Parallel Refinement of Slanted 3D Reconstruction using Dense Stereo Induced from Symmetry

Ricardo Ralha[1] · **Gabriel Falcao**[1] · **Joao Amaro**[1] · **Vasco Mota**[2] · **Michel Antunes**[3] · **Joao Barreto**[2] · **Urbano Nunes**[2]

Abstract Traditional dense stereo estimation algorithms measure photo-similarity to calculate the disparity between image pairs. SymStereo is a new framework of matching cost functions that measure symmetry to evaluate the possibility of two pixels being a match. This article proposes a fully functional real-time parallel 3D reconstruction pipeline that uses dense stereo based photo-symmetry. The logN variant of SymStereo achieves superior results for images with slanted surfaces, when compared with other algorithms [1]. This is of particular interest for areas of computer vision such as the processing of datasets for urban scene reconstruction and also for tracking in robotics or intelligent autonomous vehicles. The output results obtained are analyzed by tuning distinct matching cost, aggregation and refinement parameters, targeting the most suitable combinations for slant dominated images. Also, the parallel approach for the aforementioned pipeline consists of a hybrid dual GPU system capable of calculating from 2 up to 132 volumes per second for high- and low-resolution images, respectively.

Keywords Dense stereo estimation · 3D Reconstruction · SymStereo · High resolution images · Parallel Processing · Multiple-GPU processing.

[1] Instituto de Telecomunicações, Department of Electrical and Computer Engineering, University of Coimbra, Portugal (E-mail: {rralha, gff, jamaro}@co.it.pt).

[2] Institute of Systems and Robotics, Department of Electrical and Computer Engineering, University of Coimbra, Portugal (E-mail: {jpbar, urbano}@isr.uc.pt).

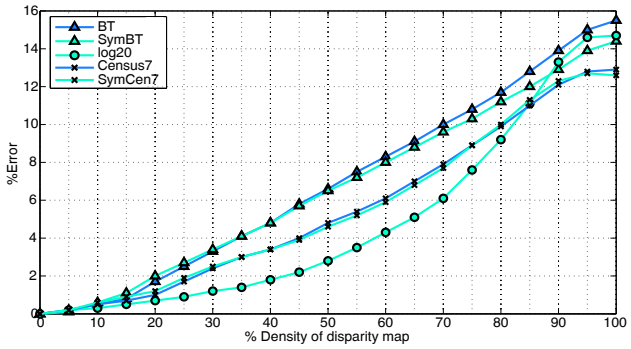
[3] Interdisciplinary Centre for Security, Reliability and Trust (SnT), University of Luxembourg, Luxembourg
(E-mail: michel.antunes@uni.lu).

1 Introduction

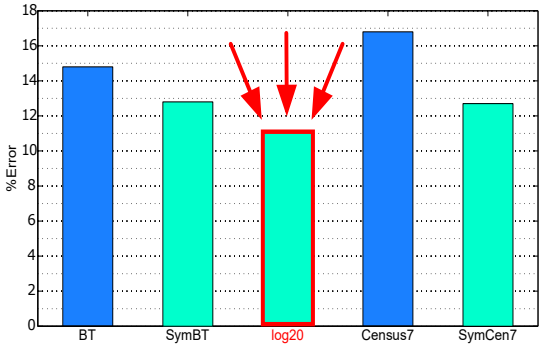
Nowadays, 3D reconstruction of urban scenes for navigation systems has become a recurring topic. The end-application greatly depends on the high quality of the generated map but new challenges arise when reconstructing these types of datasets. Urban scenes are mainly composed of slanted surfaces, which makes them a very important issue to be dealt with. Also, since nowadays we are dealing with high-resolution data, a computationally demanding method can generate consuming execution times.

In this paper, we address the problem of 3D reconstruction of slant dominated urban scenes for building 3D maps in real time. Depending on the methods adopted and the characteristics of the surface being reconstructed, these maps usually need very intensive processing and still have many imperfections. This can lead to reading errors by autonomous systems that can compromise the mission.

