3D Perception with Slanted Stixels on GPU

Daniel Hernandez-Juarez, Antonio Espinosa, David Vazquez, Antonio M. Lopez, and Juan C. Moure

Abstract—This work presents a GPU-accelerated software design of the recently proposed model of Slanted Stixels, which represents the geometric and semantic information of a scene in a compact and accurate way. 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. The proposed massively parallel scheme and data layout for the irregular computation pattern that corresponds to a Dynamic Programming paradigm is described and carefully analyzed in performance terms. Performance is shown to scale gracefully on current generation embedded GPUs. We assess the proposed methods in terms of semantic and geometric accuracy as well as run-time performance on three publicly available benchmark datasets. Our approach achieves real-time performance with high accuracy for 2048×1024 image sizes and 4×4 Stixel resolution on the low-power embedded GPU of an NVIDIA Tegra Xavier.

Index Terms—Stereo Vision, Stixel World, Autonomous Vehicles, Scene Understanding, Computer Vision, Embedded Systems, GPU Acceleration.

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 [1], [2] 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 [2]. The field of intelligent vehicles has been using this model over the last years [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13]. The Stixel world 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 urban 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 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. 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 [14], [15], [16], [17], [18], [19], [20].

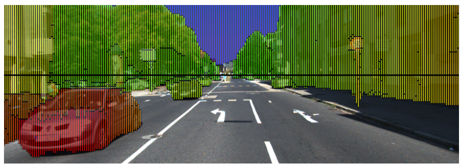
The *Slanted* Stixel world model recently proposed in [11], [12] generalizes the original proposal with a more flexible plane model that overcomes the previous rather restrictive constant depth and constant height assumptions for *object* and *ground* Stixels, respectively, and accurately represents arbitrary kinds of slanted objects and non-flat roads. Basically, the Slanted model defines a plane in the disparity space defined by two random variables: line slope and intercept. It has been proved to provide substantially better quality on scenarios with non-flat roads.

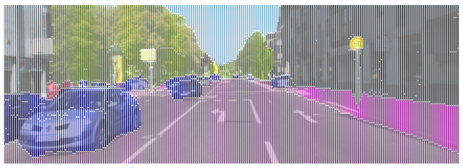
Stixel segmentation is a highly computationally complex optimization problem, and the higher representation quality of the slanted model comes at the price of even higher complexity: if the input image contains h rows and w columns, the algorithmic complexity is $\mathcal{O}(w \times h^3)$. One way to reduce complexity is to reduce the size of the input images (depth and semantics), both in horizontal and vertical dimensions. The problem is that increasing the granularity of the Stixels (or, equivalently, decreasing the resolution of the Stixel) also reduces the quality of the segmented representation.

2048-by-1024-pixel images scaled 4 and 8 times are segmented, respectively, at 0.7 and 6.6 frames per second (fps) on a six-core Intel i7-6800K processor [12]. These performance results do not meet the real-time or energy efficiency requirements of autonomous driving applications. Real-time performance is achieved by further reducing the resolution of the input images to levels that sacrifice the quality of the resulting segmentation. Dedicated hardware designs (*e.g.* FPGA or ASIC) may provide faster implementations,

- *D. Hernandez-Juarez, A. Espinosa, A. M. Lopez and J. C. Moure are with the Universitat Autonoma de Barcelona.*
  E-mail: dhernandez0@gmail.com
- *A. M. Lopez is also with Computer Vision Center.*
- *D. Vazquez is with Element AI.*

Manuscript received Month XX, 20XX; revised Month XX, 20XX.


(a) Geometric representation of Stixels


(b) Semantic representation of Stixels

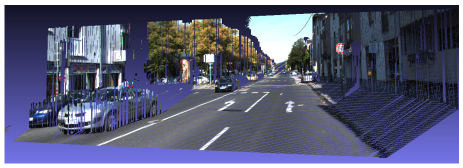

(c) 3D representation of Stixels

Fig. 1: Scene representation of a challenging street environment obtained by our method. Geometric or Depth (left), semantic (center) and 3D (right) representations are shown. In fig. 1a, color encodes depth from close (red) to far (green). In fig. 1b, color encodes the semantic class following [21].

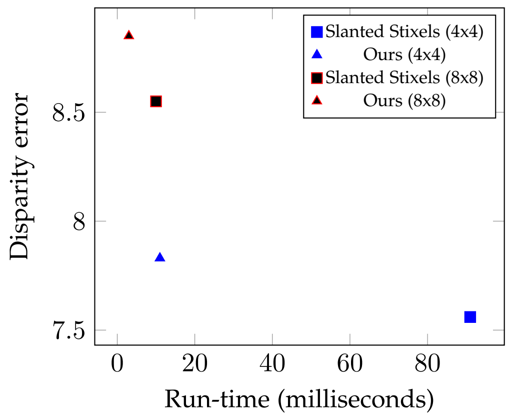


Fig. 2: This chart illustrates the trade-off between accuracy and run-time. We compare Slanted Stixels [12] and our method, both with Stixel resolution 4×4 and 8×8. Disparity error corresponds to results for SYNTHIA-SF [11], and run-time was measured on the NVIDIA Tegra Xavier embedded GPU. Our model with 4×4 Stixel resolution (blue triangle) shows a good trade-off between accuracy and run-time, being the one closer to the bottom-left corner.

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. GPUs are very well suited for algorithms exhibiting massive parallelism, like the dynamic programming techniques used for Stixel segmentation. However, to achieve competitive performance, inefficiencies due to existing data dependencies that lead to explicit synchronizations, must be reduced by careful work distribution and cooperation, along with a proper data layout design.

Our goal is to develop a faster algorithm and GPU-accelerated implementation solving the optimization problem for Slanted Stixel segmentation. We want to process high-resolution input images at high speed in order to reach real-time performance with low energy consumption, but without sacrificing segmentation accuracy. To this extent, we propose a novel depth measurement model that enables the application of several algorithmic techniques to reduce the computational complexity of Slanted Stixels from $\mathcal{O}(w \times h^3)$ to $\mathcal{O}(w \times h^2)$. Very briefly, the proposed cost equations can be formally derived to find an optimal solution of the resulting weighted least squares problem in closed form, an then multiple pre-computed Summed Area Tables (SATs) can be used to compute the cost of each Stixel in constant time. The definition of Slanted Stixels using two random variables (slant and intercept) instead of one, disables the usage of the same algorithmic strategies used for the basic Stixels [2]. For more details, see section 3.4.

