets we presented are collected in flat road environments. Then, they help us validate we are not decreasing accuracy with respect to previous approaches in this kind of scenes. However, we need to evaluate our hypothesis that our method does better in non-flat environments, therefore, we need a new dataset.

To that end, we introduce a new synthetic dataset inspired by [46]. We generated this dataset in order to evaluate our model; but it contains enough information to be useful in other related tasks, such as semantic and instance segmentation.

SYNTHIA-San Francisco (*SYNTHIA-SF*) contains 2224 images that we use to evaluate both semantic and disparity accuracy in non-flat road environments. This synthetic dataset was rendered from a virtual city with photo-realistic quality and provides precise pixel-level depth and semantic annotations for 19 labels *c.f.* fig. 3.10.

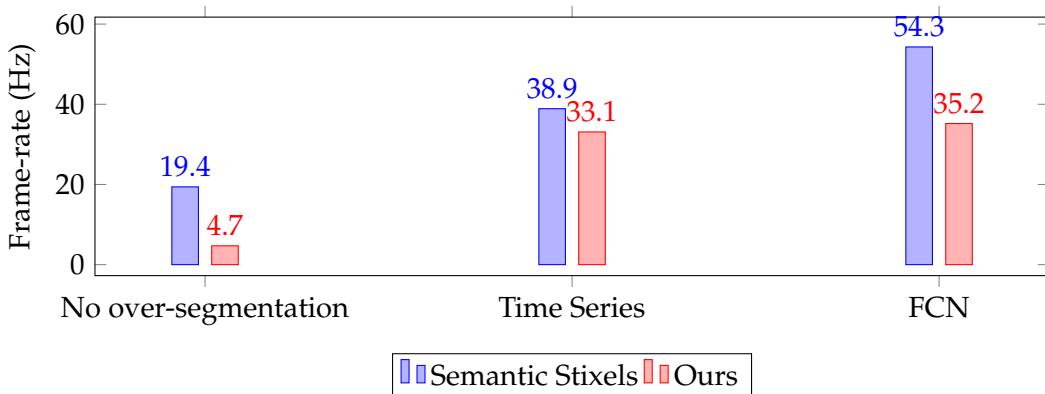


FIGURE 3.11: Frame-rate of our method (measuring only Stixel computation step and corresponding over-segmentation approach) compared to Semantic Stixels [51] for *SYNTHIA-SF* (image resolution of 1920×1080) on a multi-threaded CPU implementation (Intel i7-6800K) computed with a Stixel width of 8 pixels and same Stixel height. Two over-segmentation approaches are compared: Time Series *c.f.* section 3.4.1, FCN *c.f.* section 3.4.2.

3.5.2 Experiment details

Metrics

Our goal is to evaluate our method both in terms of semantic and depth accuracy using two metrics. The depth accuracy is generated as a rate of outliers of the disparity estimates, it is the standard metric used on the *KITTI* benchmark [19]. An outlier is

TABLE 3.1: We report frame-rate for each stage of our pipeline for a stereo pair of 1242×375 image resolution. OS stands for *Over-segmentation*. Stixel methods and SGM run-time is obtained using a CPU Intel i7-6800K. A NVidia Maxwell Titan X is used for obtaining *Semantic Segmentation* frame-rate. Note that Stixel frame-rate is variable when using an over-segmentation method, a representative run-time is provided. The total frame-rates are reported for the whole pipeline.

Stage	Frame-rate (Hz)
SGM	55
Semantic Segmentation	47.6
Our Stixels (OS: Time Series)	116
Our Stixels (OS: FCN)	130
Our Stixels (No OS)	61
Total (OS: Time Series)	20.92
Total (OS: FCN)	21.33
Total (No OS)	18

TABLE 3.2: Accuracy of our approach compared to Semantic Stixels [51], raw SGM and FCN. We provide results on four datasets: Ladicky [34], KITTI 15 [19], Cityscapes [11] and SYNTHIA-SF using metrics: Disparity Error (less is better) and Intersection over Union (more is better) *c.f.* section 3.5.1 and section 3.5.2. *Fast* (over-segmentation) methods are presented in section 3.4.1 and section 3.4.2. Significantly best results are highlighted in bold.

Metric	Dataset	Input		No over-segmentation		Fast: Time Series		Fast: FCN	
		SGM	FCN	Sem. Stixels	Ours	Sem. Stixels	Ours	Sem. Stixels	Ours
Disp Error (%)	Ladicky	16.66	-	17.38	16.84	17.60	17.01	17.44	16.84
	KITTI 15	11.01	-	11.05	11.21	11.9	11.9	11.21	11.24
	SYNTHIA-SF	11.06	-	29.33	12.99	30.60	14.20	31.12	14.19
IoU (%)	Ladicky	-	69.8	66.2	66.1	66.0	66.0	66.2	66.1
	Cityscapes	-	66.7	65.4	65.8	64.9	65.0	65.5	65.6
	SYNTHIA-SF	-	48.1	46.0	48.5	45.7	48.0	47.0	48.6

defined as a disparity with an absolute error larger than 3 pixels or a relative deviation larger than 5% with respect to the ground-truth. The metric used for semantic accuracy is the standard average Intersection-over-Union (IoU) over all classes [17]. To evaluate the complexity of the representation, we provide the number of Stixels per image. Finally, inference speed is shown as Frame-rate (Hz), this is useful to evaluate if we meet our real-time performance requirements. Execution times of *Stixels* and *SGM* are generated using a multi-threaded CPU implementation on standard off-the-shelf hardware: Intel i7-6800K. Frame-rate is also measured for the semantic segmentation neural network, the FCN is run using NVidia Maxwell Titan X GPU. The over-segmentation approach is included in the Stixel frame-rate. Note that the Stixel frame-rate is variable because of the over-segmentation method, *i.e.* it will depend on the number of Stixel cuts removed, therefore we provide a representative frame-rate. Similarly to [12], we compute SGM and semantic segmentation in parallel to maximize the throughput, however, we will have one frame delay.

TABLE 3.3: Number of Stixels (10^3) obtained by our methods compared to Semantic Stixels [51] and raw input (total number of pixels). We provide results on four datasets: Ladicky [34], KITTI 15 [19], Cityscapes [11] and SYNTHIA-SF *c.f.* section 3.5.1. *Fast* (oversegmentation) methods are presented in section 3.4.1 and section 3.4.2.

Dataset	Input SGM/FCN	No over-segmentation		Fast: Time Series		Fast: FCN	
		Sem. Stixels	Ours	Sem. Stixels	Ours	Sem. Stixels	Ours
Ladicky	454	0.6	0.6	0.6	0.6	0.6	0.6
KITTI 15	452	0.7	0.7	0.7	0.7	0.7	0.7
Cityscapes	2 k	1.4	1.5	1.3	1.4	1.4	1.5
SYNTHIA-SF	2 k	1.5	1.7	1.2	1.3	1.3	1.3

Baseline

Our baseline for comparison is Semantic Stixels [51], because they are the state-of-the-art in terms of Stixel accuracy. We provide accuracy of our new model, *c.f.* section 3.3. Finally, we evaluate the complexity of our Stixel cut approach section 3.3.4, we evaluate our two proposed over-segmentation methods presented in section 3.4.1 (labeled as *Time Series*) and section 3.4.2 (labeled as *FCN*).

Input

The inputs of our method are disparity maps via SGM [29] and semantic segmentation for each pixel computed by a CNN [38]. To allow fair comparison of results, we use the same neural network as [51] without retraining. For the same reason, we set Stixel width to 8 pixels with the same down-sampling in the vertical direction. All hyperparameters are extracted from [51].

We use the camera parameters obtained after calibration to set the expected values of μ_{ground}^a and μ_{ground}^b . For *object* Stixels, we set $\sigma_{\text{object}}^b \rightarrow 0, \mu_{\text{object}}^b = 0$ because the disparity is too noisy for the slanted object model. Finally, since *sky* Stixels can not have slanted surfaces, we set: $\mu_{\text{sky}}^a = 0, \mu_{\text{sky}}^b = 0, \sigma_{\text{sky}}^a \rightarrow 0, \sigma_{\text{sky}}^b \rightarrow 0$.

3.5.3 Results

We provide quantitative results of our method and baseliens are presented in tables 3.2 and 3.3 and fig. 3.11.

We can see that our method obtains comparable or slightly better results on all flat-road datasets *c.f.* compare *Semantic Stixels* to *Ours* for *Ladicky*, *KITTI 15* and *Cityscapes* datasets in table 3.2. With this results, we verify that our more flexible approach does not get worse results for these scenarios.

The second observation is that our model accurately models non-flat scenes in contrast to the original Stixel approach, substantially improving depth accuracy by more than 16% *c.f.* compare *Semantic Stixels* to *Ours* for the *SYNTHIA-SF* dataset in table 3.2. We experimentally show that this holds for real-data by providing a video of the Stixel 3D representation of a challenging non-flat scene as supplementary material. As a consequence of the joint semantic and depth inference of Stixels, semantic accuracy is also improved because of our new depth model.

Our *Fast: FCN* over-segmentation approach provides an accuracy almost equal to not using over-segmentation in all cases but one. Note that a perfect over-segmentation method would find the optimal cuts, then, it would provide the same accuracy of not

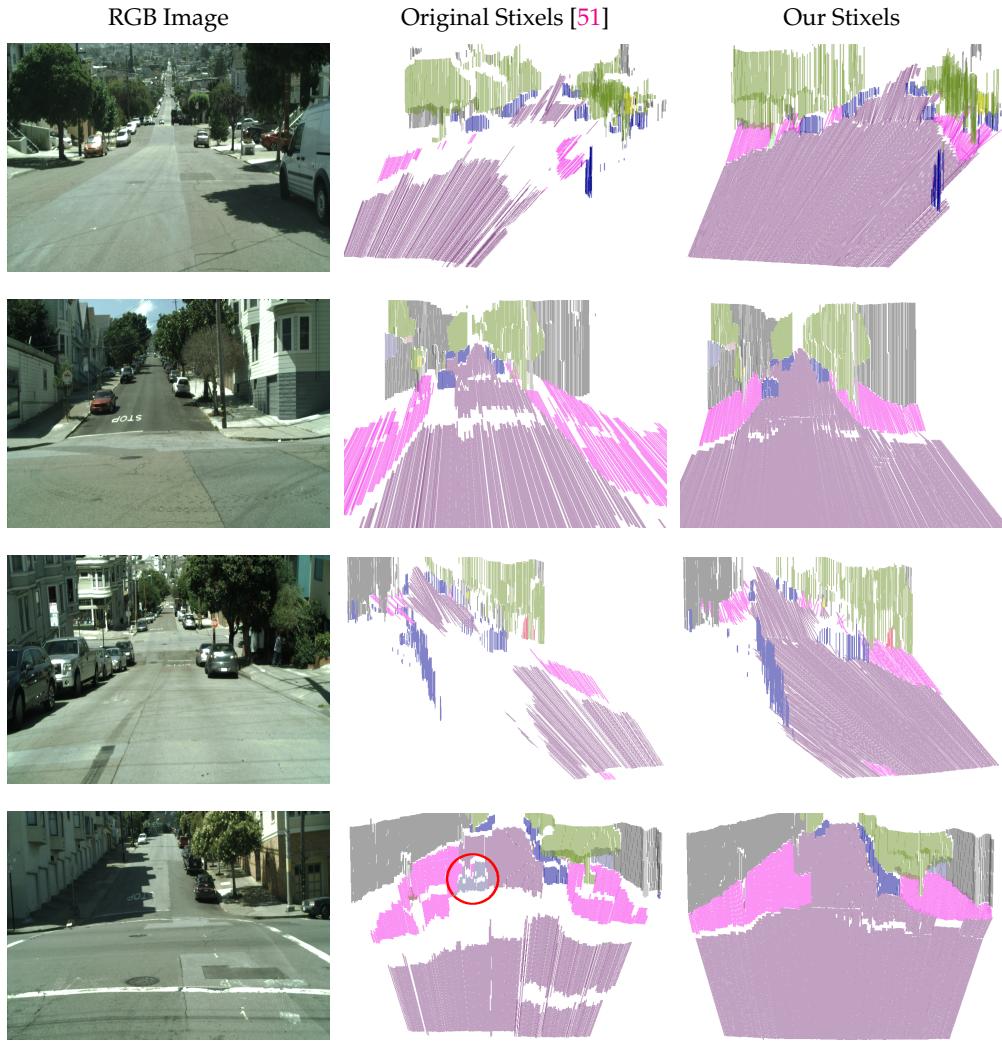


FIGURE 3.12: Some results on real data: with non-flat roads our model correctly models the scene, while retaining accuracy on objects. We show a failure case in the last row, our approach detects part of the road as sidewalk because of incorrect semantic input. As we highlight with a red circle, the original approach detects a wall, this could lead to an unnecessary emergency break.

using over-segmentation. This means that our *Fast: FCN* method is close to that hypothetical perfect approach. As expected, *Fast: FCN* is superior to *Fast: Time Series* method in all cases *c.f.* when comparing both methods for the *SYNTHIA-SF* dataset in table 3.2.

