Title
Feature-Based Speed Limit Sign Detection Using a Graphics Processing Unit

Authors
Glavtchev, Vladimir
Muyan-Ozcelik, Pinar
Ota, Jeffrey M.
et al.

Publication Date
2011

DOI
10.1109/IVS.2011.5940539

Peer reviewed

Feature-Based Speed Limit Sign Detection Using a Graphics Processing Unit

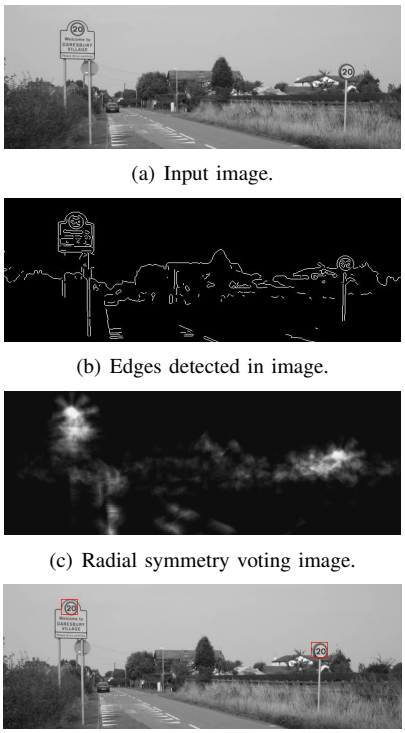
Vladimir Glavtchev, Pınar Muyan-Özçelik, Jeffrey M. Ota, and John D. Owens

***Abstract*— In this study we test the idea of using a graphics processing unit (GPU) as an embedded co-processor for real-time detection of European Union (EU) speed-limit signs. The input to the system is a set of grayscale videos recorded from a forward-facing camera mounted in a vehicle. We introduce a new technique for implementing the radial symmetry detector (RSD) efficiently using the native rendering capabilities of a GPU. The technique maps the algorithms to the hardware such that the detection of speed-limit sign candidates is significantly accelerated. The system reaches up to 88% detection rate and runs at 33 frames per second on video sequences with VGA (640x480) resolution on an embedded system with an Intel Atom 230 @ 1.67 GHz CPU and a NVIDIA GeForce 9400M GS GPU.**

I. Introduction

A target of research since the mid 1980's, computer-based driver assistance has been a key technological goal for automotive manufacturers. A large part of the research has focused on vision-based systems using a camera mounted inside the vehicle. Despite its long history of research, a lot of the driver-assistance ideas have never left the laboratory because of their computational cost-prohibitiveness. However, with commodity hardware rapidly increasing in performance, it is worth spending some effort mapping existing algorithms efficiently onto current-generation commodity hardware.

An example driver-assistance application that has been proven effective and efficient is speed-limit sign detection. Implementations of some of the algorithms for speed-limit sign detection have been gradually optimized and reached real-time performance, but only on high-end processors or highly-specific architectures. Here we present a new technique to enable real-time detection on resource-constrained hardware using a low-end GPU that can be embedded in a car. The input to our system is a 640x480 video sequence of EU roads taken from a moving vehicle. The video is processed in real-time to detect speed-limit signs as depicted in Figure 1.



(a) Input image.

(b) Edges detected in image.

(c) Radial symmetry voting image.

(d) Detected speed-limit sign candidates.

Fig. 1. Detection of circular speed-limit signs. (a) The original input image. (b) Edges found using a 3x3 Sobel with thresholding. (c) Voting intensity map with bright areas representing circular features. (d) Corresponding speed-limit sign candidates found using the detection algorithm.

A. Our Contribution

We implement a hardware-accelerated version of the radial symmetry detector as presented by Loy and Barnes [1] to achieve real-time performance in detecting speed-limit signs on a graphics processor. We use the Compute Unified Device Architecture (CUDA) programming model in conjunction with OpenGL to extract maximum parallelism from the detection process. To our knowledge, this is the first real-time implementation of the radial symmetry detector using embedded-level hardware. Previous implementations have focused mainly on optimizing the algorithms for executing on a CPU architecture. This has offered a few close to real-time solutions, but only on high-end processors with large memory and processing resources. Missing from these solutions has been the exploitation of parallelism inherent in the algorithms used for detection. Thus, in order to achieve the desired performance, we focused on the following key points: 1.) Using a platform whose architecture reflects the natural parallelism of the computer-vision techniques used in traffic-sign detection. The GPU's massively parallel computing resources are a good fit. 2.) Utilize the hardware as effectively as possible using novel approaches and known state-of-the-art techniques where needed. Our RSD implementation exploits hardware acceleration in ways such as massively-parallel vertex computations, texture caching, and rendering using the native rasterization hardware of the GPU. 3.) With those in mind, we pick hardware which is inexpensive, widely-available, and most importantly, available in the embedded space. In our tests, we target an embedded CPU-