The methods used to generate 3D maps using stereo matching involve the calculation of disparity maps. It has recently been proposed the measure of symmetry instead of photo-similarity to compute these maps [1]. This method is named SymStereo and originated a new group of symmetric dense matching cost functions. The SymStereo framework is composed of three functions: *SymBT* (modification of the *BT* metric [2] for measuring symmetry instead of similarity), *SymCen* (non-parametric symmetry metric based on the *Census* [3] transform) and *logN* (uses a bank of log-Gabor wavelets for quantifying symmetry) [1]. By testing the algorithms against each other for different situations, Antunes et al. [1] concluded that *logN* is more suited to deal with slanted surfaces than *SymBT* and *SymCen*. Fig.1, shows that for images composed of low textures and some discontinuities (1a), *logN* over performs *SymBT* and *SymCen* by presenting a smaller percentage of erroneous pixels than its competing symmetry-based algorithms. For heavily slanted



(a) Average percentage of disparity errors for low-textured images.



(b) Percentage of disparity errors in heavily slanted images.

Fig. 1: Comparing different algorithms for generating disparity maps from slanted datasets (results reported in [1]).

images (1b), *logN* performs even better, indicating that this algorithm suits more appropriately the calculation of disparities for slanted images.

Despite the good results, *logN* is a compute-intensive algorithm that takes a lot of time when running on the Central Processing Unit (CPU). Therefore, in order to accelerate the code, this work introduces parallel computing support. Currently, two main frameworks seem to dominate the development of parallel code for processing on Graphics Processing Units (GPUs): the Compute Unified Device Architecture (CUDA) [4] and the Open Computing Language (OpenCL) [5]. CUDA is exclusive to NVIDIA's GPUs while OpenCL is supported by a vast set of devices such as CPUs, GPUs, Digital Signal Processors (DSPs) or Field-programmable gate arrays (FPGAs). Due to portability, OpenCL is an extremely popular framework used in a vast number of studies by the parallel programming community. To illustrate this variety, recent works include [6] and [7], where signal and image processing kernels' performance is evaluated on FPGAs, [8], with OpenCL kernels being used to simulate neural networks on CPUs, GPUs, mobile GPUs and FPGAs, or [9], that explores two H.264/AVC motion compensation kernels on CPU and GPU.

Despite being limited to a number of GPUs, CUDA is highly optimized to NVIDIA's architectures and will perform 10 to 20% better than OpenCL performing the same task [10]. The CUDA framework was used to parallelize the *logN* metric, allowing superior throughput performance when compared to its sequential counterpart.

In fact, CUDA allows to develop a hybrid dual-GPU real-time stereo parallel pipeline for the construction of 3D maps using SymStereo's matching cost function *logN*. In order to improve the quality of reconstructed scenes, local aggregation and visual enhancing post-processing algorithms such as left-right consistency check and occlusion pixel filling were added to obtain depth maps with less imperfections and, consequently, higher-quality 3D maps. The complete pipeline is parallelized and optimized using the CUDA computing language, in order to fully exploit the processing power of two top performer Nvidia GTX Titans.

The main contributions of this paper can be summarized as:

- Full parallelization of the *logN* matching cost, aggregation method, left-right consistency check and 2D to 3D volume mapping, creating a pipeline capable of generating 3D volumes in real-time for high- and low-resolution images;
- Tuning of pipeline parameters, namely the number of scales, shape-factor, scaling step, center frequency of the mother wavelet, aggregation window and left-right consistency check threshold, in order to evaluate the best parameters combinations for slant dominated images and analyze the trade off between visual quality and processing time.

The rest of the paper is organized as follows. Section 2 summarizes related work, section 3 features the *logN* algorithm and section 4 presents the parallelization of the pipeline. Section 5 discusses the experimental results and section 6 closes the paper.