We have also developed a completely new massively parallel design for GPU execution based on some ideas taken from our previous work [4], which implements the algorithm for the original Stixel model [2]. Compared to [4], our proposal makes better use of local (register) and shared memory for storing cost, index and Summed Area tables, and avoids some of the synchronization operations. These enhancements are achieved by modifying the distribution of work among threads, and by reusing the same memory areas at different phases of the execution, see section 4.

Table 1 helps comparing our contribution with respect to the previous work. The last row in the table describes our proposal for a novel cost equation that basically removes the usage of a uniform distribution to model the occurrence of disparity measurement outliers by the usage of the confidence of the depth estimation and of the semantic cues. Thanks to this reformulation, our algorithm has lower computational complexity than the original algorithm for Slanted Stixels. Also, the last column in the table describes the characteristics of the two GPU implementations that we have developed and evaluated in this paper. The proposed cost equation does not only lead to an algorithm that computes Stixel costs in constant time, but also allows using smaller SATs (Summed Area Tables), which fit into the shared memory of the GPU and contribute to a more efficient execution.

We have done an in-depth evaluation of the two GPU-accelerated programs in terms of run-time as well as semantic and depth accuracy, carried out on several benchmarks. Figure 2 shows some of the main results to illustrate the trade-off between accuracy and run-time. Our proposal using the modified cost equation increases the average disparity error around 3.5% for both Stixel resolutions (4x and 8x), but the resulting program runs on an embedded low-power GPU more than 8 times faster. In this particular H/W system, our GPU implementation of the original Slanted Stixels achieves real-time performance only for Stixel resolution 8x8 (red square), while our GPU implementation using the novel model achieves real-time performance for Stixel resolution 4x4 (blue triangle), which improves the disparity error by more than 9%. More details are provided in section 5.

Model	Inputs	Complexity	Accuracy	GPU version
Stixel World [2]	Disparity	$w \times h^2$	Bad	Implemented in [4] Most SATs in global memory Cost and Index table in shared memory Cost computed in constant time
Slanted Stixels [12]	Disparity Disparity confidence Semantic segmentation	$w \times h^3$	Good	Implemented in this work Semantic SATs in shared memory Cost and Index table in registers Disparity SATs not feasible: cost computed in linear time Very unbalanced work
Novel Disparity Cost Measurement [ours]	Disparity Disparity confidence Semantic segmentation	$w \times h^2$	Good	Implemented in this work Disparity and Semantic SATs in shared memory Cost and index table in registers Cost computed in constant time

TABLE 1: Comparison of the different Stixel models in terms of inputs, complexity and GPU implementation details.

To sum up, we provide two versions for GPU-accelerated Stixel segmentation that can achieve real-time and low-energy consumption in our embedded system hardware. Both versions offer a different trade-off between segmentation accuracy and running time. In a time-limited and low-power scenario, the version using our proposed cost measurement equation ends up providing better segmentation accuracy.

The remainder of this paper is structured as follows. Section 2 reviews the state of the art. Section 3 presents the formulation of Slanted Stixels, while the modification of the measurement model and its rationale is detailed in section 3.4. Section 4 explains basic concepts for efficient GPU acceleration and describes our proposed GPU-based optimizations for real-time Stixel computation. Section 5 presents the experiments we carried out and analyzes the accuracy and execution speed of our proposed method. Finally, we state our conclusions in section 6.

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 [17], [22], [23], [24]. 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 non-observable 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 [7]. 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 [3]: 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 [12]: 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 [1], [5], [8], [9], 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 [10] uses edge-based disparity maps to compute Stixels. Their method is fast but gives inferior accuracy compared to the original Stixel model [25].

Finally, there are some works proposing fast implementations for Stixel computation. The FPGA implementation from [17] runs at 25 Hz with an image resolution of 1 megapixels and a Stixel resolution of 5 pixels.

An earlier work [11], [12] proposed a method to rule out the most unlikely Stixel cuts, thereby saving computational time by applying the Stixel algorithm only to the probable Stixel cuts. The methods for generating the over-segmentation of the image where restricted to have linear time complexity so that the added run-time is not high. Two methods were analyzed, one based on times series compression and one using a trained neural network. In practice, most of the time the number of feasible Stixel cuts remaining in the over-segmentation is significantly small. But the biggest drawback of this approach is that the run-time of the algorithm is variable and then non-predictable, which is a problem for building a real-time system. Anyway, this method is orthogonal to our proposal and both could be combined.

A previous GPU-accelerated version of the Stixel's method [4] only includes the original non-slanted model [2]. The GPU version proposed in this work implements a richer Stixel method that incorporates more cues, *e.g*. semantic segmentation [3], disparity confidence [25], as well as, a novel depth model for slanted scenes [12]. Compared to [4], the usage of local (register) and shared memory is improved by an enhanced arrangement of the Summed Area Tables (SATs) and the cost and index tables, by a better distribution of the work among the parallel threads, and by a simpler synchronization required during the execution.

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. 3. 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.

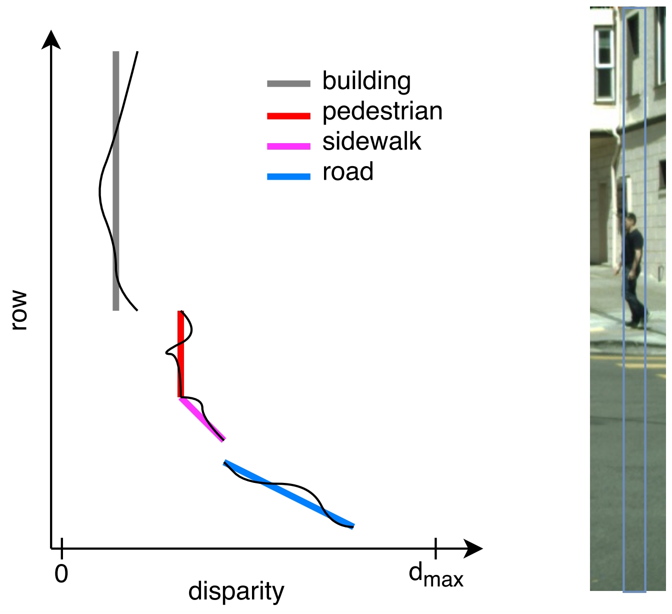


Fig. 3: Example of the Stixel segmentation and labeling of a column in a typical scene (on the right). The input disparity measurements (black thin lines) and output Stixels encoded with semantic colors (colored thick lines) are shown on the left. Taken from [12].