GPU platform, an under-studied environment for embedded vision-based driver assistance. Thus, our work contributes to the field and helps fill in the gaps.

B. Choosing the Platform

We present a platform consisting of a coordinating host CPU and a GPU as the main processor. Our choice for the GPU as an embedded co-processor is motivated by several of its characteristics. GPUs today are massively parallel, programmable, and are widely available. A GPU replaces the need for application-specific processors as it increases reusability and allows for time-sharing between processing tasks. The hardware is also largely scalable as its variants include more resources while keeping the basic architecture the same and the programming paradigm completely uniform.

The GPU is an obvious choice for a co-processor in many computer-vision detection, recognition, and augmented reality schemes. This is due in part to the exposure of its programmable features through C-like languages. Before general purpose languages such as CUDA, Brook, and OpenCL became widely available, many experts in the computer-vision field were already harnessing the power of GPUs through shader languages like Cg and GLSL. Fung and Mann [2] demonstrated several algorithms ported to the GPU using Cg that are accelerated when compared to their CPU counterparts or made possible in real-time due to the processing power available in GPUs. Projects such as OpenVIDIA [2] and GPU4VISION [3] have developed a steadily growing community of programmers. Inspired from the success of GPU-based computer-vision projects, we choose the GPU as our detection co-processor of choice due to its highly-parallel architecture, large local and global memory, and its programmability.

II. Previous Work

A. Grayscale Versus Color Detection

Detection using grayscale images inherently relies solely on the characteristics of the objects present in the scene. Characteristics such as edges, corners, and gradient angles can be used either individually or in groups to detect target shapes. Not having color information restricts a given recognition technique to focus only on the characteristics that define the detection targets. However, that is in fact the main advantage of using grayscale images, argue Gavrila [4] and Barnes and Zelinsky [5]. These characteristics are invariant to lighting conditions and fading of sign colors over time [6]. Garcia et al. [7] claims that grayscale approaches have an advantage over color-based detection because the latter is less precise due to the fact that the blue and red color spaces can overlap in signs that are mostly white (as is the case with speed-limit signs). Examples of color-based approaches include studies proposed by Priese et al. [8] and Zheng et al. [9].

B. Template-based Versus Feature-based

Detection using template-based techniques is usually performed by "sliding" (a convolution in the spatial domain) a target template over the entire input scene and computing a value indicative of the similarity between the input and the target. Piccioli et al. [10] use cross-correlation, which is the most commonly applied technique. This method has the disadvantages of performing expensive convolutions and often requiring multiple passes. The convolutions can be avoided when working in the frequency domain, as proposed by Javidi et al. [11]. This technique relies on the ability to quickly perform FFTs. This also might require multiple passes to detect signs of different sizes, orientations, or different traffic signs in general. A major disadvantage is that objects which are of shape or orientation not present in the target template set can be missed altogether.

On the other hand, feature-based approaches utilize prominent characteristics in the input scene such as corners, line segments, and edges. Techniques of different complexities can be used to extract these features such as a simple Sobel filter, a more involved Canny edge detector, or an intelligent edge-chaining algorithm such as Pavlidis's used by Piccioli et al. [10].

Traffic-sign detection algorithms have naturally sought to take advantage of the geometry of target signs. For rectangular and otherwise polygonal (diamond, octagon, etc.) signs, algorithms targeting straight edges that form line segments have been used widely. Perhaps the most widely used algorithm has been the general Hough Transform (HT). Examples include detection of stop-signs, lane markers, and diamond curve signs [12]. For circular signs, the circular HT is the preferred algorithm and is an adapted version of the general transform. Both forms of the HT require the use of sine and cosine computations for each input pixel and the algorithms are considered computationally expensive and memory-demanding [13]. In a more holistic approach, the General Symmetry Transform (GST) [14] measures the contribution of edge pixel pairs to a central point (a one-to-many approach), thus searching for edges of potential geometric shapes which exhibit horizontal or vertical symmetry. However, the computational cost of the GST is also high as each pixel searches a neighborhood of a varying size for edges contributing to a symmetry around it.