Both over-segmentation approaches obtain higher error for our *SYNTHIA-SF* dataset; this could be because of the more difficult road Stixel cuts of these scenes, that are more difficult to predict, *c.f.* compare *No over-segmentation* to *Fast* methods in table 3.2.

All methods are a compact model of the scene, because the complexity of Stixel representation is small compared to high resolution of the inputs, *c.f.* table 3.3.

As last remark, we want to highlight that the proposed *Fast* methods improve run-time with respect to the original Stixel approach by up to $2\times$, and $7\times$ with respect to the Slanted Stixel method, while maintaining competitive accuracy, only a slight drop in depth accuracy *c.f.* fig. 3.11. The speed-up increases with higher resolution of inputs because of the quadratic and cubic computational complexity of

original and Slanted Stixel methods, respectively. For the interested reader, we also provide per-stage run-time measurements *c.f.* table 3.1.

We also provide a qualitative evaluation on real data between our proposal and previous work. Figure 3.12 shows some of these examples, we can see that our method is capable of correctly representing non-flat roads as well as objects. The previous Stixel model produces an incomplete road representation and in one instance generates a false object that could cause an emergency break.

3.6 Conclusions

This chapter presented a novel depth model for the Stixel world that is able to account for non-flat roads and slanted objects in a compact representation that overcomes the previous restrictive constant height and depth assumptions. This change in the way Stixels are represented is required for difficult environments that are found in many real-world scenarios. Moreover, in order to significantly reduce the computational complexity of the extended model, a novel approximation has been introduced that consists of checking only reasonable Stixel cuts inferred using fast methods. We showed in extensive experiments on several related datasets that our depth model is able to better represent slanted road scenes, and that our approximation is able to reduce the run-time drastically, with only a slight drop in accuracy.

As future work, we would like to focus on circumventing the limitations of our method. Namely, (1) the vertical/column independence assumed by the model is clearly not true. A more global representation, *e.g.* super-pixels that span vertically and horizontally, would be more compact and less prone to errors; (2) some surfaces are not well represented by a linear model, *e.g.* cars. A more complex depth model and specific models for each semantic class could represent more faithfully the scene. Nonetheless, a model with more free variables could also lead to a bad representation because of the noise; (3) the proposed over-segmentation algorithm has a non-predictable run-time. And this is a bad characteristic for a real-time system. The worst-case scenario, *i.e.* no Stixel cuts removed, is as slow as not using over-segmentation at all (although very unlikely); (4) in case of movement of the stereo rig during operation, there could be an offset in roll effectively breaking the vertical world assumption.

Chapter 4

Stixels on the GPU

4.1 Introduction

Advanced driver assistance systems (ADAS), autonomous vehicles, robots and other intelligent devices need to understand their environment. Stereo camera systems provide geometric (distance) and semantic (classification) data to estimate both the semantic class and the distance of objects and the free space in a given scene. The large amount of low-level per-pixel data is very costly to transmit and process and commonly a medium-level representation known as the Stixel World [3, 44] is used. It relies on the fact that man-made environments mostly present horizontal and vertical planar surfaces, like roads, sidewalks or soil (horizontal), and buildings, pedestrians or cars (vertical). This medium-level representation must be computed in real-time to serve as a building block of higher-level modules, such as localization and planning.

The Stixel world has been successfully used for representing traffic scenes, as introduced in [44]. The intelligent vehicles community has shown an increasing interest in this model over the last years [5, 8, 10, 12, 26–28, 31, 35, 51]. It defines a compact representation of the dense 3D disparity data obtained from stereo vision that uses rectangles, the so called *Stixels*, as elements. Stixels are classified either as *ground-like* planes, upright *objects* or *sky*, which are the primitive geometric elements found in man-made environments. This representation transforms millions of disparity pixels to hundreds or thousands of Stixels. At the same time, most scene structures, such as free space and obstacles, which are relevant for autonomous driving tasks, are adequately represented.

The Stixel world can model the scene structures found in urban environments with certain constraints, *e.g.* sky is above the horizon line and objects usually lie on the ground. Generally, the geometric constraints of a scene are tightened to the vertical direction. Hence, the environment can be modeled as a column-wise segmentation of the image with a 3D stick-like shape, *i.e.* a set of Stixels, *c.f.* fig. 4.1. The segmentation of the image is estimated by solving a column-wise energy minimization problem, taking depth and semantic cues as inputs as well as *a priori* information that is used to regularize the solution. The Stixel model has been successfully used for automotive vision applications either to decrease parsing time, increase accuracy or both [6, 14, 15, 36, 39, 43, 49].

Stixel estimation is a problem with a high computational complexity. As a consequence, the algorithm implemented on a multi-core CPU by [27] does not fulfill the real-time nor the energy-efficiency requirements of autonomous driving applications. Dedicated hardware designs (*e.g.* FPGA or ASIC) may achieve these goals, but are very inflexible and expensive regarding changes in the algorithms.

Embedded GPU-accelerated systems, like the NVIDIA Jetson and DrivePX platforms, allow low-cost and low-energy consumption, real-time Stixel computation.

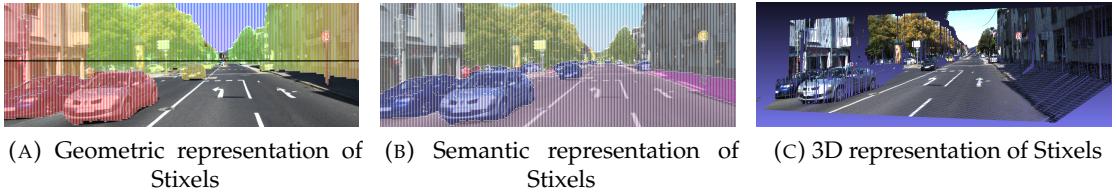


FIGURE 4.1: Scene representation of a challenging street environment obtained by our method. Geometric (left), semantic (center) and 3D (right) representations are shown. In fig. 4.1a, color encodes the distance from close (red) to far (green). In fig. 4.1b, color encodes the semantic class following [11].

GPUs are very well suited for algorithms exhibiting massive parallelism, but may suffer high performance inefficiencies with algorithms that contain inherent dependencies, as those using dynamic programming techniques like the Stixel algorithm presented by [27, 44]. Careful work distribution and task cooperation, coupled with an appropriate data layout design, may overcome those difficulties and achieve competitive performance.

This work describes and evaluates a GPU-accelerated software design of the Slanted Stixel world recently proposed in [27, 28], that achieves real-time on current low-power embedded systems. Slanted Stixels generalize the original proposal with a more flexible depth model that defines a plane in the disparity space by using two random variables (line slope and intercept). This reformulation overcomes the previous restriction of having constant depth models for *object* and *ground* Stixels and accurately represents arbitrary kinds of slanted objects and non-flat roads. It has been proved to provide substantially better quality on scenarios with non-flat roads. The better accuracy, though, comes at the price of a higher computational complexity; for example, on a six-core Intel i7-6800K processor, disparity images of 2048×1024 pixels are segmented with a Stixel resolution of 8 pixels at 6.6 frames per second (fps) [27].

The proposed fine-grained parallelization is based on our previous work done for the original Stixel model [26]. This paper modifies parts of the GPU-accelerated design and extends the description, the experimental data and the performance analysis. We show that on an embedded Tegra Xavier GPU, the same disparity images are segmented at 102.4 fps (15.5 \times faster).

In this work, we propose a reformulation of the measurement depth model that reduces the computational complexity of Slanted Stixels to the level of the original Stixels proposal, with a slight accuracy drop. Instead of using a uniform distribution to make the model robust to outlier measurements, we rely on using the confidence of the depth estimation and the identification of invalid values to handle outliers. The proposed reformulation leads to a more efficient GPU parallelization, which runs the same example 3.4 times faster on the same GPU.

The remainder of this paper is structured as follows. Section 4.2 reviews the state of the art. Section 4.3 presents the Slanted Stixel formulation, while the modification of the measurement model and its rationale is detailed in section 4.3.4. Section 4.4 explains basic concepts for efficient GPU acceleration and describes our proposed GPU-based optimizations for real-time Stixel computation. Section 4.5 presents the experiments we carried on and analyzes the accuracy and execution speed of our proposed method. Finally, we state our conclusions in section 4.6.

4.2 Related work

We will first comment on works proposing different road scene models. Occupancy grid maps are models used to represent the surrounding of the vehicle [13, 39, 40, 55]. Typically, a grid in bird’s eye perspective is defined and used to detect occupied grid cells and then, from this information, to extract the obstacles, navigable area, and unobservable areas from range data. These grids and the Stixel world both represent the 2D image in terms of column-wise stripes allowing to capture the camera data in a polar fashion. Also, the Stixel data model is similar to the forward step usually found in occupancy grid maps [12]. However, the Stixel inference method in the image domain presents important differences compared to classical grid-based approaches.

Our work builds upon the proposal from [51]: they use semantic cues in addition to depth to extract a Stixel representation, which is able to provide a rich yet compact representation of the traffic scene. We also base our method on [27]: the Slanted Stixels model incorporates a novel plane model together with effective priors on the plane parameters, and it is able to represent scenes with complex non-flat roads.

There are some methods [3, 5, 31, 35], that represent simplified scene models with a single Stixel per column. The advantage of these approaches is that the computational complexity of the underlying algorithms is linear, but they cannot represent some complex scenarios found in the real world, *e.g.* a pedestrian and a building in the same column.

A recent work [8] uses edge-based disparity maps to compute Stixels. Their method is fast but they show that it gives inferior accuracy compared to the original Stixel model [45].

Finally, there are some works proposing fast implementations for Stixel computation. The FPGA implementation from [39] runs at 25 Hz with a Stixel width of 5 pixels, but the authors do not indicate the image resolution.

Our proposed method introduces a novel Stixel-based scene representation that overcomes the high computational complexity of Slanted Stixels [27], *c.f.* fig. 4.1.

Our GPU parallelization proposal is based on our previous work [26], however, we incorporate a richer Stixel method by using more cues *e.g.* semantic segmentation [51], disparity confidence [45], as well as, the enhanced depth model for slanted scenes [27], this increases memory requirements and bandwidth usage and computational complexity. Compared to [26], we use registers (thread local memory) instead of shared memory to store the cost and index tables. This is possible because each loop iteration of the Dynamic Programming process only one thread has to send information to the rest of the threads and, then, the shared memory is necessary only to share information from one thread. Also, the index table is allocated into local registers and is moved to shared memory just before the backtracking stage, so that we can overwrite the SATs (Summed Area Table) used on the previous stage.

Our previous work [27, 28] proposed to use an over-segmentation of the optimal Stixel cuts in order to speed up the execution of the algorithm. First, the over-segmentation has to be computed in linear time so that the added run-time is not high. The biggest drawback of this approach is that even if we can guarantee that most of the time the number of Stixel cuts in the over-segmentation is significantly lower than the original cuts, the run-time of the algorithm is variable and then non-predictable, which is a must for building a real-time system. Anyway, our method is orthogonal to the over-segmentation idea, and both methods could be combined.

Our main contributions are: (1) a novel depth measurement model that overcomes the high computational complexity of Slanted Stixels [27, 28] and is also inherently GPU-friendly in contrast to the previous model; (2) a more efficient parallelization of the GPU; (3) an in-depth evaluation in terms of run-time as well as semantic and depth accuracy carried out on several benchmarks. Compared to the existing state-of-the-art approaches, our method provides a faster and predictable run-time proposal with only a slight drop in accuracy.

4.3 The Stixel Model

The Stixel world is a compressed representation of a 3D scene that preserves its relevant structure. Given that the vertical dimension dominates the structure of street environments, the Stixel world segments the image into independent columns composed of stick-like super-pixels with a 3D planar depth model and semantic labels. There are three structural classes derived exclusively from depth data: *ground* (Stixels with a slant similar to the expected ground plane), *object* (almost vertical Stixels, usually lying on the ground), and *sky* (Stixels at infinite distance). Semantic classes are refinements to those structural classes (*e.g.* road or sidewalk are ground classes, whereas building and vehicle are object classes). Prior to the segmentation, the per-pixel input images are downsized to the desired vertical and horizontal Stixel resolution.

An example of Stixel segmentation is presented in fig. 4.2. The column highlighted in the image on the right is downsized, and the disparity measurements (inverse of depth) for each Stixel on the column are shown on the left. The resulting Stixel segmentation and labeling are defined by the colored thick lines.