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.

3.1 Mathematical formulation

A Stixel column segmentation $\boldsymbol{S}$ consists of an arbitrary number N of Stixels, $\boldsymbol{s}_i$, 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 $P(\boldsymbol{S} \mid \boldsymbol{M})$, a Maximum A Posteriori estimation problem (MAP) defined over a Stixel segmentation $\boldsymbol{S}$ given all measurements $\boldsymbol{M}$ from that particular column.

Applying the Bayes' theorem, the posterior probability can be rewritten using the unnormalized likelihood and prior distributions as $\frac{1}{Z}\tilde{P}(\boldsymbol{M} \mid \boldsymbol{S})\,\tilde{P}(\boldsymbol{S})$. In order to avoid numerical problems with small magnitudes of the individual probabilities, the likelihoods are transformed to log-likelihoods via $P(\boldsymbol{S} \mid \boldsymbol{M}) = e^{-E(\boldsymbol{s},\boldsymbol{m})}$, 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, $E_{prior}(\cdot)$.

$$E(\boldsymbol{s},\boldsymbol{m}) = \sum_{i=1}^{N} \left(E_{data}(\boldsymbol{s}_i,\boldsymbol{m}) + E_{prior}(\boldsymbol{s}_i)\right) . \tag{1}$$

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

$$E_{data}(\boldsymbol{s}_i,\boldsymbol{m}) = E_{depth}(\boldsymbol{s}_i,\boldsymbol{d}) + w_l \cdot E_{sem}(\boldsymbol{s}_i,\boldsymbol{l}) . \tag{2}$$

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 [3]. The semantic likelihood term is

$$E_{sem}(\boldsymbol{s}_i,\boldsymbol{l}) = \sum_{v=v_i^{\mathrm{b}}}^{v_i^{\mathrm{t}}} -log(l_v(c_i)) . \tag{3}$$

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 $\boldsymbol{s}_i$

$$E_{depth}(\boldsymbol{s}_i,\boldsymbol{d}) = \sum_{v=v_i^{\mathrm{b}}}^{v_i^{\mathrm{t}}} -\log(P_v(D_v = d_v \mid \boldsymbol{S}_i = \boldsymbol{s}_i)) . \tag{4}$$

Following Slanted Stixels [12], 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 $D(\boldsymbol{s}_i, v)$ using two random variables defining a plane in the disparity space (slope and intercept) that evaluates to the disparity in row v via

$$D(\boldsymbol{s}_i, v) = b_i \cdot v + a_i . \tag{5}$$

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

$$P_v(D_v \mid \boldsymbol{S}_i) = \frac{p_{\text{out}}}{Z_U} + \frac{1-p_{\text{out}}}{Z_G(\boldsymbol{s}_i)} e^{-\left(\frac{c_v(d_v - D(\boldsymbol{s}_i,v))}{\sigma(\boldsymbol{s}_i)}\right)^2} . \tag{6}$$

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 $D(\boldsymbol{s}_i, v)$ given the depth model of the

Stixel and has confidence c_v. Z_U and $Z_G(\boldsymbol{s}_i)$ normalize the distributions. Similarly to [25], we use the confidence of the depth estimates c_v to influence the shape of the distribution $\sigma(\boldsymbol{s}_i)$.

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 [7], [12] for a more comprehensive definition of the priors.

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

$$E_{plane}(\boldsymbol{s}_i) = \left(\frac{a-\mu^a_{c_i}}{\sigma^a_{c_i}}\right)^2 + \left(\frac{b-\mu^b_{c_i}}{\sigma^b_{c_i}}\right)^2 - \log(Z) \ . \quad (7)$$

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.* $\mu^b_{object} = 0$. The expected road slant μ^a_{ground} can be set using prior knowledge or by means of an specific method for road surface detection.

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: $ob^t_b = \{v^b, v^t, object\}$, $gr^t_b = \{v^b, v^t, ground\}$, and $sk^t_b = \{v^b, v^t, sky\}$. 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:

$$\begin{aligned} OB^0 &= E_{data}(ob^0_0) + E_{prior}(ob^0_0) \\ GR^0 &= E_{data}(gr^0_0) + E_{prior}(gr^0_0) \\ SK^0 &= E_{data}(sk^0_0) + E_{prior}(sk^0_0) \end{aligned} \quad (8)$$

$$OB^k = min \begin{cases} E_{data}(ob^k_0) + E_{prior}(ob^k_0) \\ E_{data}(ob^k_1) + E_{prior}(ob^k_1, ob^0) + OB^0 \\ E_{data}(ob^k_1) + E_{prior}(ob^k_1, gr^0) + GR^0 \\ E_{data}(ob^k_1) + E_{prior}(ob^k_1, sk^0) + SK^0 \\ ... \\ E_{data}(ob^k_k) + E_{prior}(ob^k_k, ob^{k-1}) + OB^{k-1} \\ E_{data}(ob^k_k) + E_{prior}(ob^k_k, gr^{k-1}) + GR^{k-1} \\ E_{data}(ob^k_k) + E_{prior}(ob^k_k, sk^{k-1}) + SK^{k-1} \end{cases} \quad (9)$$

Equation (8) defines the solution for the base case problem, which is the case of one Stixel made by the first single pixel. Equation (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 $C^k_{min} = min(OB^k, GR^k, SK^k)$.