Loy and Zelinsky [15] introduced the radial symmetry detector (RSD) which inverts the symmetry-seeking technique of the GST to a many-to-one approach. Individual pixels vote for a common center of symmetry with their results accumulating in a common voting map. In the first known application of the RSD, Loy and Barnes [1] demonstrate the algorithm's robustness in an implementation of a driver-assistance system capable of recognizing octagonal stop signs, triangular warning signs, and diamond indicator signs. More recently, Barnes and Zelinsky [16] show that the algorithm can handle up to 30% noise while maintaining high detection rates and approaching real-time performance.

C. Focus of Recent Literature

Even though automotive recognition systems are meant to run on embedded platforms in an automobile, recent literature has not paid sufficient attention to providing techniques

which can run on such platforms. Previous studies which perform real-time speed-limit-sign recognition mostly use dual-core desktop or laptop architectures [6], [17] as their running platforms. Even in one of the most recent studies, Barnes and Zelinsky [16] demonstrate a system which runs in real-time, but the implementation only does so on a server-grade Intel Xeon 3.40 GHz machine. An approach which is most-closely targeted for the embedded space is introduced by Coersmeier et al. [18]. The authors propose a real-time speed-limit-sign recognition system utilizing four ARM processors in parallel to test the speedup of a traffic sign recognition system for future mobile devices.

III. Methods

A. Preprocessing

In our approach, the detection process begins with an input image of 640x480 pixels. For preprocessing, a 3x3 Sobel edge detector is applied to the entire input scene. The result is an image of gradient angles of pixels whose gradient magnitude is larger than a specified threshold. All other pixels are marked as invalid. The gradient image is then passed on to the next stage.

B. Shape Detection

Each element with a non-zero gradient angle casts a vote (increments an element on a cumulative voting image V) for a potential circle center a distance r away (where r is the radius of the circle which we are targeting for detection) in the direction of its gradient angle. Since we do not know the direction of the centroid of the targeted circle, voting is performed both in the direction of the gradient and 180 degrees across from it. In order to take into account skewing due to viewing perspective or out-of-plane sign rotation, the vote is extended to a line of pixels (of width w) that are oriented perpendicularly to the voting pixel's gradient angle. Figure 2 (left) shows line-voting for one specific radius. The voting width is defined by $w = \text{round}(r \tan \frac{\pi}{n})$ where n represents the number of sides in the n-sided polygon. In this case, we target octagonal shapes ($n = 8$) as an approximation to circles. Treating circles as octagons simplifies the vertex computations as it avoids sine and cosine evaluations for each vertex depending on the different gradient angles. Octagonal approximation allows us to have votes cast only in directions separated by $\frac{360}{8}$ degrees. This is an optimization to using $n = \infty$, as should be the case with circles, which have an infinite number of sides.

Since speed-limit signs will vary in radius depending on their distance away from the camera, we target a range of possible radii from R_{min} to R_{max}. Noticing that w increases with increasing r, the overall voting pattern for each pixel becomes triangular if $R_{\text{min}} = 0$. This assumption simplifies the shape of the voting pattern and has performance implications as discussed in the Implementation section. Thus, each pixel casts its votes in a triangular pattern on image V, as shown in Figure 2 (middle). The overall voting image is formed by summing the votes of all edge pixels, as Figure 2 (right) shows. The points voted for are called affected pixels

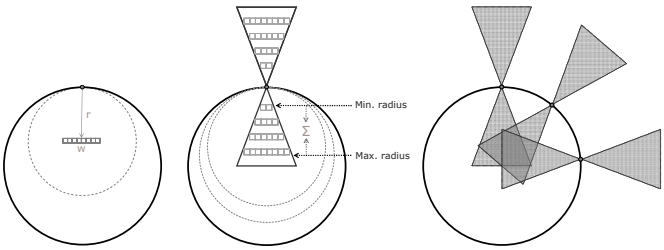


Fig. 2. (Left) Votes cast by one edge pixel in the direction of its gradient at a distance of the targeted circle radius. Omitted in this diagram are the votes cast 180 degrees opposite of the gradient angle. (Middle) Triangular estimation of each edge pixel's voting pattern. (Right) Votes accumulate where the triangles overlap to create a combined voting image.

and are defined as $p_{aff}(p) = p \pm \text{round}(rg(p))$, where $g(p)$ is the unit gradient for pixel p. Figure 2 illustrates the steps of the detection algorithm and the possible speed-limit sign candidates extracted from the input image.