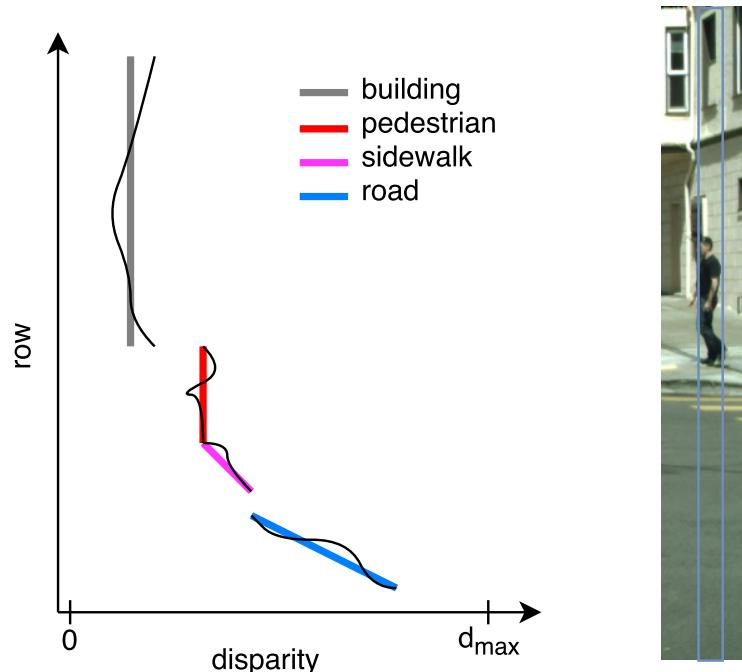


FIGURE 4.2: Example of the Stixel segmentation and labeling of a column in a typical scene (on the right). The input disparity measurements (black thin lines) and output Stixels encoded with semantic colors (colored thick lines) are shown on the left. Taken from [27].

The rest of this section defines the mathematical formulation of the Stixel model and how to solve the problem of joint optimization through dynamic programming. The last subsection presents our proposal for modifying the mathematical model in order to reduce the computational complexity of the problem.

4.3.1 Mathematical formulation

A Stixel column segmentation S consists of an arbitrary number N of Stixels, s_i , each representing four random variables: the Stixel extent via bottom row V_i^b and top row V_i^t , as well as its semantic class C_i and depth model D_i (slope and intercept). Thereby, the number of Stixels itself is a random variable that is optimized jointly during inference. The joint segmentation and labeling problem is carried out independently for each image column via optimization of the posterior distribution $P(S | M)$, a Maximum A Posteriori estimation problem (MAP) defined over a Stixel segmentation S given all measurements M from that particular column.

Applying the Bayes' theorem, the posterior probability can be rewritten using the unnormalized likelihood and prior distributions as $\frac{1}{Z} \tilde{P}(M | S) \tilde{P}(S)$. In order to avoid numerical problems with small magnitudes of the individual probabilities, the likelihoods are transformed to log-likelihoods via $P(S | M) = e^{-E(s, m)}$, and the MAP estimation problem is then converted to a cost minimization problem, where $E(\cdot)$ is the energy (or cost) function.

The energy function is the summation of the energies of the whole Stixel segmentation, which can be separated into the **likelihood** or **data** term, $E_{data}(\cdot)$, and the **prior** term, $E_{prior}(\cdot)$.

$$E(s, m) = \sum_{i=1}^N (E_{data}(s_i, m) + E_{prior}(s_i)) . \quad (4.1)$$

The **likelihood** or **data** term $E_{data}(\cdot)$ rates how well the measurements m fit to the overlapping Stixel s_i . This energy is further split in a semantic term and a depth term

$$E_{data}(s_i, m) = E_{depth}(s_i, d) + w_l \cdot E_{sem}(s_i, l) . \quad (4.2)$$

The parameter w_l controls the influence of the semantic data term. The input is provided by a fully convolutional network (FCN) that delivers normalized semantic scores $l_v(c_i)$ with $\sum_{c_i} l_v(c_i) = 1$ for all classes c_i at pixels v . The semantic energy favors semantic classes of the Stixel that fit to the observed pixel-level semantic input [51]. The semantic likelihood term is

$$E_{sem}(s_i, l) = \sum_{v=v_i^b}^{v_i^t} -\log(l_v(c_i)) . \quad (4.3)$$

The depth term is defined by a probabilistic and generative sensor model $P_v(\cdot)$ that considers the accordance of the depth measurement d_v at row v to the depth model of Stixel s_i

$$E_{depth}(s_i, d) = \sum_{v=v_i^b}^{v_i^t} -\log(P_v(D_v = d_v | S_i = s_i)) . \quad (4.4)$$

Following Slanted Stixels [27], we use a plane depth model that overcomes the previous rather restrictive constant depth and constant height assumptions for *object* and *ground* Stixels, respectively. To this end, we formulate the depth model $\mu(s_i, v)$

using two random variables defining a plane in the disparity space (slope and intercept) that evaluates to the disparity in row v via

$$\mu(s_i, v) = b_i \cdot v + a_i . \quad (4.5)$$

Note that we assume narrow Stixels and thus can neglect one plane parameter, *i.e.* the roll.

The measurement model for disparities is then defined as a combination of a Gaussian and a uniform distribution

$$P_v(D_v | s_i) = \frac{p_{\text{out}}}{Z_U} + \frac{1 - p_{\text{out}}}{Z_G(s_i)} e^{-\left(\frac{c_v(d_v - \mu(s_i, v))}{\sigma(s_i)}\right)^2} . \quad (4.6)$$

The Gaussian distribution models the typical disparity noise and the uniform distribution, weighted by a constant probability for outliers p_{out} , makes the model more robust to outliers. The Gaussian sensor noise model is centered at the expected disparity $\mu(s_i, v)$ given the depth model of the Stixel and has confidence c_v . Z_U and $Z_G(s_i)$ normalize the distributions. Similarly to [45], we use the confidence of the depth estimates c_v to influence the shape of the distribution $\sigma(s_i)$.

The **prior** or smoothness term captures the knowledge about the traffic scene, such as, *sky* Stixels are unlikely below the horizon line, objects tend to be close to the ground, or there is a small number of objects in the scene. In order to model the complexity of the segmentation, we include a constant term for each segment to favor configurations composed of fewer Stixels. The Markov property is used to reduce the prior definition to pairwise mutual dependencies of each pair of adjacent Stixels and the likelihood of the bottom Stixel. Refer to [12, 27] for a more comprehensive definition of the priors.

We define a prior term for the depth model of Stixels, $E_{\text{plane}}(s_i)$, that expects the two random variables A, B representing the plane parameters of a Stixel to be Gaussian distributed, *i.e.*

$$E_{\text{plane}}(s_i) = \left(\frac{a - \mu_{c_i}^a}{\sigma_{c_i}^a}\right)^2 + \left(\frac{b - \mu_{c_i}^b}{\sigma_{c_i}^b}\right)^2 - \log(Z) . \quad (4.7)$$

This prior favors planes in accordance to the expected 3D layout corresponding to the particular geometric class c_i . *E.g.* *object* Stixels are expected to have an approximately constant disparity, *i.e.* $\mu_{\text{object}}^b = 0$. The expected road slant μ_{ground}^a can be set using prior knowledge or by means of an specific method for road surface detection.

4.3.2 Algorithm based on dynamic programming

Dynamic Programming (DP) solves a complex problem by dividing it into simpler sub-problems and storing the partial solutions on memory. This way, when the same sub-problem appears computation time is saved by retrieving the partial solution from memory instead of solving the sub-problem repeatedly.

We apply the DP strategy to compute the column segmentation with minimum global cost. In order to express the optimization problem as a recursive resolution of smaller sub-problems we use a special notation for the three different structural classes: $ob_b^t = \{v^b, v^t, \text{object}\}$, $gr_b^t = \{v^b, v^t, \text{ground}\}$, and $sk_b^t = \{v^b, v^t, \text{sky}\}$. OB^k (*respectively*, GR^k and SK^k) refers to the aggregated cost corresponding to the optimal Stixel segmentation from position 0 to k of the given column, assuming that the last

Stixel is an *object* (respectively, *ground* and *sky*). Given the previous notation, we next show the recursive definition of the problem:

$$\begin{aligned} OB^0 &= E_{data}(ob_0^0) + E_{prior}(ob_0^0) \\ GR^0 &= E_{data}(gr_0^0) + E_{prior}(gr_0^0) \\ SK^0 &= E_{data}(sk_0^0) + E_{prior}(sk_0^0) \end{aligned} \quad (4.8)$$

$$OB^k = \min \left\{ \begin{array}{l} E_{data}(ob_0^k) + E_{prior}(ob_0^k) \\ E_{data}(ob_1^k) + E_{prior}(ob_1^k, ob^0) + OB^0 \\ E_{data}(ob_1^k) + E_{prior}(ob_1^k, gr^0) + GR^0 \\ E_{data}(ob_1^k) + E_{prior}(ob_1^k, sk^0) + SK^0 \\ \dots \\ E_{data}(ob_k^k) + E_{prior}(ob_k^k, ob^{k-1}) + OB^{k-1} \\ E_{data}(ob_k^k) + E_{prior}(ob_k^k, gr^{k-1}) + GR^{k-1} \\ E_{data}(ob_k^k) + E_{prior}(ob_k^k, sk^{k-1}) + SK^{k-1} \end{array} \right\} \quad (4.9)$$

Equation (4.8) defines the solution for the base case problem, which is the case of one Stixel made by the first single pixel. Equation (4.9) indicates how to solve a problem of size k , *i.e.* how to compute the partial solutions OB^k , GR^k , and SK^k , using the solutions for smaller problems. We only show the case for *object* Stixels, but the other cases are solved similarly. All the possible *object* Stixels ending at position k (and starting at positions from 1 to k) are connected with the last Stixel of the segmentation with minimal cost of the corresponding size, which were previously computed and memorized in C . Connections are evaluated for the three Stixel structural classes using the prior term.

Once the cost table C is completely computed, a backtracking procedure retrieves the resulting Stixel segmentation by starting from the top row of C and computing the successive minimum value $C_{min}^k = \min(OB^k, GR^k, SK^k)$.

4.3.3 Reduce the Algorithm's Complexity using SATs

The time complexity of the algorithm is estimated by noticing that, as shown by eq. (4.9), solving a sub-problem of size k requires computing the minimum cost of all the k possible positions of a cut between Stixels for the 3 possible structural classes. Since the number of structural classes is constant and k ranges from 0 to the total number of pixels in a column, h , then the Stixel segmentation problem for a single column requires $\mathcal{O}(h^2)$ steps. The backtracking phase can be done in a linear number of steps, $\mathcal{O}(h)$, by creating an index table linking each Stixel and the next Stixel with minimum cost during the DP solving phase.

Each step of the DP process must compute the prior and data terms of one single Stixel. The prior term is a function of the parameters of one or two Stixels, and can be computed in a constant number of operations. The data term, though, depends on the depth, confidence, and semantic class measurements of all the pixels composing the Stixel, and therefore requires a number of operations proportional to the Stixel length, which ranges between 1 and h . The challenge is to express the computation

of the data term as a constant number of operations. We achieve this goal by pre-computing partial results derived from the measurement data (similarly to [26, 42]) and by a slight modification of eq. (4.6) that facilitates the parallelization.

The semantic cost of a Stixel is the summation of the logarithm of the probabilities corresponding to the individual pixels (*c.f.* eq. (4.3)). We pre-compute the values corresponding to each pixel and store them into a Look-Up Table (LUT), one for each semantic class. Then, we pre-compute the prefix sum or Summed-Area Table (SAT [56]) of each LUT, *i.e.* the successive accumulated costs corresponding to all the previous pixels. The computation of the semantic data term of a Stixel is then performed in constant time as follows:

$$E_{sem}(\mathbf{s}_i, \mathbf{l}) = SAT_{c_i}(v_i^t) - SAT_{c_i}(v_i^b - 1). \quad (4.10)$$

Notice that the total computation complexity for creating the SATs corresponding to an image column is $\mathcal{O}(h)$, which is lower than the computation complexity of the DP algorithm: $\mathcal{O}(h^2)$.

The data term of *sky* Stixels only depends on the disparity and confidence of the pixel individually, and can be computed in constant time by using SATs. But the data term of *ground* and *object* Stixels depends not only on the measurements but also on the depth model used.

In a previous work [26, 42], they implemented a non-slanted depth model with a pre-determined constant road slant and a constant depth for objects. For *ground* Stixels they substituted the depth model $\mu(s_i, v)$ in eq. (4.6) and pre-computed the LUTs and the corresponding SATs for each image column. *Object* Stixels, though, have a constant model that is set as the mean disparity of the Stixel. They solved the problem by creating a separate SAT for each possible integer value for $\mu(s_i, v)$; *i.e.* they quantized the mean disparities into integer values. This approach proved empirically to be both accurate and efficient, with time and memory complexities proportional to $\mathcal{O}(h \times d_{max})$, where d_{max} is the maximum disparity measurement, which in practice is lower than h .

4.3.4 Modified measurement model for slanted Stixels

Compared to the model used in [26], the Stixel model considered here and described in section 4.3.1 is much more elaborated. One crucial drawback is that, since the slanted plane depth model defined by eq. (4.5) depends on two random variables (slant and intercept) and not one, the quantization approach is no longer viable, for the time and memory complexity to create the SATs would exceed the work saved. The advantage of the new model is that it incorporates semantic cues and confidence for the disparity measurements, that can be used to slightly modify the measurement model without significantly affecting accuracy.