3.3 Reduce the Algorithm's Complexity using SATs

As shown by eq. (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. We use this to calculate the time complexity of the algorithm. 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 [4], [26]) and by a slight modification of eq. (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. (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 [27]) 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:

$$E_{sem}(\boldsymbol{s}_i, \boldsymbol{l}) = SAT_{c_i}(v_i^t) - SAT_{c_i}(v_i^b - 1) \ . \tag{10}$$

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 [4], [26], 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 $D(\boldsymbol{s}_i, v)$ in eq. (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 $D(\boldsymbol{s}_i, v)$; *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.

3.4 Modified measurement model for slanted Stixels

Compared to the model used in [4], the Stixel model considered here and described in section 3.1 is much more elaborated. One crucial drawback is that, since the slanted plane depth model defined by eq. (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. (6) that allows computing the data term cost in constant time.

Similarly to [11], when optimizing for the plane parameters a_i, b_i of a certain Stixel $\boldsymbol{s}_i$, all other optimization parameters are independent of the actual choice of the plane parameters, and we can simplify

$$\underset{a_i, b_i}{\operatorname{argmin}}\, E(\boldsymbol{s}, \boldsymbol{m}) = \underset{a_i, b_i}{\operatorname{argmin}}\, E_{stixel}(\boldsymbol{s}_i, \boldsymbol{m}) + E_{plane}(\boldsymbol{s}_i) \ . \tag{11}$$

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 $\boldsymbol{s}_i$ can be substituted in eq. (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, $E_{depth}(\boldsymbol{s}_i, \boldsymbol{d})$, as in eq. (4). This is how the computation is implemented in [11], [12], giving raise to a total algorithm complexity of $\mathcal{O}(h^3)$ steps per image column, which is very expensive.

Equation (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

$$P_v(D_v \mid \boldsymbol{S}_i) = \frac{1}{Z_G(\boldsymbol{s}_i)} e^{-\left(\frac{c_v(d_v - D(\boldsymbol{s}_i, v))}{\sigma(\boldsymbol{s}_i)}\right)^2} \tag{12}$$

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 [4] 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. (12), will be similar to the accuracy provided by the original model, eq. (6).

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.

4.1 GPU architecture and performance

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 budged. 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 (or number of vector lanes).

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.

4.2 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). 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 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 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, distributes the average reduction of the data in image blocks 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.

4.3 Computation of Summed Area Tables

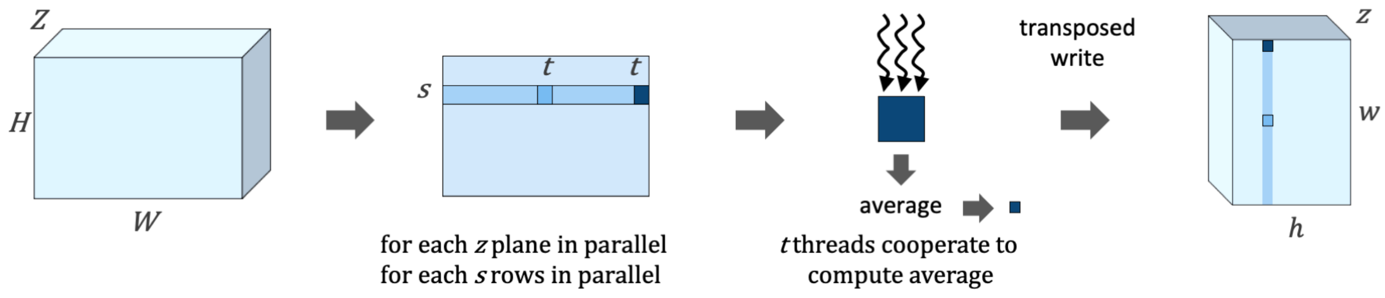
As explained in section 3, our implementation makes extensive use of Summed Area Tables (SATs). There are five SATs per input image column for computing the plane parameters a and b determining the depth model of a stixel, and three additional SATs for computing the depth term cost for the three structural classes. The semantic term cost requires one SAT per semantic class; we use 18 classes in the experiments presented in section 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. 5), 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 stage and the following stages into the same CUDA kernel 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. 5) 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



Computational Analysis	
compute work	$H \times W \times z$
memory reads	$H \times W \times z$
memory writes	$h \times w \times z$
arithmetic intensity	*constant*

Parallel Scheme Analysis	
work per thread	$s + log_2(t)$
# of threads	$h \times W \times z$
global memory loads	$H \times W \times z$
global memory stores	$h \times w \times z$

Fig. 4: 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 $w = \frac{W}{t}$.

algorithm proposed by Harris *et al*. [28], 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, [4], 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 3.3 and section 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 Dynamic Programming stage

The Dynamic Programming (DP) computation stage, both on the original model of Slanted Stixels [12] and our proposal, has the higher computational complexity (*c.f.* left table on fig. 5), and for practical cases is the most time-consuming step. Our proposed design exploits the locality of the data accesses to move most memory 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. 5).

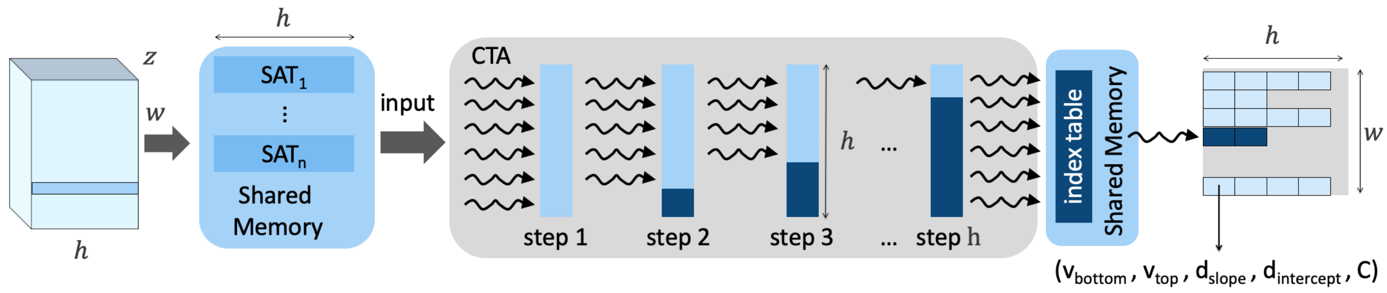
The DP recurrence shown in eq. (9) defines how to calculate the minimum cost of a problem OB^k with k pixels using the results computed for smaller problems. The most straightforward parallel design option (A) is to use k+1 of the CTA threads to cooperatively compute OB^k (and GR^k and SK^k) for each 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. 5.