C. Classification and Tracking

The coordinates of the center as obtained during detection are used to segment the regions of interest (ROIs) from the image. Each of the selected ROIs are then classified using a FFT template-matching approach. The classification provides a confidence value for the best match for each candidate along with an estimated size and in-plane rotation. The classifier is used only to validate our detection stage and is discussed in a prior publication [19].

One-frame results do not provide enough confidence that a detected candidate is in fact a real sign. Therefore, we apply a temporal integration tracking technique that accumulates candidate confidence over a range of several frames. Temporal integration is implemented as a running tracking table of sign candidates across the last ten (the average number of frames in which a sign appears in our database) frames. Sign scaling and orientation are factored in; confidence values of signs appearing larger in size than their last appearance in a frame are multiplied by a scaling factor. This reduces the possibility that artifacts present in several consecutive frames but remain the same or decrease in size will obtain high confidence levels and be falsely recognized as signs. The orientation factor is used similarly and prevents artifacts with a largely varying rotation from affecting tracking of real signs.

D. Extension: Finding the Size

Due to the fact that we span the entire range of targeted radii using the triangular voting patterns, the size of the detected signs are not recovered, only their location. If needed, the size of the detected sign could be recovered with an incremental voting process using a set of voting images, where each voting image targets a specific range of radii. The size is then recovered in two steps: a.) finding the maxima in an accumulated image S of all the results and b.) finding the voting image V_i (where V_i represents a voting image for

a specific range of radii) which has a maximum with a high magnitude and also appears in S.

IV. Implementation

From our discussion thus far, it can be seen that both preprocessing and detection are highly parallel processes. Each algorithm operates on a per-pixel basis with no data or control dependencies. Thus we map both of these algorithms entirely to the GPU in order to take advantage of its many processing cores.

The incoming image is copied over to the GPU's video memory and is loaded as a texture, as shown in Figure 3. A Sobel filter is used for preprocessing and is implemented as a CUDA kernel which runs per pixel. Each pixel samples its immediate neighbors (3x3 pixel area) using fast texture sampling. The input image remains as a texture and is unmodified, as it might be needed for classification in case of successful detection. Results of the Sobel filter (an edge map containing gradient angles) are saved to global video memory. Then, a per-pixel radial symmetry kernel runs using the gradient angle image as its input. Each element with a valid gradient angle calculates its voting areas. Values calculated at this stage are (x,y) coordinate pairs for the vertices of the voting triangles. Each pixel stores its result in a Vertex Buffer Object (VBO). Once the radial symmetry kernel finishes, OpenGL uses this VBO to draw triangles defined by the coordinate pairs. Using CUDA-OpenGL interoperability [20], there are no memory transfers between these two stages.

With blending enabled, each triangle is then rendered with a high transparency (lowest non-zero alpha) value. The graphics hardware blends together all overlapping triangles, which causes these areas to appear brighter. An intensity map is produced as a result with accumulated votes of the radial symmetry stage. A large gain here comes from hardware drawing and blending done automatically by the GPU. A high number of overlapping incremental votes causes contention and serialization in most architectures, but the GPU hardware is optimized for this operation. In this stage, our technique takes full advantage of the present resources in a highly-efficient voting strategy. Using the CUDA-OpenGL interoperability is crucial here to avoid unnecessary memory transfers and achieve maximum performance.

For the final stage (maxima detection), we use a parallel maximum reduction implementation. The voting image is split into 4,800 blocks of 8x8 pixels and the highest value of each block is found. The results are then again divided in blocks of 8x8, which now consist of the maxima of the last stage. Reduction is performed iteratively until there are only 80x60 maximum values remaining. This process is highly parallelizable as a find maximum operation can be performed in parallel among as many blocks of the image as the hardware will allow simultaneously. Resulting candidate centroid coordinates are copied back to the host processor. On the host, a reduction is performed to locate the desired number of blocks with the highest values. The maximum-reduction which is performed on the CPU is only a 80×60 pixel block and executes fast enough on the host processor as to not affect the overall runtime of the detection process.