First, we will describe how to compute the optimal parameters a, b for a given Stixel in constant time. Next we will explain the modification in eq. (4.6) that allows computing the data term cost in constant time.

Similarly to [28], when optimizing for the plane parameters a_i, b_i of a certain Stixel \mathbf{s}_i , all other optimization parameters are independent of the actual choice of the plane parameters, and we can simplify

$$\underset{a_i, b_i}{\operatorname{argmin}} E(\mathbf{s}, \mathbf{m}) = \underset{a_i, b_i}{\operatorname{argmin}} E_{stixel}(\mathbf{s}_i, \mathbf{m}) + E_{plane}(\mathbf{s}_i). \quad (4.11)$$

and minimize the global energy function with respect to the plane parameters of all Stixels and all geometric classes independently. By deriving the mathematical expressions we can find an optimal solution of the resulting weighted least squares problem in closed form. The calculation of the solution in constant time relies on the pre-computation of multiple SATs.

The optimal plane parameters of a Stixel s_i can be substituted in eq. (4.6) to compute the depth cost of each pixel and then compute the summation of the cost to obtain the overall cost of the Stixel, $E_{depth}(s_i, d)$, as in eq. (4.4). This is how the computation is implemented in [27, 28], giving raise to a total algorithm complexity of $\mathcal{O}(h^3)$ steps per image column, which is very expensive.

Equation (4.6), in its current form, cannot be formally derived to apply the same kind of mathematical and computational transformations as the ones done for computing in constant time the plane parameters. The problem is due to the uniform distribution that was proposed in the original Stixel world, and was critical to model the occurrence of disparity measurement outliers. Our model, though, includes alternatives to soften the effect of those outliers, like the usage of confidence for the disparity measurements (invalid disparities are modelled as having zero confidence) and the usage of semantic cues. Our proposal, then, is to remove the uniform distribution from the depth model

$$P_v(D_v | s_i) = \frac{1}{Z_G(s_i)} e^{-\left(\frac{c_v(d_v - \mu(s_i, v))}{\sigma(s_i)}\right)^2} \quad (4.12)$$

The logarithm of the previous equation can be computed in constant time by using multiple pre-computed SATs. An additional advantage of this computational design versus the proposal in [26] is that all the required SATs have time and memory complexity $\mathcal{O}(h)$, instead of $\mathcal{O}(h \times d_{max})$, and that the disparity range is not quantized.

If the input image contains w columns, then the time complexity for the proposed algorithm is $\mathcal{O}(w \times h^2)$. If outlier disparity measurements get a low confidence estimation, c_v , or the semantic data is robust for those outliers, then the accuracy provided by the proposed depth model, eq. (4.12), will be similar to the accuracy provided by the original model, eq. (4.6).

4.4 Massive Parallelization

This section describes and discusses the massively parallel organization and data layouts designed for the Stixel computation pipeline. We first start with a brief explanation of the performance-critical elements of a GPU architecture and then follow with a description of the GPU-accelerated design and the analysis of the design trade-offs.

4.4.1 Downsampling and transpose

The input to the Stixel segmentation pipeline is a collection of z dense images of width W and height H (*c.f.* fig. 4.3). The first image contains the disparity for each pixel, the second image holds the disparity confidence, and the remaining images contain the probabilities corresponding to each semantic class. The first stage in the algorithm pipeline downsizes the inputs, both in the horizontal and vertical dimensions, to produce a more compact representation and also to reduce the computational load of the subsequent stages. Since the downsized 3D output matrix will be

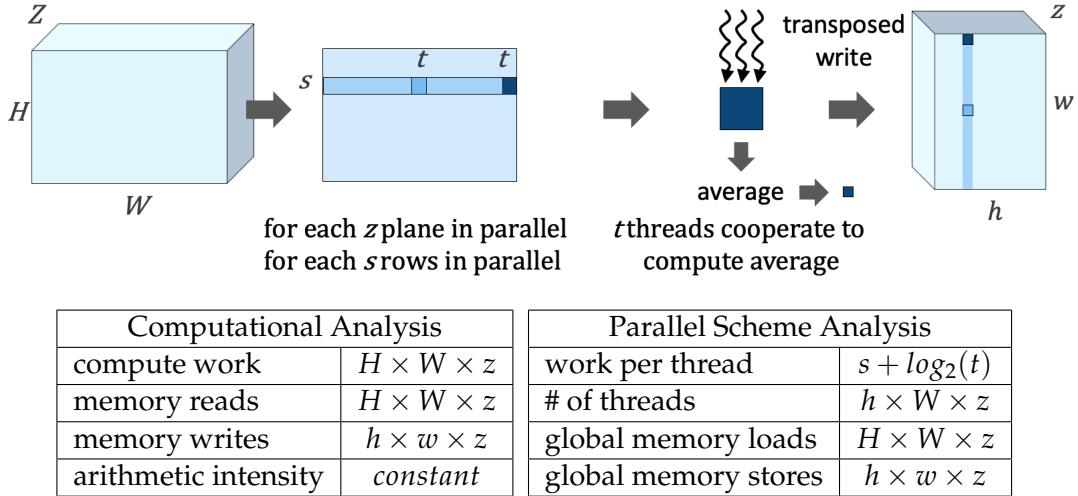


FIGURE 4.3: Downsampling and Transpose: computational analysis and parallel scheme. The width and height of the input images are W and H . There are z channels corresponding to disparity map, disparity confidence and semantic probabilities. The horizontal and vertical Stixel resolution is defined by s and t , respectively.

$$\text{Accordingly, } h = \frac{H}{s} \text{ and } w = \frac{W}{t}.$$

later processed by columns, the output data is transposed on the fly, stored as consecutive columns of memory (column-wise) instead of consecutive rows of memory (row-wise). Fusing the downsampling and transposition stages saves expensive intermediate reads and writes to global memory.

The left table in fig. 4.3 depicts the computational analysis of the algorithm. Each input data element must be read once, and must be added to its neighbor elements to provide a mean value written to the output matrix. Since the amount of input and output data on practical scenarios is too large to fit into the last-level cache of a GPU, memory operations will be solved on the device memory. Although the theoretical arithmetic intensity (ratio of abstract compute operations to memory operations) is constant, since device memory accesses are more expensive than the involved compute operations, the performance of executing this stage on a GPU will be limited by the performance of the device memory. Since there are much more memory reads than writes, this analysis encourages a thread layout aimed at maximizing the read bandwidth from the device memory.

The proposed parallel scheme, depicted in the upper part of fig. 4.3, distributes the processing of data tiles of size $s \times t$ (where s and t are the horizontal and vertical Stixel resolution, respectively) to cooperative groups of t threads, with each group operating independently to calculate a single output value. Each thread first accumulates the values corresponding to a column of s pixels, then the t threads in the group perform a cooperative horizontal reduction, and finally the first thread in the group writes the average result in the transposed position. The reduction operation is implemented using shuffle operations when t is a power of two, or else using Shared Memory.

The CTA size have been set to 256 threads and the SM occupancy is 70%, limited by the available register storage. However, the performance bottleneck has been empirically measured to be the read bandwidth to the device memory, which approaches between 70% and 90% of the peak bandwidth. The most important performance issue is to make consecutive threads (from the same group and from consecutive groups) to read data from consecutive pixels (row-wise) from the device

memory, and then promote the coalescing of memory read operations.

4.4.2 Computation of Summed Area Tables

As explained in section 4.3, our implementation makes extensive use of Summed Area Tables (SATs). There are five SATs per input image column for computing the plane parameters a and b determining the depth model of a stixel, and three additional SATs for computing the depth term cost for the three structural classes. The semantic term cost requires one SAT per semantic class; we use 18 classes in the experiments presented in section 4.5.

The generation of each SAT involves three steps: (1) read values from the input 3D matrix generated in the previous pipeline stage; (2) compute the data terms and storing them in a LUT; and (3) calculate the prefix sum of the LUT to generate the SAT. The arithmetic intensity is constant but relatively high (*c.f.* left table on fig. 4.4), due to the expensive mathematical operations that are applied to compute the data term costs.

There are several parallel configurations that are efficient for this stage. However, we opt to fuse this and the following stages in order to save the intermediate reads and writes to global memory, and reuse data on the Shared Memory. Then, the best thread configuration is determined by the computational characteristics of the next stage.

The proposed parallel scheme (upper-left part of fig. 4.4) consists of w cooperative groups of h threads. Each cooperative group generates the 26 SATs corresponding to one image column and stores them into the Shared Memory. Threads read the input (transposed) column data from Global Memory in a coalesced way, compute the LUT values in parallel and write them to Shared Memory.

The prefix sum is computed on the Shared Memory and involves a cooperative parallel pattern, requiring communication and synchronization. We use the parallel scan algorithm proposed by Harris *et al.* [22], but modify the original implementation using register-to-register *shuffle* instructions, in order to afford Shared Memory reads and writes. The collective prefix sum operations involve $\log_2(h)$ extra computation steps with respect to the serial computation. Using less than h threads improves the work-efficiency of the algorithm, and using $warp_{sz}=32$ threads is the best option, thanks to the fast hardware support for synchronization and communication at the warp level. However, in practice, since the prefix sum stage involves a very small percentage of the total computation load, we do not see any performance difference.

The GPU implementation of the original Stixels, [26], used a very large SAT that did not fit into the Shared Memory and provoked a large amount of accesses to the device memory that reduced the performance. The proposals described in section 4.3.3 and section 4.3.4 to implement the Slanted Stixels model reduce the memory requirements for the SATs and allows storing them completely into the Shared Memory.

4.4.3 Dynamic Programming stage

The Dynamic Programming (DP) computation stage, both on the original model of Slanted Stixels [27] and our proposal, has the higher computational complexity (*c.f.* left table on fig. 4.4), and for practical cases is the most time-consuming step. Our proposed design exploits the locality of the data accesses to move most memory

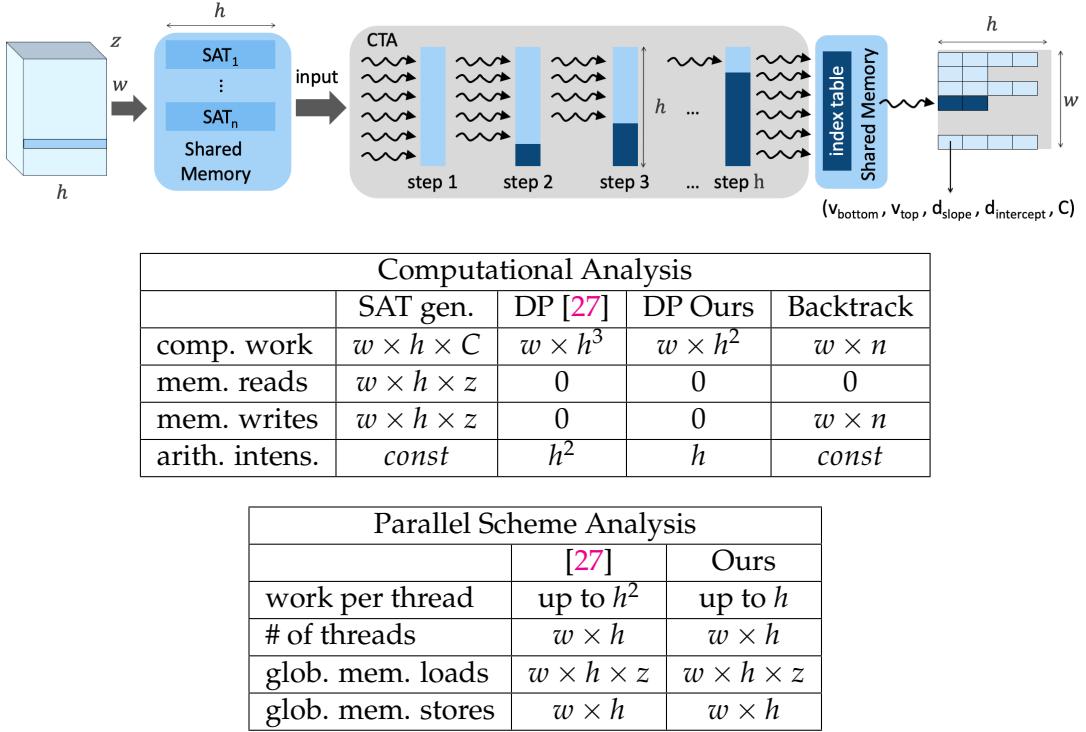


FIGURE 4.4: Fused stages for generating SATs, Dynamic Programming (DP) computation, and Backtracking. The computational analysis and parallel scheme are shown for the original proposal [27] and our proposal. w, h, z are defined in fig. 4.3. $v_{bottom}, v_{top}, d_{slope}, d_{intercept}$, and C define the Stixel properties: bottom & top row, depth model and semantic class.

accesses to the Shared Memory and the Local Memory, making the arithmetic intensity proportional to h and, therefore, we can ignore the device memory accesses. However, this stage is the most elusive for massive parallelization.