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



Computational Analysis				
	SAT gen.	DP [12]	DP Ours	Backtrack
comp. work	$w \times h \times C$	$w \times h^3$	$w \times h^2$	$w \times n$
mem. reads	$w \times h \times z$	0	0	0
mem. writes	$w \times h \times z$	0	0	$w \times n$
arith. intens.	*const*	h^2	h	*const*

Parallel Scheme Analysis		
	[12]	Ours
work per thread	up to h^2	up to h
# of threads	$w \times h$	$w \times h$
glob. mem. loads	$w \times h \times z$	$w \times h \times z$
glob. mem. stores	$w \times h$	$w \times h$

Fig. 5: Stages for generating SATs, Dynamic Programming (DP) computation, and Backtracking, which are fused into the same CUDA kernel. The computational analysis and parallel scheme are shown for the original proposal [12] and our proposal. w, h, z are defined in fig. 4. v_{bottom}, v_{top}, d_{slope}, $d_{intercept}$, and C define the Stixel properties: bottom & top row, depth model and semantic class.

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), 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 [11], [12] is higher ($\mathcal{O}(w \times h^3)$) than that of our novel depth model ($\mathcal{O}(w \times h^2)$), described in section 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.

4.5 Backtracking and Data compaction

The backtracking step is an inherently sequential process for each column. As described in section 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. 5. 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. 5, we fuse the backtracking stage with the two previous stages into the same CUDA kernels. 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.

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 [12]. For that purpose, we evaluate on both synthetic and real data, *c.f.* section 5.1.1, and report quantitative and qualitative results, *c.f.* section 5.1.3. We also show and discuss the performance advantage of our novel depth model for GPUs and show quantitative results, *c.f.* section 5.2.

5.1 Accuracy and Compression experiments

5.1.1 Datasets

We use two datasets with real images, KITTI 2015 [29], [30] and Cityscapes [21], and one dataset with synthetic images, SYNTHIA-SF (SYNTHIA San Francisco) [11].

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.

5.1.2 Experiment Details

Slanted Stixels [12] serves as **baseline** for the comparison with our proposal, *c.f.* section 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) [31] and pixel-level semantic labels computed by a fully convolutional network (FCN) [32]. The parameters are taken from [3] for fair comparison. For the same reason, we use the same FCN architecture from [3]. However, FCN weights are not the same, but accuracy (IoU) of the input is provided for reference. Similarly, SGM implementation details differ between our implementation and those of previous works, therefore, input disparity accuracy is also provided in the respective tables.

Following [12], we use the known camera calibration to obtain expected μ^a_{ground} and μ^b_{ground}. We assume that objects are vertical and set $\sigma^b_{object} \to 0, \mu^b_{object} = 0$, 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 [29], 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 [12]. 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 [33].

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

5.1.3 Results

The quantitative results of our proposal and baseline as described in section 5.1.2 are shown in table 2 and table 3 .

The first observation, taken from table 2, 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 3 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 2: Data compression measured in pixels per Stixel of our method and [12]. We evaluate on three datasets: KITTI 2015 [29], Cityscapes [21] and SYNTHIA-SF [11], *c.f.* section 5.1.1.

	Stixel resolution: 8×8		Stixel resolution: 4×4	
Dataset	Slanted [12]	Ours	Slanted [12]	Ours
KITTI 2015	587	572	254	242
Cityscapes	1105	877	475	331
SYNTHIA-SF	1439	1379	637	606

The observations from the quantitative evaluation are confirmed also in the qualitative results, *c.f.* fig. 6.

Figure 7 shows the distribution of the disparity errors on the individual images of the KITTI dataset when using both Slanted Stixels and our model. The distributions of the errors are very similar, with most of the images containing a small number of errors and some images containing more errors. The last chart shows a histogram of the relative differences of the disparity error for each individual image. The differences in the disparity error are relatively small; sometimes they benefit our proposal and to a somewhat greater extent they harm our proposal.

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. All the metrics are measured using NVIDIA performance tools. We assume the input 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 [34]). 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.

5.2.1 Results

Figures 8a and 8b show the performance throughput (frames per second, or fps) of the CUDA code implementing both Slanted Stixel models (original and ours), on two GPU systems, 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. 8a), 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. 8b). On the Tegra Xavier and for a low Stixel resolution (8×8), our method is $3.4\times$ faster than Slanted Stixels [12] (344.3 fps vs

TABLE 3: Accuracy of our method compared to Slanted Stixels [12], input SGM and FCN. We evaluate on three datasets: KITTI 15 [29], Cityscapes [21] and SYNTHIA-SF [11] using these metrics: Disparity Error (lower is better) and Intersection over Union (higher is better), *c.f.* section 5.1.1 and section 5.1.2. *Ours* is detailed in section 3.4.

		Input		Stixel resolution: 8×8		Stixel resolution: 4×4	
Metric	Dataset	SGM	FCN	Slanted Stixels [12]	Ours	Slanted Stixels [12]	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