1.1 Notation

The pipeline proposed has six parameters essential for the conducted study. Throughout the paper, they will be referenced as follows: the number of scales of the log-Gabor filters is represented by N, the shape-factor is defined by Ω, the scaling step is depicted by s, the center frequency of the mother wavelet is denoted by ω_0, the aggregation window is represented by A_w and, finally, the left-right consistency check (LRCCheck) threshold is depicted by T.

Regarding the algorithms presented in this paper, A is the $W \times H$ input image spectrum matrix, B represents the $W \times N$ log Gabor coefficients matrix, C is the $W \times H \times N$ filtered input spectrum matrix and D and E are the $W \times H \times N$ left

and right side filtered input spectrum matrices, respectively. These matrices are defined as $float2$ variables (*Complex* in the algorithms) because they contain complex numbers.

DSI is the $W \times H \times drange + 1$ Disparity Space Image matrix, $DispMap$ is the $W \times H$ disparity map matrix, $DispMapL$ and $DispMapR$ are the $W \times H$ disparity map matrices calculated with the left and right images as reference, respectively, $DispMapLRC$ is the $W \times H$ disparity map matrix after the LRCCheck, $DispMapF$ represents the $W \times H$ final disparity map matrix before the 3D reconstruction and, finally, $3DMap$ is the $W \times H \times 3$ 3D map matrix. W and H are the input image width and height, respectively, and *drange* is the number of disparities to be represented in the disparity map.

2 Related work

Depth maps can be calculated with two or more images (multiview stereo) using sparse or dense stereo techniques. Sparse stereo extracts potentially matchable image locations (edges or object discontinuities) and then searches for corresponding locations in the other images [11]. This is useful for matching profile curves, that occur at the boundaries of objects. However, if we have to perform a 3D reconstruction of an entire room or a street, where there are low textured regions, sparse stereo is not the most appropriate solution. In this case, dense stereo is used as it tries to find the corresponding pixel for every pixel in the reference image. A taxonomy of algorithms used for two-image dense stereo can be seen in [12]. We are not going to discuss multiview stereo algorithms as they lie outside the scope of this paper but a taxonomy is also available [13].

To generate depth maps for 3D reconstruction in dense stereo, we can choose between global optimization or local algorithms. Global stereo methods such as belief propagation [14], graph cuts [15] [16] or dynamic programming [17] allow achieving good results but are computationally heavy at a level that can compromise real-time operation. Local methods like squared intensity differences [18] [19] or absolute intensity differences [2] use an aggregation method [20] [21] to compute the correct disparity. Compared to global optimization, local algorithms have slightly worse quality outcomes but are less heavy computationally.

Regarding 3D reconstruction, [22] describes a method to generate a 3D mosaic representation of urban scenes captured by a camera on a mobile platform using a two step procedure that uses a segmentation-based stereo matching algorithm. Despite providing good urban representations, they are not calculated in real-time like the pipeline proposed here and only compute mosaics for a 480x640 image resolution. We calculate 3D maps for higher resolution images. In [23], the authors propose a new 3D reconstruction algorithm that uses a sequential structure-from-motion technique and depth maps calculated by segmentation-based stereo. Similarly, this algorithm does not process data in real-time like our does. 3D reconstructions can also be obtained with the help of Microsoft's Kinect [24]. The collection of depth data is done by its sensors, enabling the reconstruction of an environment. Despite collecting depth data in real-time, the depth map treatment and 3D reconstruction is performed in another interface, which does not perform in real-time. In [25] a rare method for calculating 3D maps using hyperspectral images is presented. After computing various 3D maps, one for each image at different wavelengths, a final 3D model is estimated considering the information of all previously computed models. Performance wise, this is not a real-time method and the maps calculated are for small resolution images. In [26] the authors introduce a sequential 3D reconstruction system that uses dense stereo matching. This pipeline is comprised of two phases, camera parameter estimation and dense stereo correspondence matching. If necessary, more image pairs can be added to the pipeline to achieve better results. As before, this is not a real-time method since it takes several seconds to calculate the final 3D map for an image with only one target. Finally, [27] proposes a system with a sliding camera for 3D reconstruction of indoor and outdoor scenes. The results shown are good but the system isn't currently able to perform real-time processing.