The parallel processing of each input column is simple, but not enough to efficiently exploit current GPUs for the image sizes considered in typical applications. The challenge is to extract fine-grain parallelism when processing each column, since there are data dependencies and irregular parallelism that complicate the task. To this end, we assign a Cooperative Thread Array (CTA) of h threads to a DP task associated with each column (see fig. 4.4).

The DP recurrence shown in eq. (4.9) defines how to calculate the minimum cost of a problem OB^k with k pixels using the results computed for smaller problems. The most straightforward parallel design option (A) is to use $k+1$ of the CTA threads to cooperatively compute OB^k (and GR^k and SK^k) for each problem size k ($0 \leq k < h$). An alternative option (B) is to assign each CTA thread, i , the task of computing OB^i (and GR^i and SK^i). Both parallel schemes, A and B: (1) do not balance the computation work evenly; and (2) involve data dependencies that reduce parallelism and require additional synchronization.

The first parallel design (A) starts using a single thread and increases the number of running threads progressively. Each step requires a cooperative parallel minimum operation. Option B starts using h threads and decreases the number of active threads on every step of the DP solving process. Each step involves a broadcast of the cost values computed by the running thread with minimum identifier. This last option is the one selected, and depicted on fig. 4.4.

Both parallel options involve multiple reads to consecutive positions or to the

same position on the 26 SATs. As explained before, the SATs are stored into Shared Memory, which provides efficient accesses. Only option B allows each thread to hold into the thread-private registers (or *Local Memory*) its corresponding portion of both the cost table and the index table. In each iteration, the thread with minimum identifier computes the final value in the corresponding cost table entry, and then uses the Shared Memory to broadcast that value; a barrier is used to enforce the required synchronization; and finally, that thread becomes idle. Option A is slower because it requires more synchronization operations and data movements.

The performance of this stage is latency-bounded due to the lack of parallelism. There are three causes for the limited parallelism: (1) the relatively high memory requirements on the Shared and Local Memory; (2) the decreasing amount of independent work as the recurrence loop advances; and (3) the synchronization barriers between recurrent steps, which reduce the effective parallelism.

Specifically, each thread holds an average of 26 float numbers in the Shared Memory and uses 79 local registers. The best thread block configuration contains 256 threads, which requires 19.75 Ki registers and 26 KiB of Shared Memory per block. Both Pascal and Volta CUDA architectures provide 64 Ki registers per SM (see table 4.3), which pose a limit of three 256-thread blocks per SM ($3 \times 19.75 = 59.25$ Ki registers out of 64). However, the Pascal architecture only provides 64 KiB of Shared Memory, which allows allocating two blocks of threads, while the Volta architecture provides 96 KiB of Shared Memory, and allows reaching the limit of 3 blocks of threads. Overall, the maximum GPU occupancy is 512 threads out of 2048 (25%) in a Pascal GPU, and 768 out of 2048 threads (37.5%) in a Volta GPU. The effective average GPU occupancy is almost half of the peak values, due to the reduction of parallelism in the algorithm (2), and the synchronization operations (3). Even with this hard limitations, the GPU computation cores have an utilization between 30% and 50%. Moving some data to Global Memory releases space on the Shared and Local Memory and increases the potential thread-level parallelism, but results in a much higher instruction count and performance becomes limited by the device memory latency.

The computational complexity of the original model of Slanted Stixels [27, 28] is higher ($\mathcal{O}(w \times h^3)$) than that of our novel depth model ($\mathcal{O}(w \times h^2)$), described in section 4.3.4. Also, since the original algorithm computes the cost of a Stixel with linear time complexity on the Stixel height, it suffers from a high load unbalance, which reduces the effective parallelism and, therefore, the utilization of the GPU resources.

4.4.4 Backtracking and Data compaction

The backtracking step is an inherently sequential process for each column. As described in section 4.3.2, the program navigates back on an index table created during the DP solving stage and produces a variable-size list of Stixels, *c.f.* fig. 4.4. The lack of parallelism seems to discourage a GPU implementation, but the time to transfer the index tables to the CPU, or even from Shared Memory to Global Memory, is higher than the time to perform the task on the GPU (less than 0.5% of the overall execution time).

As shown in fig. 4.4, we fuse the backtracking stage with the two previous stages. The CTA threads copy the index table from local registers to Shared Memory (reusing the space devoted to the SATs, not needed in the backtracking stage), and then a single thread processes the index table and generates the final output. A fixed (and conservatively large) amount of Global Memory is allocated per column to hold the

variable-size lists of Stixels. A final and very fast execution kernel is used to compact the information into a contiguous region of Global Memory.

4.5 Experiments

This section assesses the accuracy and performance of our proposal. We first verify that our method maintains the same accuracy level as the previous Slanted Stixel model [27]. For that purpose, we evaluate on both synthetic and real data, *c.f.* section 4.5.1, and report quantitative and qualitative results, *c.f.* section 4.5.1. We also show and discuss the performance advantage of our novel depth model for GPUs and show quantitative results, *c.f.* section 4.5.2.

4.5.1 Accuracy and Compression experiments

Datasets

We use two datasets with real images, KITTI 2015 [1, 19] and Cityscapes [11], and one dataset with synthetic images, SYNTHIA-SF (SYNTHIA San Francisco) [28].

The well-known stereo challenge KITTI 2015 contains images with sparse disparity ground truth obtained from a laser scanner and semantic segmentation ground truth. Cityscapes is a highly complex dataset with dense annotations of 19 classes. SYNTHIA-SF (SYNTHIA San Francisco) is a synthetic dataset that consists of photo-realistic frames rendered from a virtual city, with precise pixel-level depth and semantic annotations.

We evaluate depth accuracy on the training images of KITTI (200) and all the images of SYNTHIA-SF (2224). The semantic accuracy is measured on KITTI, the validation images of Cityscapes (500), and SYNTHIA-SF.

Experiment Details

Slanted Stixels [27] serves as **baseline** for the comparison with our proposal, *c.f.* section 4.3, because it represents the state-of-the-art results in terms of Stixel accuracy.

As **input**, we use disparity maps obtained via semi-global-matching (SGM) [29] and pixel-level semantic labels computed by a fully convolutional network (FCN) [38]. The parameters are taken from [51] for fair comparison.

Following [27], we use the known camera calibration to obtain expected μ_{ground}^a and μ_{ground}^b . We assume that objects are vertical and set $\sigma_{\text{object}}^b \rightarrow 0$, $\mu_{\text{object}}^b = 0$, because the disparity is too noisy for the slanted object model. Sky Stixels are assumed to be vertical and very far.

We use three **metrics** to evaluate our proposed method in terms of depth and semantic accuracy, and also in terms of data compression.

The depth accuracy is defined as the same standard metric used to evaluate on KITTI [19], which is the outlier rate of the disparity estimates. We generate back the dense disparity image from the segmentation obtained from our method and from [27]. Then, an outlier is a disparity estimation with an absolute error larger than 3 px or a relative deviation larger than 5% compared to the ground truth.

The semantic accuracy is evaluated as the average Intersection-over-Union (IoU) over all 19 classes, which is also a standard measure for semantics [17].

Data compression is measured as the average number of pixels per Stixel, and quantifies the complexity of the obtained representation.

Results

The quantitative results of our proposal and baseline as described in section 4.5.1 are shown in table 4.1 and table 4.2.

The first observation, taken from table 4.1, is that both variants provide compact representations of the surrounding, with a compression larger than $100\times$ compared to the high resolution input images. Segmentations generated by our proposal are around 5% less compact for KITTI and SYNTHIA-SF, and 36% and 45% less compact for Cityscapes, which is the more complex scenario.

Second, results from table 4.2 indicate that our method achieves very similar accuracy results on all datasets, with an increase of less than 3.5% of the disparity error and a decrease of less than 1% of the IoU. We consider that this slight degradation of the accuracy results is a small price to pay for the speed improvement that we will show on the next section.

Finally, a $4\times$ higher Stixel resolution (4×4 versus 8×8) decreases the compression of the representation between 2 and 3 times but, in return, is more accurate. Note that the disparity error of the compact representation is lower than the one of the input generated by the SGM algorithm. The Stixel world model helps removing input noise thanks to the joint inference of semantic and depth data.

TABLE 4.1: Data compression measured in pixels per Stixel of our method and [27]. We evaluate on three datasets: KITTI 2015 [19], Cityscapes [11] and SYNTHIA-SF [28], *c.f.* section 4.5.1.

Dataset	Stixel resolution: 8×8		Stixel resolution: 4×4	
	Slanted [27]	Ours	Slanted [27]	Ours
KITTI 2015	587	572	254	242
Cityscapes	1105	877	475	331
SYNTHIA-SF	1439	1379	637	606

TABLE 4.2: Accuracy of our method compared to Slanted Stixels [27], input SGM and FCN. We evaluate on three datasets: KITTI 15 [19], Cityscapes [11] and SYNTHIA-SF [28] using these metrics: Disparity Error (lower is better) and Intersection over Union (higher is better), *c.f.* section 4.5.1 and section 4.5.1. *Ours* is detailed in section 4.3.4.

Metric	Dataset	Input		Stixel resolution: 8×8		Stixel resolution: 4×4	
		SGM	FCN	Slanted Stixels [27]	Ours	Slanted Stixels [27]	Ours
Disp Error (%)	KITTI 15	8.51	-	8.53	8.72	7.81	7.93
	SYNTHIA-SF	10.26	-	8.55	8.85	7.56	7.83
IoU (%)	KITTI 15	-	44.51	43.97	43.63	44.54	44.23
	Cityscapes	-	68.22	66.87	66.75	67.92	67.78
	SYNTHIA-SF	-	34.01	33.41	33.39	33.82	33.83

The observations from the quantitative evaluation are confirmed also in the qualitative results, *c.f.* fig. 4.5.

4.5.2 Performance experiments

The main goal of our performance analysis is to evaluate run-time and efficiency on embedded devices such as the NVIDIA Tegra X2 and Tegra Xavier *c.f.* table 4.3. All the metrics are measured using NVIDIA performance tools. We assume the input

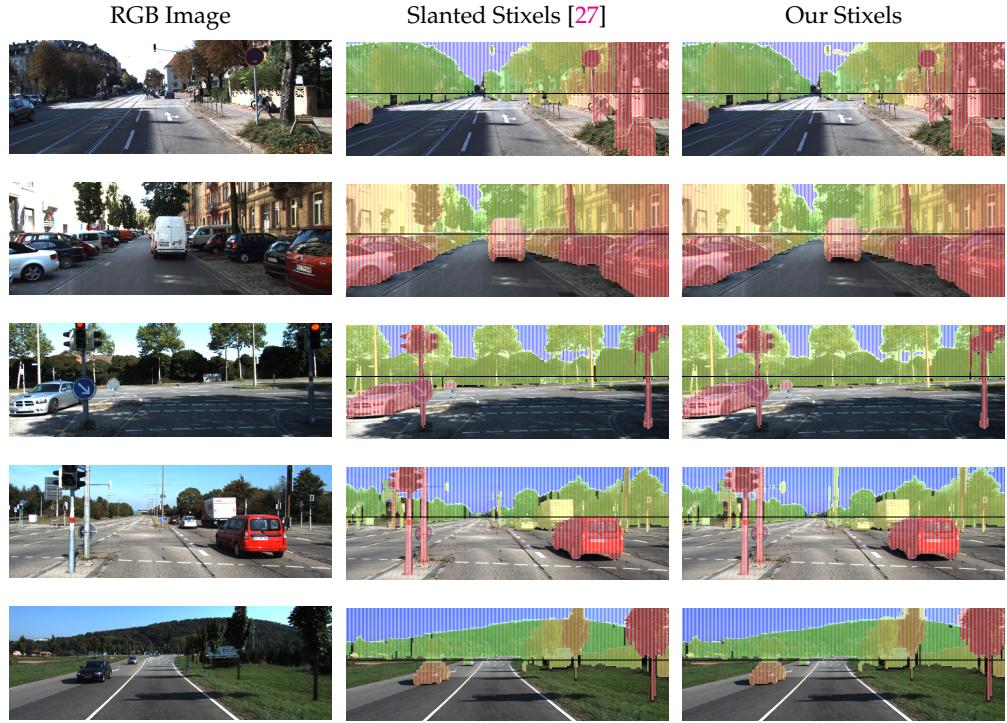


FIGURE 4.5: Exemplary outputs on real data (KITTI 2015): RGB Image (left), Slanted Stixels [27] (center) and Our Stixels (right) representations are shown. Color encodes the distance from close (red) to far (green). As we can see, the representation of our proposal is visually similar to the baseline.

data for Stixel segmentation is already into the GPU memory, and we do no add the time for moving these data from the CPU memory: Stixel estimation is just a stage in a computer-vision pipeline that receives the semantic segmentation and disparity map from stages (stereo matching and FCN) that are expected to be both executed on the GPU (*e.g.* using SGM implemented on the GPU [25]). The list of Stixels generated by the computation could be post-processed on the GPU, or is small enough to discard the time for transferring the data to the CPU memory.