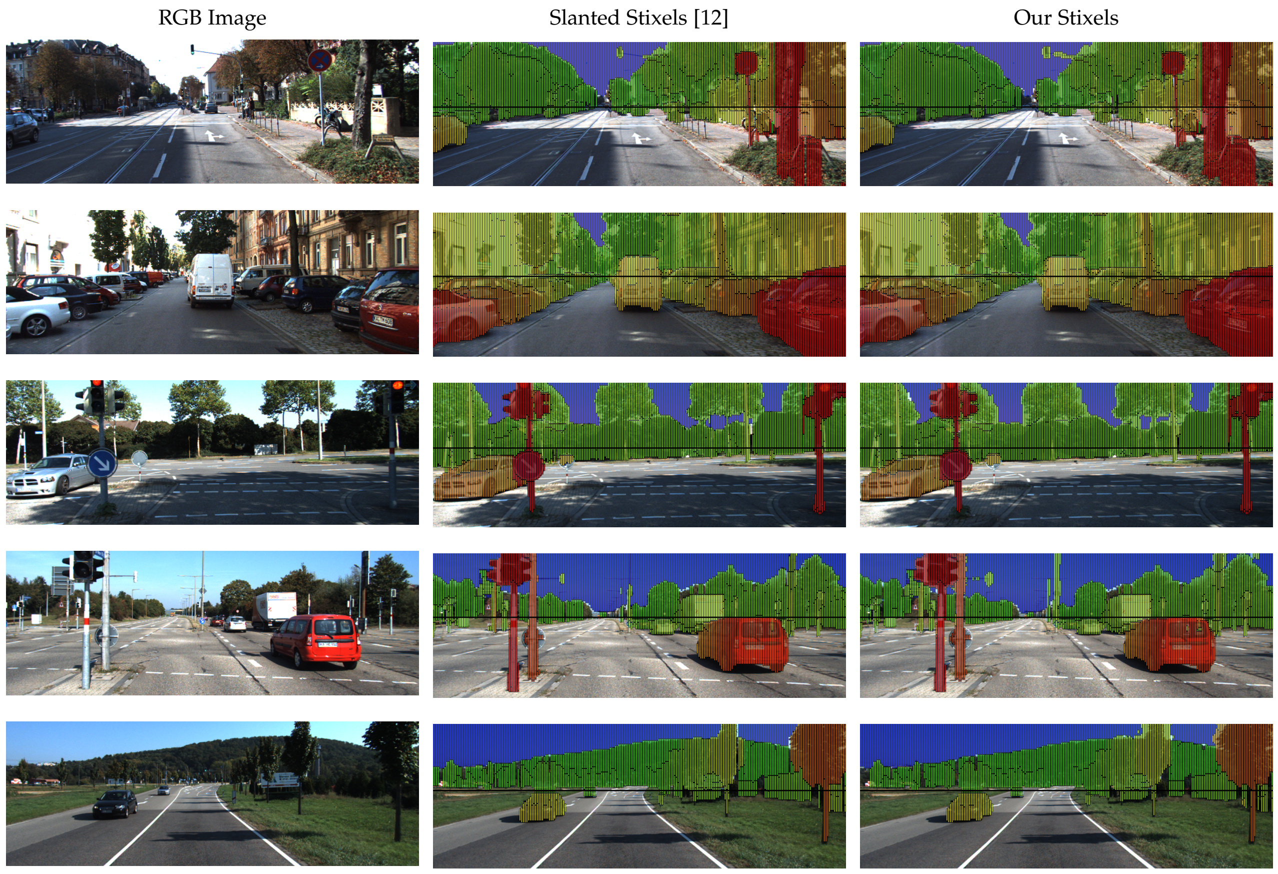
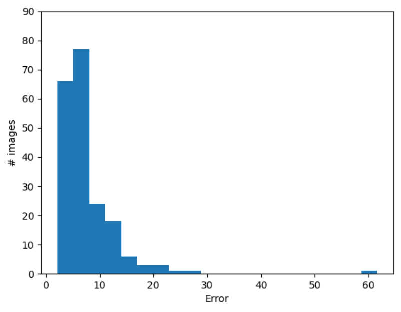


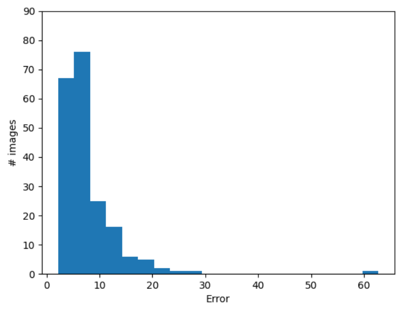
Fig. 6: Exemplary outputs on real data (KITTI 2015): RGB Image (left), Slanted Stixels [12] (center) and Our Stixels (right) representations are shown. Color encodes the distance from close (red) to far (green). As we can see, the representation of our proposal is visually similar to the baseline.

102.4 fps). A higher resolution (4×4) provides more accurate segments (*c.f.* table 3), but while our proposal achieves practical frame rates (92.3 fps), the implementation following [12] 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 a six-core Intel i7-6800K CPU [12]. Our implementation reaches 344.3 fps on a Tegra Xavier, *i.e.* more that 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 9 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


(a) Disparity errors with Slanted Stixels


(b) Disparity Errors with our method

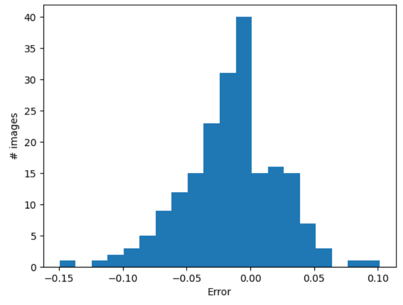

(c) Relative differences of the disparity error

Fig. 7: Histogram of the average disparity errors per image (KITTI 15 dataset [29] and 4×4 Stixel resolution) when using (a) Slanted Stixels [12] and (b) our model; and (c) histogram of the relative differences of the disparity error for each individual image (Slanted Stixels - Ours, meaning that negative numbers indicate the Slanted Stixels outperforms.

TABLE 4: 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

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 [12] on the Tegra Xavier. Our implementation achieves higher IPC ratios (1.31× and 1.33× better) on each SM, which means that our approach exhibits more parallelism. But most of the speedup is due to a 5.8× and 8.7× reduction on the total number of machine instructions executed by the GPU (these data can be derived from the results in fig. 9). This corroborates the better algorithmic scalability of our approach.