In [28] the authors describe a multi-stage stereo algorithm on a NVIDIA GeForce GTX 580 GPU using the Compute Unified Device Architecture (CUDA) parallel programming model [4], achieving 62 fps for small images. Similarly, [29] presents a real-time local stereo algorithm, attaining 30 fps. [30] presents a real-time stereo method that uses bitwise fast voting and achieves 93 fps on a NVIDIA GeForce 8800 GTX using CUDA. For the same low resolution image, the method proposed in this paper achieves 121 fps. A real-time stereo matching system running on a NVIDIA GeForce GT 540M that achieves 20 fps is presented in [31], while our proposed pipeline achieves 80 fps for the same image resolution. A 3D reconstruction technique for casual scenes with GPU acceleration by a NVIDIA GeForce GT 740 is proposed in [32]. This algorithm achieves a run time of 15 seconds for image resolutions of 600x800, while our proposed method calculates the final disparity map in 0.25 seconds for 375x1242 resolution images. Finally, a 3D environment sensing system for autonomous vehicles is proposed in [33]. Running on a NVIDIA GeForce GTX TITAN, this system achieves 5 fps for 480x640 resolution images with a maximum of 40 disparities. Our pipeline achieves 4 fps for 375x1242 resolution images with 30 disparities.

Developments regarding the partial parallelization of the SymStereo-based pipeline have been made recently. In [34], *logN* was parallelized on a single-GPU machine and [35]

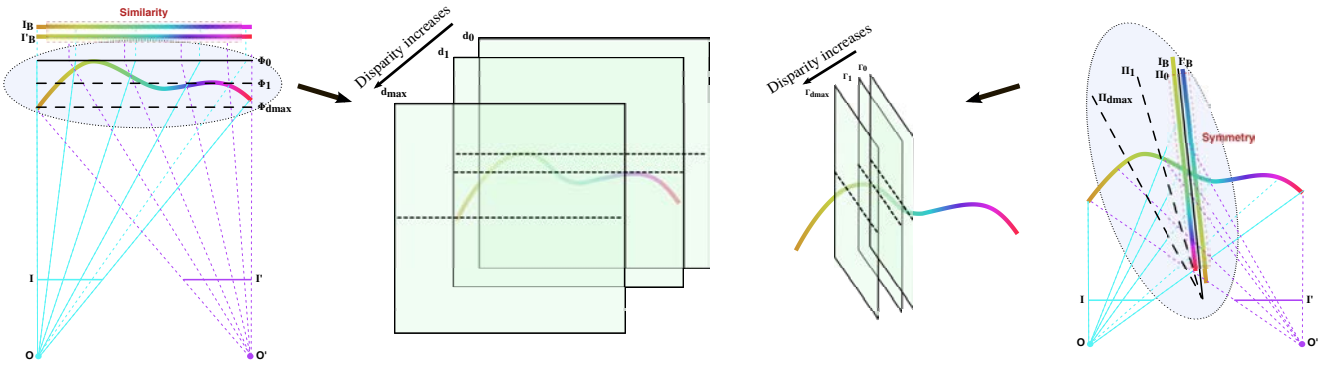


Fig. 2: Differences between Plane sweeping and Symstereo. While images a) and d) represent Plane Sweeping and Symstereo, respectively, b) and c) depict how the DSI is originated with each method.

encompasses the study of the impact of the aggregation stage on the final 3D volumes on a dual-GPU system. In this paper, we create a full 3D pipeline for dual-GPU devices, aiming for real-time execution. Moreover, a complete dataset was captured to test the algorithm with high resolution images and several parameters were tuned with the objective of achieving superior visual results for images with slanted surfaces, without significantly compromising the processing speed of the pipeline.