Results

Figures 4.6a and 4.6b show the performance throughput (frames per second, or fps) on the two GPU systems considered, using an image size of 2048×1024 and for both 8×8 and 4×4 Stixel resolution.

It is remarkable that real-time rates higher than 90 fps are achieved by the Tegra Xavier GPU for both Stixel resolutions (see fig. 4.6a), and even the older Tegra X2 GPU is able to achieve real-time rates for Stixel resolution of 8×8 pixels (76 fps, *c.f.* fig. 4.6b). On the Tegra Xavier and for a low Stixel resolution (8×8), our method is $3.4 \times$ faster than Slanted Stixels [27] (344.3 fps vs 102.4 fps). A higher resolution (4×4) provides more accurate segments (*c.f.* table 4.2), but while our proposal achieves practical frame rates (92.3 fps), the implementation following [27] is $8.5 \times$ slower, which impedes real-time execution. In fact, the bigger the problem size (either increasing the Stixel resolution or the image size), the higher the advantage of our proposal.

A multi-threaded implementation of the original Slanted Stixels model was evaluated for the same images and a Stixel resolution of 8 pixels, and reached 6.6 fps on

TABLE 4.3: Specifications of GPUs employed in our experiments.

	Tegra X2	Tegra Xavier
architecture	Pascal	Volta
clock frequency	1465 MHz	1377 MHz
number of SMs	4	8
number of cores	256	512
registers per SM	64 Ki	64 Ki
shared memory per SM	64 KiB	96 KiB
device memory size	8 GB	32 GB
device memory bandwidth	59.7 GB/s	136.5 GB/s
L2 cache size	4096 KiB	6144 KiB
GFLOPS (single precision)	750	1410
TDP	15 W	30 W

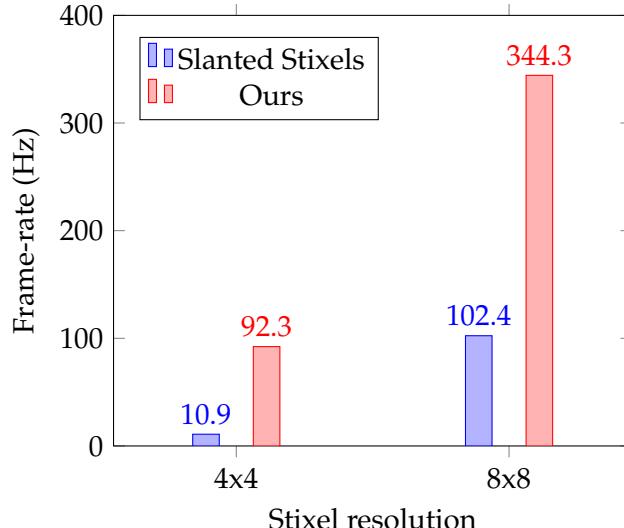
a six-core Intel i7-6800K CPU [27]. Our implementation reaches 344.3 fps on a Tegra Xavier, *i.e.* more than 50 times faster, with a reduced cost and power envelop (TDP of 30 Watt compared to 140 Watt). The alternative over-segmentation variant that selects promising Stixel cuts by means of a FCN runs at an average of 27.5 fps on the same six-core CPU. This approach has the drawback that the execution time is dependent on the image content (not predictable), with a worst-case scenario were all the Stixel cuts must be evaluated and runs even slower than the original version. Our GPU-accelerated design runs 12 times faster and with predictable times.

Figure 4.7 presents a breakdown of the elapsed time and the IPC (ratio of machine instructions executed per clock cycle and per SM). For low Stixel resolution (8×8), the time for the common *Down-sampling & Transpose* stage represents a substantial portion of the total time: 62% on the Tegra Xavier, and 47% on the Tegra X2. The performance bottleneck of this stage is the GPU memory bandwidth and its execution time is proportional to the size of the original and down-sampled images. Increasing the Stixel resolution makes the time of the *Dynamic Programming* stage to dominate on both GPUs, since the computational complexity of the DP stage with respect to the image height after down-sampling is quadratic for our proposal, while it is cubic for the original Slanted Stixels proposal.

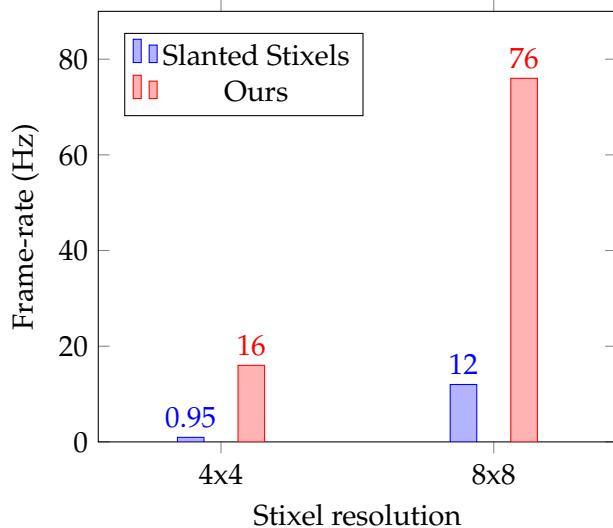
Considering only the *Dynamic Programming* stage, our proposal executes 7.2 times (8×8) and 10.9 times (4×4) faster than [27] on the Tegra Xavier. Our implementation achieves higher IPC ratios ($1.31 \times$ and $1.33 \times$ better) on each SM, which means that our approach exhibits more parallelism. But most of the speedup is due to a $5.8 \times$ and $8.7 \times$ reduction on the total number of machine instructions executed by the GPU (these data can be derived from the results in fig. 4.7). This corroborates the better algorithmic scalability of our approach.

We now assess the performance differences when using both GPUs. The Tegra Xavier contains 8 Volta SMs (512 cores) running at a slightly lower clock frequency than the 4 Pascal SMs (256 cores) in the older Tegra X2 (*c.f.* table 4.3). This means a potential raw performance advantage of $1.88 \times$. The speedup on the execution time of the *Dynamic Programming* stage is around 6.5 times for both resolutions, which means that our implementation is using the Volta cores more efficiently than the Pascal cores, partially due to a higher GPU occupancy (see section 4.4.3), which improves the IPC ratio (from $1.34 \times$ to $1.48 \times$), and partially due to a better low-level

codification efficiency (less machine instructions to implement the same basic operations) of around 2.2 times (derived fig. 4.7). The speedup of the *Down-sampling & Transpose* stage is around $3.6\times$, closer but higher than the $2.3\times$ improvement on the device memory bandwidth (from 59.7 to 136.5 GB/s). Therefore, our proposal achieves very good scalability when ported to the new GPU Xavier architecture.



(A) Computed on NVIDIA Tegra Xavier



(B) Computed on NVIDIA Tegra X2

FIGURE 4.6: Frame-rate of our method compared to Slanted Stixels [27] for 2048×1024 image resolution on the NVIDIA Tegra Xavier and Tegra X2 embedded GPUs.

4.6 Conclusion

We have described and assessed the performance of the first GPU-accelerated implementation of Slanted Stixels and we show that our algorithmic proposal is efficient for GPU parallelization. Our proposal achieves real-time performance for realistic problem sizes, proving that the low-power envelope and remarkable performance

GPU Kernel	Stixel resolution: 8×8		Stixel resolution: 4×4	
	Time (ms)	IPC	Time (ms)	IPC
Down-sampling & Transpose	1.80	1.94	2.65	2.14
Dynamic Programming [27]	7.97	1.81	89.47	1.56
Dynamic Programming (Ours)	1.11	2.38	8.20	2.08

(A) Computed on NVIDIA Tegra Xavier

GPU Kernel	Stixel resolution: 8×8		Stixel resolution: 4×4	
	Time (ms)	IPC	Time (ms)	IPC
Down-sampling & Transpose	6.38	3.26	9.85	3.68
Dynamic Programming [27]	72.77	1.05	1029.83	0.73
Dynamic Programming (Ours)	6.74	1.78	53.80	1.41

(B) Computed on NVIDIA Tegra X2

FIGURE 4.7: Breakdown of low-level performance metrics for the two main stages of our method: *Down-sampling & Transpose* stage (common) and *Dynamic Programming* stage (ours vs. [27]). *IPC* is the ratio of machine instructions executed per clock cycle and per SM. Image size is 2048×1024 .

of embedded CPU-GPU hybrid systems make them good target platforms for most real-time image processing tasks.

The reformulation of the measurement depth model proposed for Slanted Stixels improves the performance and scalability of the original proposal, while slightly reducing precision. However, in a real environment with run time limitations, the shorter execution time with respect to the original proposal allows to increase the resolution of the stixels and then improve the overall accuracy of the segmentation process. Compared to the over-segmentation proposal, our approach is more accurate, faster and with predictable run-times.

The proposed parallel scheme and data layout for the irregular computational pattern corresponding to the Dynamic Programming stage follows general optimization rules based on a simple GPU performance model. We have shown that the parallel implementation scales from a previous generation embedded GPU system to a new generation GPU, and we expect it to scale gracefully on the forthcoming GPU architectures. Our parallelization strategy is general enough to be applied to similar Dynamic Programming computational patterns, where parallelism may decrease along the processing task.

Chapter 5

Conclusions

This thesis presents a set of algorithmic modifications to 3D reconstruction methods (Semi-Global Matching and Stixel World algorithm) as well as efficient GPU parallelizations.

Our first contribution addresses the limitations of state-of-the-art GPU and FPGA parallelizations of Semi-Global Matching (SGM), and is described in chapter 2. We proposed baseline parallel schemes and data layouts for the SGM that follow general optimization rules based on a simple GPU performance model. They are designed to gracefully scale on the forthcoming GPU architectures. Then, we optimized the baseline code and improved performance around 3 times with different specific strategies, like vectorization or CTA-to-Warp conversion, that are also expected to be valid for forthcoming architectures. We show that our implementation of depth computation for stereo-camera systems is able to reach real-time performance on a NVIDIA Tegra X1 embedded GPU, proving that this kind of low-consumption systems are well capable of attaining real-time processing demands.

In chapter 3, we analyzed the limitations of the Stixel World algorithm, *i.e.* the inability to accurately represent challenging slanted roads. We presented a novel depth model for the Stixel world that is able to account for non-flat roads and slanted objects in a compact representation that overcomes the previous restrictive constant height and depth assumptions. This change in the way Stixels are represented is required for difficult environments that are found in many real-world scenarios. Moreover, in order to significantly reduce the computational complexity of the extended model, a novel approximation has been introduced that consists of checking only reasonable Stixel cuts inferred using fast methods. We showed in extensive experiments on several related datasets that our depth model is able to better represent slanted road scenes, and that our approximation is able to reduce the run-time drastically, with only a slight drop in accuracy.

In chapter 4, we have described and assessed the performance of the first GPU-accelerated implementation of Slanted Stixels and we show that our algorithmic proposal is efficient for GPU parallelization. Our proposal achieves real-time performance for realistic problem sizes, proving that the low-power envelope and remarkable performance of embedded CPU-GPU hybrid systems make them good target platforms for most real-time image processing tasks. The reformulation of the measurement depth model proposed for Slanted Stixels improves the performance and scalability of the original proposal, while slightly reducing precision. However, in a real environment with run time limitations, the shorter execution time with respect to the original proposal allows to increase the resolution of the stixels and then improve the overall accuracy of the segmentation process. Compared to the oversegmentation proposal, our approach is more accurate, faster and with predictable run-times. The proposed parallel scheme and data layout for the irregular computational pattern corresponding to the Dynamic Programming stage follows general

optimization rules based on a simple GPU performance model. We have shown that the parallel implementation scales from a previous generation embedded GPU system to a new generation GPU, and we expect it to scale gracefully on the forthcoming GPU architectures. Our parallelization strategy is general enough to be applied to similar Dynamic Programming computational patterns, where parallelism may decrease along the processing task.

Overall, the three contributions of this thesis represent the main algorithms required to implement a Stixel-based autonomous driving system. Our parallelization and algorithmic improvements make a high performance, real-time, GPU-based implementation feasible. Some code used for the thesis is freely available. It is worth noting that parallelization discussions of this thesis are also useful for other architectures, because we described all parallel and non-parallel stages of the studied algorithms.

5.1 Future work

Regarding the SGM parallelization, a suitable future exploration would be to explore real-time implementations with larger images and a higher number of disparity levels. Also, interesting extensions could be post-filtering steps such as Left-Right Consistency Check, subpixel calculation, and adaptive P2, which are well-known methods of increasing accuracy.

Regarding the Stixel representations, an interesting future work could be to focus on circumventing the limitations of the method. Namely, (1) the vertical/column independence assumed by the model is clearly not true. A more global representation, *e.g.* super-pixels that span vertically and horizontally, would be more compact and less prone to errors; (2) some surfaces are not well represented by a linear model, *e.g.* cars. A more complex depth model and specific models for each semantic class could represent more faithfully the scene. Nonetheless, a model with more free variables could also lead to a bad representation because of the noise; (3) the proposed over-segmentation algorithm has a non-predictable run-time. And this is a bad characteristic for a real-time system. The worst-case scenario, *i.e.* no Stixel cuts removed, is as slow as not using over-segmentation at all (although very unlikely); (4) in case of movement of the stereo rig during operation, there could be an offset in roll effectively breaking the vertical world assumption.