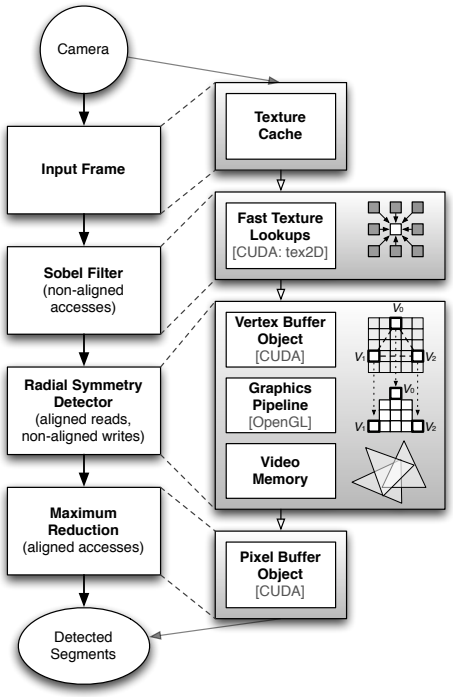


Fig. 3. Computation (left) and data (right) flow through the speed-limit recognition pipeline.

A. Memory

A target for efficiency in the embedded space is minimizing expensive data transfers. The overhead latency of data transfers between devices is largely independent of the data transfer size. Overhead cost can be amortized if the data transfer is relatively large. In our implementation, the incoming video image is copied to the GPU's video memory at arrival and remains there for the duration of the detection process. In the preprocessing stage, memory is only accessed using fast texture lookups. During the radial symmetry stage, input values are loaded into shared memory and the results are written out to global memory in large blocks. During the voting stage of the radial symmetry algorithm, the results from global memory are mapped to a VBO which eliminates the need for any memory transfers. Voting results are accumulated in the video frame buffer which is mapped to a pixel buffer object (PBO). These results can be readily used for the maximum reduction and to be displayed visually on a screen. Only a minimal memory transfer from the GPU to the host processor is used to transfer the potential candidate coordinates for the final reduction step.

1) Hardware Acceleration: Our implementation takes advantage of hardware acceleration in several key ways. First, the preprocessing stage uses texture sampling for accessing its input. Memory accesses through texture look-ups have several benefits when compared to global or shared memory reads (as per the CUDA Programming Guide [20]). Textures

are cached, which allows for higher bandwidth provided that there is locality in the fetching patterns. In the case of the Sobel kernel, the texture is accessed in 3x3 blocks which exhibits good spatial locality. Another advantage is that data cached as a texture does not exhibit the same loss in performance as do global and shared memory due to uncoalesced accesses and bank conflicts. Lastly, texture fetches have automatic boundary case handling, which eliminates the need for auxiliary functions or extra branch statements that handle the image boundaries.

Hardware acceleration provides the most significant performance benefit in the radial symmetry voting stage. First, the Arithmetic Logic Units (ALUs) in the hardware shaders of the GPU are used to calculate triangle vertex coordinates in parallel. Since RSD is pixel-based without any data dependencies between pixels, as many individual pixels can be processed in parallel as there are ALUs available on the GPU provided that scheduling permits it. The most significant speed-up, however, is in the voting process. Drawing and blending of the voting areas are performed exclusively using the native rasterization hardware. In fact, rasterizers have always been the "secret sauce" of graphics architectures which have differentiated GPUs from CPUs in rendering performance by orders of magnitude.

The last step, parallel maximum reduction, also benefits largely from hardware acceleration. It is performed using a slightly modified version of the very efficient CUDA reduction algorithm from the CUDA SDK. The input from the voting stage is read using fast texture lookups and the results are computed with maximum GPU hardware occupancy.

V. Results and Analysis

The video footage used for testing was chosen to cover as many different environments as present in our database. Environments we consider to be the type of road the vehicle is driving on including its surroundings. We define five different environments: inter-city highway, urban, rural, construction zone, and tunnel.