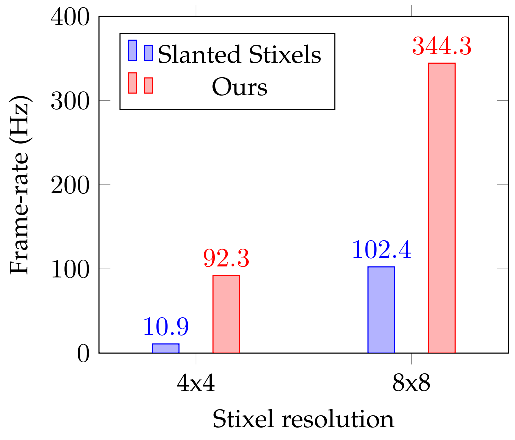
We now assess the performance differences when using both GPUs. The Tegra Xavier contains 8 Volta SMs (512 cores) running at a slightly lower clock frequency than the 4 Pascal SMs (256 cores) in the older Tegra X2 (*c.f.* table 4). This means a potential raw performance advantage of 1.88×. The speedup 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), which improves the IPC ratio (from 1.34× to 1.48×), 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. 9). The speedup of the *Down-sampling & Transpose* stage is around 3.6×, closer but higher than the 2.3× 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.

6 Conclusion

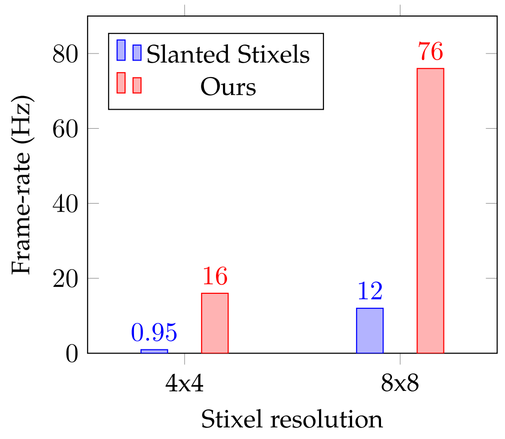
We have described and assessed the performance of the first GPU-accelerated implementation of Slanted Stixels and we show that our algorithmic proposal is efficient for GPU parallelization. Our proposal achieves real-time performance for realistic problem sizes, proving that the low-power envelope and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



(a) Computed on NVIDIA Tegra Xavier



(b) Computed on NVIDIA Tegra X2

Fig. 8: Frame-rate of our method compared to Slanted Stixels [12] for 2048 $\times$ 1024 image resolution on the NVIDIA Tegra Xavier and Tegra X2 embedded GPUs.

Acknowledgments

The contribution of Daniel Hernández, Antonio Espinosa and Juan C. Moure has focused on the GPU acceleration of Stixels, thus, they acknowledge the support of the project TIN2017-84553-C2-1-R of the Ministerio de Economía, Industria y Competitividad. The contribution of Daniel Hernández, David Vázquez, and Antonio M. López has focused on the Slanted Stixels concept itself from the Computer Vision viewpoint, consequently Antonio acknowledges the support of project TIN2017-88709-R (MINECO/AEI/FEDER, UE). Antonio M. López also thanks the support by ICREA under the ICREA Academia programme. As CVC member, Antonio M. López also thanks the Generalitat de Catalunya for its CERCA Program and the ACCIO agency.

References

[1] H. Badino, U. Franke, and D. 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.

[2] D. Pfeiffer and U. 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. [Online]. Available: http://dx.doi.org/10.5244/C.25.51

[3] L. Schneider, M. Cordts, T. Rehfeld, D. Pfeiffer, M. Enzweiler, U. Franke, M. Pollefeys, and S. Roth, "Semantic stixels: Depth is not enough," in *2016 IEEE Intelligent Vehicles Symposium, IV 2016, Gotenburg, Sweden, June 19-22, 2016*, 2016, pp. 110–117. [Online]. Available: http://dx.doi.org/10.1109/IVS.2016.7535373

[4] D. Hernandez-Juarez, A. Espinosa, J. C. Moure, D. Vázquez, and A. M. López, "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. [Online]. Available: https://doi.org/10.1109/WACV.2017.122

[5] R. Benenson, R. Timofte, and L. J. V. 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. [Online]. Available: http://dx.doi.org/10.1109/ICCVW.2011.6130495

[6] M. Cordts, L. Schneider, M. Enzweiler, U. Franke, and S. Roth, "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.

[7] M. Cordts, T. Rehfeld, L. Schneider, D. Pfeiffer, M. Enzweiler, S. Roth, M. Pollefeys, and U. Franke, "The stixel world: A medium-level representation of traffic scenes," *Image and Vision Computing*, pp. –, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0262885617300331

[8] O. Ignat, "Disparity image segmentation for free-space detection," in *2016 IEEE 12th International Conference on Intelligent Computer Communication and Processing (ICCP)*, Sept 2016, pp. 217–224.

[9] D. Levi, N. Garnett, and E. 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. [Online]. Available: http://dx.doi.org/10.5244/C.29.109

[10] D. A. P. Carrillo and A. Sutherland, "Fast obstacle detection using sparse edge-based disparity maps," in *3D Vision (3DV), 2016 Fourth International Conference on*. IEEE, 2016, pp. 66–72.

[11] D. Hernandez-Juarez, L. Schneider, A. Espinosa, D. Vázquez, A. M. López, U. Franke, M. Pollefeys, and J. C. Moure, "Slanted stixels: Representing san francisco's steepest streets," in *British Machine Vision Conference, BMVC 2017, London, UK, September 4-7, 2017*, 2017.

[12] D. Hernandez-Juarez, L. Schneider, P. Cebrian, A. Espinosa, D. Vazquez, A. M. Lpez, U. Franke, M. Pollefeys, and J. C. Moure, "Slanted stixels: A way to represent steep streets," *International Journal of Computer Vision*, pp. 1–16, 9 2019. [Online]. Available: https://doi.org/10.1007/s11263-019-01226-9

[13] T. M. Hehn, J. F. P. Kooij, and D. M. Gavrila, "Instance stixels: Segmenting and grouping stixels into objects," in *2019 IEEE Intelligent Vehicles Symposium, IV 2019, Paris, France, June 9-12, 2019*. IEEE, 2019, pp. 2542–2549. [Online]. Available: https://doi.org/10.1109/IVS.2019.8814243

[14] R. Benenson, M. Mathias, R. Timofte, and L. J. V. Gool, "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.