Regarding Stixel parallelization, the exploration of the usage of alternative hardware, such as FPGA or ASIC, would be interesting. Also, studying the accuracy-speed trade off in more detail is interesting to provide real-time and high quality representations of the scene.

Bibliography

- [1] Hassan Abu Alhaija et al. "Augmented Reality meets Deep Learning". In: *British Machine Vision Conference 2017, BMVC 2017, London, UK, September 4-7, 2017*. BMVA Press, 2017. URL: <https://www.dropbox.com/s/4mua02grgimu316/0369.pdf?dl=1>.
- [2] Hamid R. Arabnia. "A distributed stereocorrelation algorithm". In: *Proceedings of the 4th International Conference on Computer Communications and Networks (IC-CCN '95), September 20-23, 1995, Las Vegas, Nevada, USA*. IEEE Computer Society, 1995, p. 479. DOI: [10.1109/ICCCN.1995.540163](https://doi.org/10.1109/ICCCN.1995.540163). URL: <https://doi.org/10.1109/ICCCN.1995.540163>.
- [3] Hernán Badino, Uwe Franke, and David Pfeiffer. "The Stixel World - A Compact Medium Level Representation of the 3D-World". In: *Pattern Recognition, 31st DAGM Symposium, Jena, Germany, September 9-11, 2009. Proceedings*. 2009, pp. 51–60. DOI: [10.1007/978-3-642-03798-6_6](https://doi.org/10.1007/978-3-642-03798-6_6). URL: http://dx.doi.org/10.1007/978-3-642-03798-6_6.
- [4] Christian Banz et al. "Real-time stereo vision system using semi-global matching disparity estimation: Architecture and FPGA-implementation". In: *Proceedings of the 2010 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (IC-SAMOS 2010), Samos, Greece, July 19-22, 2010*. Ed. by Fadi J. Kurdahi and Jarmo Takala. IEEE, 2010, pp. 93–101. DOI: [10.1109/ICSAMOS.2010.5642077](https://doi.org/10.1109/ICSAMOS.2010.5642077). URL: <https://doi.org/10.1109/ICSAMOS.2010.5642077>.
- [5] Rodrigo Benenson, Radu Timofte, and Luc J. Van Gool. "Stixels estimation without depth map computation". In: *IEEE International Conference on Computer Vision Workshops, ICCV 2011 Workshops, Barcelona, Spain, November 6-13, 2011*. 2011, pp. 2010–2017. DOI: [10.1109/ICCVW.2011.6130495](https://doi.org/10.1109/ICCVW.2011.6130495). URL: <http://dx.doi.org/10.1109/ICCVW.2011.6130495>.
- [6] Rodrigo Benenson et al. "Fast Stixel Computation for Fast Pedestrian Detection". In: *Computer Vision - ECCV 2012. Workshops and Demonstrations - Florence, Italy, October 7-13, 2012, Proceedings, Part III*. 2012, pp. 11–20. DOI: [10.1007/978-3-642-33885-4_2](https://doi.org/10.1007/978-3-642-33885-4_2). URL: http://dx.doi.org/10.1007/978-3-642-33885-4_2.
- [7] Cristian Bucila, Rich Caruana, and Alexandru Niculescu-Mizil. "Model compression". In: *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA, August 20-23, 2006*. 2006, pp. 535–541. DOI: [10.1145/1150402.1150464](https://doi.org/10.1145/1150402.1150464). URL: <http://doi.acm.org/10.1145/1150402.1150464>.
- [8] Dexmont Alejandro Peña Carrillo and Alistair Sutherland. "Fast Obstacle Detection Using Sparse Edge-Based Disparity Maps". In: *Fourth International Conference on 3D Vision, 3DV 2016, Stanfordin, CA, USA, October 25-28, 2016*. IEEE Computer Society, 2016, pp. 66–72. DOI: [10.1109/3DV.2016.80](https://doi.org/10.1109/3DV.2016.80). URL: <https://doi.org/10.1109/3DV.2016.80>.

- [9] Alejandro Chacón et al. "Thread-cooperative, bit-parallel computation of levenshtein distance on GPU". In: *2014 International Conference on Supercomputing, ICS'14, Muenchen, Germany, June 10-13, 2014*. Ed. by Arndt Bode et al. ACM, 2014, pp. 103–112. DOI: [10.1145/2597652.2597677](https://doi.org/10.1145/2597652.2597677). URL: <https://doi.org/10.1145/2597652.2597677>.
- [10] Marius Cordts et al. "Object-Level Priors for Stixel Generation". In: *Pattern Recognition - 36th German Conference, GCPR 2014, Münster, Germany, September 2-5, 2014, Proceedings*. 2014, pp. 172–183. DOI: [10.1007/978-3-319-11752-2_14](https://doi.org/10.1007/978-3-319-11752-2_14). URL: http://dx.doi.org/10.1007/978-3-319-11752-2_14.
- [11] Marius Cordts et al. "The Cityscapes Dataset for Semantic Urban Scene Understanding". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. 2016, pp. 3213–3223. DOI: [10.1109/CVPR.2016.350](https://doi.org/10.1109/CVPR.2016.350). URL: <http://dx.doi.org/10.1109/CVPR.2016.350>.
- [12] Marius Cordts et al. "The Stixel World: A medium-level representation of traffic scenes". In: *Image and Vision Computing* (2017), pp. –. ISSN: 0262-8856. DOI: <http://doi.org/10.1016/j.imavis.2017.01.009>. URL: <http://www.sciencedirect.com/science/article/pii/S0262885617300331>.
- [13] Vikas Dhiman et al. "Modern MAP inference methods for accurate and fast occupancy grid mapping on higher order factor graphs". In: *2014 IEEE International Conference on Robotics and Automation, ICRA 2014, Hong Kong, China, May 31 - June 7, 2014*. IEEE, 2014, pp. 2037–2044. DOI: [10.1109/ICRA.2014.6907129](https://doi.org/10.1109/ICRA.2014.6907129). URL: <https://doi.org/10.1109/ICRA.2014.6907129>.
- [14] Markus Enzweiler et al. "Efficient Stixel-based object recognition". In: *2012 IEEE Intelligent Vehicles Symposium, IV 2012, Alcalá de Henares, Madrid, Spain, June 3-7, 2012*. 2012, pp. 1066–1071. DOI: [10.1109/IVS.2012.6232137](https://doi.org/10.1109/IVS.2012.6232137). URL: <http://dx.doi.org/10.1109/IVS.2012.6232137>.
- [15] Friedrich Erbs, Beate Schwarz, and Uwe Franke. "Stixmentation - Probabilistic Stixel based Traffic Scene Labeling". In: *British Machine Vision Conference, BMVC 2012, Surrey, UK, September 3-7, 2012*. 2012, pp. 1–12. DOI: [10.5244/C.26.71](https://doi.org/10.5244/C.26.71). URL: <https://doi.org/10.5244/C.26.71>.
- [16] Ines Ernst and Heiko Hirschmüller. "Mutual Information Based Semi-Global Stereo Matching on the GPU". In: *Advances in Visual Computing, 4th International Symposium, ISVC 2008, Las Vegas, NV, USA, December 1-3, 2008. Proceedings, Part I*. Ed. by George Bebis et al. Vol. 5358. Lecture Notes in Computer Science. Springer, 2008, pp. 228–239. DOI: [10.1007/978-3-540-89639-5\22](https://doi.org/10.1007/978-3-540-89639-5\22). URL: <https://doi.org/10.1007/978-3-540-89639-5\22>.
- [17] Mark Everingham et al. "The Pascal Visual Object Classes Challenge: A Retrospective". In: *International Journal of Computer Vision* 111.1 (2015), pp. 98–136. DOI: [10.1007/s11263-014-0733-5](https://doi.org/10.1007/s11263-014-0733-5). URL: <http://dx.doi.org/10.1007/s11263-014-0733-5>.
- [18] Eugene Fink and Harith Suman Gandhi. "Compression of time series by extracting major extrema". In: *J. Exp. Theor. Artif. Intell.* 23.2 (2011), pp. 255–270. DOI: [10.1080/0952813X.2010.505800](https://doi.org/10.1080/0952813X.2010.505800). URL: <https://doi.org/10.1080/0952813X.2010.505800>.

- [19] Andreas Geiger, Philip Lenz, and Raquel Urtasun. "Are we ready for autonomous driving? The KITTI vision benchmark suite". In: *2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012*. IEEE Computer Society, 2012, pp. 3354–3361. DOI: [10.1109/CVPR.2012.6248074](https://doi.org/10.1109/CVPR.2012.6248074). URL: <https://doi.org/10.1109/CVPR.2012.6248074>.
- [20] Alejandro González et al. "3D-Guided Multiscale Sliding Window for Pedestrian Detection". In: *Pattern Recognition and Image Analysis - 7th Iberian Conference, IbPRIA 2015, Santiago de Compostela, Spain, June 17-19, 2015, Proceedings*. Ed. by Roberto Paredes, Jaime S. Cardoso, and Xosé M. Pardo. Vol. 9117. Lecture Notes in Computer Science. Springer, 2015, pp. 560–568. DOI: [10.1007/978-3-319-19390-8_63](https://doi.org/10.1007/978-3-319-19390-8_63). URL: https://doi.org/10.1007/978-3-319-19390-8_63.
- [21] Istvan Haller and Sergiu Nedevschi. "GPU optimization of the SGM stereo algorithm". In: *Intelligent Computer Communication and Processing (ICCP)*. IEEE, 2010, pp. 197–202.
- [22] Mark Harris, Shubhabrata Sengupta, and John D Owens. "Parallel prefix sum (scan) with CUDA". In: *GPU gems 3.39* (2007), pp. 851–876.
- [23] Hu He and Ben Upcroft. "Nonparametric semantic segmentation for 3D street scenes". In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, Tokyo, Japan, November 3-7, 2013*. 2013, pp. 3697–3703. DOI: [10.1109/IROS.2013.6696884](https://doi.org/10.1109/IROS.2013.6696884). URL: <https://doi.org/10.1109/IROS.2013.6696884>.
- [24] Daniel Hernandez-Juarez et al. "3D Perception with Slanted Stixels on GPU". In: *under review on IEEE Trans. Parallel Distrib. Syst.* (forthcoming).
- [25] Daniel Hernandez-Juarez et al. "Embedded Real-time Stereo Estimation via Semi-Global Matching on the GPU". In: *International Conference on Computational Science 2016, ICCS 2016, 6-8 June 2016, San Diego, California, USA*. 2016, pp. 143–153. DOI: [10.1016/j.procs.2016.05.305](https://doi.org/10.1016/j.procs.2016.05.305). URL: <http://dx.doi.org/10.1016/j.procs.2016.05.305>.
- [26] Daniel Hernandez-Juarez et al. "GPU-Accelerated Real-Time Stixel Computation". In: *2017 IEEE Winter Conference on Applications of Computer Vision, WACV 2017, Santa Rosa, CA, USA, March 24-31, 2017*. 2017, pp. 1054–1062. DOI: [10.1109/WACV.2017.122](https://doi.org/10.1109/WACV.2017.122). URL: <https://doi.org/10.1109/WACV.2017.122>.
- [27] Daniel Hernandez-Juarez et al. "Slanted Stixels: A Way to Represent Steep Streets". In: *International Journal of Computer Vision* (Sept. 2019), pp. 1–16. ISSN: 0920-5691. DOI: [10.1007/s11263-019-01226-9](https://doi.org/10.1007/s11263-019-01226-9). URL: <https://doi.org/10.1007/s11263-019-01226-9>.
- [28] Daniel Hernandez-Juarez et al. "Slanted Stixels: Representing San Francisco's Steepest Streets". In: *British Machine Vision Conference 2017, BMVC 2017, London, UK, September 4-7, 2017*. BMVA Press, 2017. URL: <https://www.dropbox.com/s/m0n8ujh1sxvge1q/0406.pdf?dl=1>.
- [29] Heiko Hirschmüller. "Stereo Processing by Semiglobal Matching and Mutual Information". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 30.2 (2008), pp. 328–341. DOI: [10.1109/TPAMI.2007.1166](https://doi.org/10.1109/TPAMI.2007.1166). URL: <https://doi.org/10.1109/TPAMI.2007.1166>.
- [30] Heiko Hirschmüller and Daniel Scharstein. "Evaluation of Stereo Matching Costs on Images with Radiometric Differences". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 31.9 (2009), pp. 1582–1599. DOI: [10.1109/TPAMI.2008.221](https://doi.org/10.1109/TPAMI.2008.221). URL: <https://doi.org/10.1109/TPAMI.2008.221>.