A. Lighting and Weather

In order to evaluate the system's robustness, it was tested in various lighting and weather conditions. The data set was divided among the following: daylight, nighttime, dawn or dusk, tunnel, and snow. Snow has its own category as it adds an extra difficulty to recognition: snow flakes tend to partially or sometimes fully occlude the camera's view. We believe that our extensive testing in a large range of environments is important to demonstrating the system's robustness. In general, there has been a lack of coverage of lighting conditions in the literature. However, we believe the robustness of the RSD algorithm is highlighted best through its ability to detect signs with high accuracy in conditions where the driver could have a difficult time seeing traffic signs such as at night or on a snowy day. The distribution of the videos across the various lighting and weather conditions is shown in Figure 4.

Environment	No. Videos	Lighting	No. Videos
City	12	Daylight	41
Country road	27	Night	18
Highway	33	Dusk / dawn	8
Construction	6	Digital	9
Tunnel	2	Snow	4
Total	80		80

Fig. 4. Breakdown of lighting conditions in the input scenes. Videos containing digital signs are distributed among the other lighting conditions and are not counted separately towards the scene total.

B. System Performance

The system achieved 88% detection rate while executing at 33 frames/s using the host CPU for only issuing commands and the GPU for the main computation. Our base testing platform is an Intel Atom 230 @ 1.67 GHz and a NVIDIA GeForce 9400M GS GPU. Both of these processors are found in low-end laptops and due to their ultra-low power requirements, are an ideal match for an embedded automotive system. The detection rate was achieved on 80 video scenes totaling 42 minutes and 43 seconds filmed at 16.7 Hz on European roads.

Detection rate evaluation was performed using 43 minutes of video footage of roads in the European Union (ex: Germany, UK, France, etc.) under various daytime and nighttime conditions as listed in Figure 4. The footage contains a total of 164 signs, of which 144 were detected correctly for a detection rate of 88%. In order to maximize the detection rate, up to seven candidates per frame were examined. Candidates which return single-frame confidence values above a certain instant threshold are then tracked. Each sign is tracked throughout the interval of its first appearance in the scene until the last frame before its disappearance. A successful detection is recorded when the temporal integration process returns an aggregate confidence value higher than a certain threshold. The classifier returned 0 false positives for the test set video footage.

C. Examples

1) Robustness Evaluation: As previously mentioned, the detection system's robustness is demonstrated through its tolerance of changes in lighting and in-plane rotations. Figures 5(a) and 5(b) show successful detection of signs in the same scene city location both at daylight and at night. Figure 5(c) shows a sign that has been detected successfully despite an in-plane rotation of up to 6 degrees in that frame.

An important case for many European roads is detection of variable digital speed-limit signs. The frequency with which the signs vary makes it hard or impossible for navigation systems to keep up-to-date information in such zones. In addition, digital speed-limit signs are often used in tunnels where a GPS system without pre-stored data could lose satellite connection and be unable to alert the driver. Therefore, the ability of a camera-based system to

(a)

(b)

(c)

(d)

Fig. 5. Robustness of RSD: lighting and in-plane rotation invariance. (a) City scene at daylight. (b) The same city scene at night with street lighting, taillights, and oncoming headlights. (c) Scene with a scene which appears with an in-plane rotation of 6 degrees CW. (d) A detected sign in a tunnel.

detect and identify digital signs becomes beneficial to driver-awareness. Figure 5(d) demonstrates the system's robustness when handling digital signs both during daytime and at night.

VI. Conclusion

We have introduced a novel implementation of the feature-based radial symmetry transform which we applied to speed-limit sign detection. There was no general loss of robustness compared to the original formulation of the technique, as testing in various environments and lighting conditions showed detection rates as high as 88%. Our comprehensive testing exceeds what has been detailed in the literature to this date as a much broader range of driving conditions is covered in our study compared to previous work. Using various techniques of hardware acceleration, our implementation allows for real-time speed-limit sign detection on resource-constrained embedded platforms. Our focus on hardware acceleration and usage of commodity graphics hardware in an embedded environment fills a gap in the current literature of in-vehicle computer vision.

We presented the idea of using a GPU as an embedded general-purpose processor highly capable of computer-vision tasks. The programmability and performance scalability of the GPU make it a processor worthy of examination for embedded automotive platforms. Its programmability can cut down the need for application-specific hardware, design cycle lengths, and overall system costs. It can also allow other automotive tasks to use its resources when available. The native graphics capability is an added advantage and can be used to render a graphical user interface.

Acknowledgments Thanks to BMW, NVIDIA, UC MICRO, and the National Science Foundation (awards CCF-0541448 and CCF-1017399) for funding support. Additional thanks to James Fung and Joseph Stam for their help in architecting the RSD algorithm on the GPU using OpenGL, and John Roberts for his support on tools used for prototyping.