[15] M. Enzweiler, M. Hummel, D. Pfeiffer, and U. Franke, "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. [Online]. Available: http://dx.doi.org/10.1109/IVS.2012.6232137

[16] F. Erbs, B. Schwarz, and U. 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. [Online]. Available: https://doi.org/10.5244/C.26.71

[17] M. Muffert, N. Schneider, and U. 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. [Online]. Available: http://dx.doi.org/10.1109/CRV.2014.11

[18] D. Pfeiffer and U. Franke, "Modeling dynamic 3d environments by means of the stixel world," *IEEE Intell. Transport. Syst.*

GPU Kernel	Stixel resolution: 8×8 Time (ms)	IPC	Stixel resolution: 4×4 Time (ms)	IPC
Down-sampling & Transpose	1.80	1.94	2.65	2.14
Dynamic Programming [12]	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 Time (ms)	IPC	Stixel resolution: 4×4 Time (ms)	IPC
Down-sampling & Transpose	6.38	3.26	9.85	3.68
Dynamic Programming [12]	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

Fig. 9: 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. [12]). *IPC* is the ratio of machine instructions executed per clock cycle and per SM. Image size is 2048×1024.

Mag., vol. 3, no. 3, pp. 24–36, 2011. [Online]. Available: https://doi.org/10.1109/MITS.2011.942207

[19] T. Scharwächter, M. Enzweiler, U. Franke, and S. Roth, "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.

[20] X. Li, F. Flohr, Y. Yang, H. Xiong, M. Braun, S. Pan, K. Li, and D. M. Gavrila, "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. [Online]. Available: https://doi.org/10.1109/IVS.2016.7535515

[21] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "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. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2016.350

[22] V. Dhiman, A. Kundu, F. Dellaert, and J. J. Corso, "Modern MAP inference methods for accurate and fast occupancy grid mapping on higher order factor graphs," in *ICRA*, 2014.

[23] D. Nuss, T. Yuan, G. Krehl, M. Stuebler, S. Reuter, and K. Dietmayer, "Fusion of laser and radar sensor data with a sequential monte carlo bayesian occupancy filter," in *IV*, 2015.

[24] S. Thrun, "Robotic mapping: A survey," in *Exploring Artificial Intelligence in the New Millenium*, G. Lakemeyer and B. Nebel, Eds. Morgan Kaufmann, 2002.

[25] D. Pfeiffer, S. Gehrig, and N. 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. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2013.45

[26] D. Pfeiffer, "The stixel world - a compact medium-level representation for efficiently modeling three-dimensional environments," Ph.D. dissertation, Hu Berlin, 2014.

[27] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, vol. 1. IEEE, 2001, pp. I–I.

[28] M. Harris, S. Sengupta, and J. D. Owens, "Parallel prefix sum (scan) with cuda," *GPU gems*, vol. 3, no. 39, pp. 851–876, 2007.

[29] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for Autonomous Driving? the KITTI Vision Benchmark Suite," in *Conference on Computer Vision and Pattern Recognition*, 2012.

[30] H. A. Alhaija, S. K. Mustikovela, L. Mescheder, A. Geiger, and C. Rother, "Augmented reality meets deep learning for car instance segmentation in urban scenes," in *British Machine Vision Conference (BMVC)*, 2017.

[31] H. Hirschmüller, "Stereo processing by semiglobal matching and mutual information," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 2, pp. 328–341, 2008. [Online]. Available: http://dx.doi.org/10.1109/TPAMI.2007.1166

[32] J. Long, E. Shelhamer, and T. 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. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2015.7298965

[33] M. Everingham, S. M. A. Eslami, L. J. V. Gool, C. K. I. Williams, J. M. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective," *International Journal of Computer Vision*, vol. 111, no. 1, pp. 98–136, 2015. [Online]. Available: http://dx.doi.org/10.1007/s11263-014-0733-5

[34] D. Hernandez-Juarez, A. Chacón, A. Espinosa, D. Vázquez, J. C. Moure, and A. M. López, "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. [Online]. Available: http://dx.doi.org/10.1016/j.procs.2016.05.305

Daniel Hernandez-Juarez received the BSc degree in Computer Science from Universitat Autonoma de Barcelona (UAB) in 2014, and the MSc on Computer Vision at the UAB, UPC, UPF and UOC in 2015. Currently, he is working towards a PhD on Computer Vision supervised by Dr. Juan Carlos Moure and Dr. David Vazquez. His focus is to improve 3D perception algorithms and adapt them to GPU devices. To this end, he is immersed in the study of the Stixel World. He is interested in self-driving cars, deep learning techniques, 3D perception and embedded systems.

Antonio Espinosa is an associate professor in the Computer Architecture and Operating Systems Department at the Universitat Autonoma de Barcelona. During the last 10 years, he has participated in several European and national projects related to Computer Science, high-performance computing and computational accelerator systems in collaboration with a number of companies and research institutions.

David Vazquez is a Fundamental Research Scientist at Element AI, where he works on computer vision. Previously he was a post-doctoral researcher at Computer Vision Center of Barcelona (CVC) and Montreal Institute of Learning Algorithms (MILA) and Assistant Professor in the Department of Computer Science at the Autonomous University of Barcelona (UAB). He is an expert in machine perception for autonomous vehicles and on domain adaptation from simulation to real-world environments.

Antonio M. López is the principal investigator of the Autonomous Driving lab of the Computer Vision Center (CVC) at the Univ. Autònoma de Barcelona (UAB). He has also a tenure position as associated professor at the Computer Science department of the UAB. Antonio has a long trajectory carrying research at the intersection of computer vision, computer graphics, machine learning and autonomous driving. Antonio has been deeply involved in the creation of the SYNTHIA dataset and the CARLA open-source simulator, both for democratizing autonomous driving research. He is actively working hand-on-hand with industry partners to bring state-of-the-art techniques to the field of autonomous driving. Currently, Antonio is granted by the Catalan ICREA Academia program.

Juan C. Moure is an associate professor in the Computer Architecture and Operating Systems Department at the Universitat Autonoma of Barcelona (UAB), where he teaches Computer Architecture, Performance Engineering and Parallel Programming. His current research interests include massive parallel architectures, programming, and algorithms, mainly focused on Computer Vision, Signal Processing and Bioinformatics applications. He is the author of more than 50 papers, and has participated in several European and Spanish projects related to high-performance computing.