- [31] Oana Ignat. "Disparity image segmentation for free-space detection". In: *2016 IEEE 12th International Conference on Intelligent Computer Communication and Processing (ICCP)*. 2016, pp. 217–224. DOI: [10.1109/ICCP.2016.7737150](https://doi.org/10.1109/ICCP.2016.7737150).
- [32] Manuel Jasch, Thomas Weber, and Matthias Rätsch. "Fast and Robust RGB-D Scene Labeling for Autonomous Driving". In: *JCP* 13.4 (2018), pp. 393–400. URL: <http://www.jcomputers.us/index.php?m=content&c=index&a=show&catid=196&id=2789>.
- [33] Abhijit Kundu et al. "Joint Semantic Segmentation and 3D Reconstruction from Monocular Video". In: *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part VI*. 2014, pp. 703–718. DOI: [10.1007/978-3-319-10599-4_45](https://doi.org/10.1007/978-3-319-10599-4_45). URL: https://doi.org/10.1007/978-3-319-10599-4_45.
- [34] Lubor Ladicky, Jianbo Shi, and Marc Pollefeys. "Pulling Things out of Perspective". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*. 2014, pp. 89–96. DOI: [10.1109/CVPR.2014.19](https://doi.org/10.1109/CVPR.2014.19). URL: <http://dx.doi.org/10.1109/CVPR.2014.19>.
- [35] Dan Levi, Noa Garnett, and Ethan Fetaya. "StixelNet: A Deep Convolutional Network for Obstacle Detection and Road Segmentation". In: *Proceedings of the British Machine Vision Conference 2015, BMVC 2015, Swansea, UK, September 7-10, 2015*. 2015, pp. 109.1–109.12. DOI: [10.5244/C.29.109](https://doi.org/10.5244/C.29.109). URL: <http://dx.doi.org/10.5244/C.29.109>.
- [36] Xiaofei Li et al. "A new benchmark for vision-based cyclist detection". In: *2016 IEEE Intelligent Vehicles Symposium, IV 2016, Gotenburg, Sweden, June 19-22, 2016*. 2016, pp. 1028–1033. DOI: [10.1109/IVS.2016.7535515](https://doi.org/10.1109/IVS.2016.7535515). URL: <https://doi.org/10.1109/IVS.2016.7535515>.
- [37] Min Lin, Qiang Chen, and Shuicheng Yan. "Network In Network". In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2014. URL: <http://arxiv.org/abs/1312.4400>.
- [38] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation". In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. 2015, pp. 3431–3440. DOI: [10.1109/CVPR.2015.7298965](https://doi.org/10.1109/CVPR.2015.7298965). URL: <http://dx.doi.org/10.1109/CVPR.2015.7298965>.
- [39] Maximilian Muffert, Nicolai Schneider, and Uwe Franke. "Stix-Fusion: A Probabilistic Stixel Integration Technique". In: *Canadian Conference on Computer and Robot Vision, CRV 2014, Montreal, QC, Canada, May 6-9, 2014*. 2014, pp. 16–23. DOI: [10.1109/CRV.2014.11](https://doi.org/10.1109/CRV.2014.11). URL: <http://dx.doi.org/10.1109/CRV.2014.11>.
- [40] Dominik Nuss et al. "Fusion of laser and radar sensor data with a sequential Monte Carlo Bayesian occupancy filter". In: *2015 IEEE Intelligent Vehicles Symposium, IV 2015, Seoul, South Korea, June 28 - July 1, 2015*. IEEE, 2015, pp. 1074–1081. DOI: [10.1109/IVS.2015.7225827](https://doi.org/10.1109/IVS.2015.7225827). URL: <https://doi.org/10.1109/IVS.2015.7225827>.
- [41] Payá-Vayá et al. "VLIW architecture optimization for an efficient computation of stereoscopic video applications". In: *Green Circuits and Systems (ICGCS)*. IEEE. 2010, pp. 457–462.

- [42] David Pfeiffer. "The Stixel World". PhD thesis. Humboldt-Universität zu Berlin, Mathematisch-Naturwissenschaftliche Fakultät II, 2012. DOI: <http://dx.doi.org/10.18452/16576>.
- [43] David Pfeiffer and Uwe Franke. "Modeling Dynamic 3D Environments by Means of The Stixel World". In: *IEEE Intell. Transport. Syst. Mag.* 3.3 (2011), pp. 24–36. DOI: <10.1109/MITS.2011.942207>. URL: <https://doi.org/10.1109/MITS.2011.942207>.
- [44] David Pfeiffer and Uwe Franke. "Towards a Global Optimal Multi-Layer Stixel Representation of Dense 3D Data". In: *British Machine Vision Conference, BMVC 2011, Dundee, UK, August 29 - September 2, 2011. Proceedings*. 2011, pp. 1–12. DOI: <10.5244/C.25.51>. URL: <http://dx.doi.org/10.5244/C.25.51>.
- [45] David Pfeiffer, Stefan Gehrig, and Nicolai Schneider. "Exploiting the Power of Stereo Confidences". In: *2013 IEEE Conference on Computer Vision and Pattern Recognition, Portland, OR, USA, June 23-28, 2013*. 2013, pp. 297–304. DOI: <10.1109/CVPR.2013.45>. URL: <http://dx.doi.org/10.1109/CVPR.2013.45>.
- [46] Germán Ros et al. "The SYNTHIA Dataset: A Large Collection of Synthetic Images for Semantic Segmentation of Urban Scenes". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 3234–3243. DOI: <10.1109/CVPR.2016.352>. URL: <https://doi.org/10.1109/CVPR.2016.352>.
- [47] Ilya D. Rosenberg et al. "Real-time stereo vision using semi-global matching on programmable graphics hardware". In: *33. International Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 2006, Boston, Massachusetts, USA, July 30 - August 3, 2006, Sketches*. Ed. by John W. Finnegan and Hanspeter Pfister. ACM, 2006, p. 89. DOI: <10.1145/1179849.1179960>. URL: <https://doi.org/10.1145/1179849.1179960>.
- [48] Daniel Scharstein and Richard Szeliski. "A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms". In: *Int. J. Comput. Vis.* 47.1-3 (2002), pp. 7–42. DOI: <10.1023/A:1014573219977>. URL: <https://doi.org/10.1023/A:1014573219977>.
- [49] Timo Scharwächter et al. "Stixmantics: A Medium-Level Model for Real-Time Semantic Scene Understanding". In: *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V*. 2014, pp. 533–548. DOI: 10.1007/978-3-319-10602-1_35. URL: https://doi.org/10.1007/978-3-319-10602-1_35.
- [50] Lukas Schneider et al. "Multimodal Neural Networks: RGB-D for Semantic Segmentation and Object Detection". In: *Image Analysis - 20th Scandinavian Conference, SCIA 2017, Tromsø, Norway, June 12-14, 2017, Proceedings, Part I*. Ed. by Puneet Sharma and Filippo Maria Bianchi. Vol. 10269. Lecture Notes in Computer Science. Springer, 2017, pp. 98–109. DOI: 10.1007/978-3-319-59126-1_9. URL: https://doi.org/10.1007/978-3-319-59126-1_9.
- [51] Lukas Schneider et al. "Semantic Stixels: Depth is not enough". In: *2016 IEEE Intelligent Vehicles Symposium, IV 2016, Gotenburg, Sweden, June 19-22, 2016*. 2016, pp. 110–117. DOI: <10.1109/IVS.2016.7535373>. URL: <http://dx.doi.org/10.1109/IVS.2016.7535373>.

- [52] Sunando Sengupta et al. "Urban 3D semantic modelling using stereo vision". In: *2013 IEEE International Conference on Robotics and Automation, Karlsruhe, Germany, May 6-10, 2013*. 2013, pp. 580–585. DOI: [10.1109/ICRA.2013.6630632](https://doi.org/10.1109/ICRA.2013.6630632). URL: <https://doi.org/10.1109/ICRA.2013.6630632>.
- [53] Robert Spangenberg, Tobias Langner, and Raúl Rojas. "Weighted Semi-Global Matching and Center-Symmetric Census Transform for Robust Driver Assistance". In: *Computer Analysis of Images and Patterns - 15th International Conference, CAIP 2013, York, UK, August 27-29, 2013, Proceedings, Part II*. Ed. by Richard C. Wilson et al. Vol. 8048. Lecture Notes in Computer Science. Springer, 2013, pp. 34–41. DOI: [10.1007/978-3-642-40246-3\5](https://doi.org/10.1007/978-3-642-40246-3_5). URL: [https://doi.org/10.1007/978-3-642-40246-3\5](https://doi.org/10.1007/978-3-642-40246-3_5).
- [54] Robert Spangenberg et al. "Large scale Semi-Global Matching on the CPU". In: *2014 IEEE Intelligent Vehicles Symposium Proceedings, Dearborn, MI, USA, June 8-11, 2014*. IEEE, 2014, pp. 195–201. DOI: [10.1109/IVS.2014.6856419](https://doi.org/10.1109/IVS.2014.6856419). URL: <https://doi.org/10.1109/IVS.2014.6856419>.
- [55] Sebastian Thrun. "Robotic Mapping: A Survey". In: *Exploring Artificial Intelligence in the New Millennium*. Ed. by G. Lakemeyer and B. Nebel. Morgan Kaufmann, 2002.
- [56] Paul A. Viola and Michael J. Jones. "Rapid Object Detection using a Boosted Cascade of Simple Features". In: *2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2001), with CD-ROM, 8-14 December 2001, Kauai, HI, USA*. IEEE Computer Society, 2001, pp. 511–518. DOI: [10.1109/CVPR.2001.990517](https://doi.org/10.1109/CVPR.2001.990517). URL: <https://doi.org/10.1109/CVPR.2001.990517>.
- [57] Saining Xie and Zhuowen Tu. "Holistically-Nested Edge Detection". In: *International Journal of Computer Vision* 125.1-3 (2017), pp. 3–18. DOI: [10.1007/s11263-017-1004-z](https://doi.org/10.1007/s11263-017-1004-z). URL: <https://doi.org/10.1007/s11263-017-1004-z>.
- [58] Philippe Xu et al. "Information Fusion on Oversegmented Images: An Application for Urban Scene Understanding". In: *Proceedings of the 13. IAPR International Conference on Machine Vision Applications, MVA 2013, Kyoto, Japan, May 20-23, 2013*. 2013, pp. 189–193. URL: <http://www.mva-org.jp/Proceedings/2013USB/papers/08-04.pdf>.
- [59] Ramin Zabih and John Woodfill. "Non-parametric Local Transforms for Computing Visual Correspondence". In: *Computer Vision - ECCV'94, Third European Conference on Computer Vision, Stockholm, Sweden, May 2-6, 1994, Proceedings, Volume II*. Ed. by Jan-Olof Eklundh. Vol. 801. Lecture Notes in Computer Science. Springer, 1994, pp. 151–158. DOI: [10.1007/BFb0028345](https://doi.org/10.1007/BFb0028345). URL: <https://doi.org/10.1007/BFb0028345>.
- [60] Jure Zbontar and Yann LeCun. "Computing the stereo matching cost with a convolutional neural network". In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. IEEE Computer Society, 2015, pp. 1592–1599. DOI: [10.1109/CVPR.2015.7298767](https://doi.org/10.1109/CVPR.2015.7298767). URL: <https://doi.org/10.1109/CVPR.2015.7298767>.
- [61] Feihu Zhang et al. "GA-Net: Guided Aggregation Net for End-To-End Stereo Matching". In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 2019, pp. 185–194. DOI: [10.1109/CVPR.2019.00027](https://doi.org/10.1109/CVPR.2019.00027). URL: http://openaccess.thecvf.com/content_CVPR_2019/html/Zhang_GA-Net.html

- _Guided \ _Aggregation \ _Net \ _for \ _End - To - End \ _Stereo \ _Matching \ _CVPR\ _2019\ _paper.html.
- [62] Richard Zhang et al. "Sensor fusion for semantic segmentation of urban scenes". In: *IEEE International Conference on Robotics and Automation, ICRA 2015, Seattle, WA, USA, 26-30 May, 2015*. IEEE, 2015, pp. 1850–1857. DOI: [10.1109/ICRA.2015.7139439](https://doi.org/10.1109/ICRA.2015.7139439). URL: <https://doi.org/10.1109/ICRA.2015.7139439>.
 - [63] Julius Ziegler et al. "Making Bertha Drive - An Autonomous Journey on a Historic Route". In: *IEEE Intell. Transport. Syst. Mag.* 6.2 (2014), pp. 8–20. DOI: [10.1109/MITS.2014.2306552](https://doi.org/10.1109/MITS.2014.2306552). URL: <https://doi.org/10.1109/MITS.2014.2306552>.
 - [64] Julius Ziegler et al. "Trajectory planning for Bertha - A local, continuous method". In: *2014 IEEE Intelligent Vehicles Symposium Proceedings, Dearborn, MI, USA, June 8-11, 2014*. IEEE, 2014, pp. 450–457. DOI: [10.1109/IVS.2014.6856581](https://doi.org/10.1109/IVS.2014.6856581). URL: <https://doi.org/10.1109/IVS.2014.6856581>.