References

[1] G. Loy and N. Barnes, "Fast shape-based road sign detection for a driver assistance system," in *IEEE Intelligent Robots and Systems (IROS)*, vol. 1, 2004, pp. 70–75.

[2] J. Fung and S. Mann, "OpenVIDIA: Parallel GPU computer vision," in *ACM International Conference on Multimedia*, 2005, pp. 849–852.

[3] Graz University of Technology, "GPU 4 Vision," 2010, http://gpu4vision.icg.tugraz.at.

[4] D. Gavrila, "Traffic sign recognition revisited," in *German Association for Pattern Recognition (DAGM) Symposium*, 1999, pp. 86–93.

[5] N. Barnes and A. Zelinsky, "Real-time radial symmetry for speed sign detection," in *IEEE Intelligent Vehicles Symposium*, 2004, pp. 566–571.

[6] F. Moutarde, A. Bargeton, A. Herbin, and L. Chanussot, "Robust on-vehicle real-time visual detection of American and European speed limit signs, with a modular traffic signs recognition system," in *IEEE Intelligent Vehicles Symposium*, 2007.

[7] M. Garcia-Garrido, M. Sotelo, and E. Martm-Gorostiza, "Fast traffic sign detection and recognition under changing lighting conditions," in *IEEE Intelligent Transportation Systems Conference*, 2006.

[8] L. Priese, J. Klieber, R. Lakmann, V. Rehrmann, and R. Schian, "New results on traffic sign recognition," in *IEEE Intelligent Vehicles Symposium*, 1994, pp. 249–254.

[9] Y.-J. Zheng, W. Ritter, and R. Janssen, "An adaptive system for traffic sign recognition," in *IEEE Intelligent Vehicles Symposium*, 1994, pp. 165–170.

[10] G. Piccioli, E. De Micheli, P. Parodi, and M. Campani, "Robust road sign detection and recognition from image sequences," in *IEEE Intelligent Vehicles Symposium*, 1994, pp. 278–283.

[11] B. Javidi, M.-A. Castro, S. Kishk, and E. Perez, "Automated detection and analysis of speed limit signs," University of Connecticut, Tech. Rep. JHR 02-285, 2002.

[12] V. Kamat and S. Ganesan, "A robust Hough transform technique for description of multiple line segments in an image," in *International Conference on Image Processing*, vol. 1, 1998, pp. 216–220 vol.1.

[13] M. Lalonde and Y. Li, "Road sign recognition - survey of the state of art, Tech. Rep. CRIM-IIT Technical report for Sub-Project 2.4, 1995.

[14] D. Reisfeld, H. Wolfson, and Y. Yeshurun, "Context-free attentional operators: the generalized symmetry transform," *Int. J. Comput. Vision*, vol. 14, no. 2, pp. 119–130, 1995.

[15] G. Loy and A. Zelinsky, "Fast radial symmetry for detecting points of interest," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 8, pp. 959–973, 2003.

[16] N. Barnes, A. Zelinsky, and L. S. Fletcher, "Real-time speed sign detection using the radial symmetry detector," *IEEE Transactions on Intelligent Transportation Systems*, vol. 9, no. 2, pp. 322–332, 2008.

[17] C. G. Keller, C. Sprunk, C. Bahlmann, J. Giebel, and G. Baratoff, "Real-time recognition of U.S. speed signs," in *IEEE Intelligent Vehicles Symposium*, 2008, pp. 518–523.

[18] E. Coersmeier, S. Jaborek, P. Paul, M. Bucker, M. Hoffmann, L. Pustina, S. Schwarzer, F. Leder, and P. Martini, "Multicore processing for object recognition in mobile devices," in *Embedded World Conference*, 2008.

[19] P. Muyan-Ozcelik, V. Glavtchev, J. M. Ota, and J. D. Owens, "A template-based approach for real-time speed-limit-sign recognition on an embedded system using GPU computing," in *German Association for Pattern Recognition (DAGM) Symposium*, 2010, pp. 162–171.

[20] NVIDIA Corporation, "NVIDIA CUDA compute unified device architecture," 2009, http://developer.download.nvidia.com/toolkit/docs/NVIDIA_CUDA_Programming_Guide_2.3.pdf.