3 Pipeline for 3D reconstruction

Stereo algorithms generally include the following steps:

1. Matching cost computation;
2. Cost (support) aggregation;
3. Disparity computation;
4. Disparity refinement;
5. Disparity to 3D maps conversion.

In this section, we explain the algorithms used in each phase to calculate the final 3D map.

3.1 Matching cost computation; the *logN* algorithm

First, the matching function receives data from left and right images, which are acquired by a stereo camera. The purpose of the algorithm is to compute the matching costs by verifying the possibility of two pixels, one from each image, corresponding to each other. This evaluation is performed across all possible disparities and pixel locations. By doing this, the Disparity Space Image (DSI) [36] is created. The DSI is a 3D volume that, for each pair pixel-disparity, associates the corresponding matching cost.

New advancements show that by using symmetry rather than photo-similarity to evaluate the likelihood of two pixels being a match, stereo disparity estimation can improve significantly, in particular for slanted images [1]. In Fig.2 are represented the differences between conventional stereo matching by Plane Sweeping [37] and SymStereo. By examining the left side of the image, regarding Plane Sweeping, it is noticeable that each possible disparity d_i is associated with a virtual plane Φ_i, meaning that photo-similarity between I_b and I'_b, that are the results of back-projecting I and I' onto Φ_i, is implicitly measured by the chosen matching cost. By observing the right side of the image, in SymStereo the virtual planes Π_i that pass between the cameras intersect the scene structure. Thus, the back-projection images are reflected with respect to the curve where that intersection occurs, creating a mirroring effect. Also, each plane Π_i corresponds to an oblique plane Γ_i, which means that by choosing the appropriate set of virtual cut planes Π_i, the entire DSI domain can be covered.

As a matching function, the objective of *logN* consists of calculating the matching costs. To do so, it uses a bank of log-Gabor filters for measuring symmetry and antisymmetry energy. The matching cost will be the joint energy calculated by combining the symmetric energy with the antisymmetric equivalent. The N in *logN* stands for the number of wavelet scales that are used to filter the input images.

The matching function can be divided in two phases: the filtering and the computation of the joint energy.

Filtering phase: Consider two images, I and I', and two epipolar lines, one on each image, $I(q_0)$ and $I'(q_0)$, where q_0 is a general pixel location . In this phase, the stereo pair will be filtered with N log-Gabor wavelets. Since these are 1D analytical filters and filtering occurs in the spectral domain, a 1D Fourier transform has to be applied to both images. Being $\mathscr{I} = \mathscr{F}(I)$ the Fourier Transform along the epipolar lines of I and G the matrix of coefficients of the filter, $\mathscr{I}.G_k$ is calculated, for a given wavelet k, where G_k represents a wavelet with the same length as the epipolar line.

In order to return to the time domain, an Inverse Fourier Transform is applied. For a general pixel location q_0 and wavelet k, it can be deduced that

$$s_k(q_0) + \boldsymbol{i} a_k(q_0) = \mathscr{F}^{-1}(\mathscr{I}(q_0).G_k), \quad (1)$$
$$s'_k(q_0) + \boldsymbol{i} a'_k(q_0) = \mathscr{F}^{-1}(\mathscr{I}'_f(q_0).G_k). \quad (2)$$

I'_f is I' flipped horizontally, as it needs to be flipped before the filtering procedure.

Energy Calculation Phase: After filtering the two images, the joint energy needs to be calculated. In order to properly implement this procedure, the right-side signal needs to be shifted by an amount d_i that depends on the virtual cut plane Π_i [1]. The flipped image becomes

$$\widehat{I}(q_0) = I'_f(q_0 - d_i). \quad (3)$$

With this, the symmetry (s^S and a^S) and anti-symmetry (s^A and a^A) coefficients can now be calculated for a generic epipolar line q_0 with

$$s^S_k(q_0) + \boldsymbol{i} a^S_k(q_0) = \big(s_k(q_0) + s'_k(q_0 - d)\big) + \\ + \boldsymbol{i}\big(a_k(q_0) + a'_k(q_0 - d)\big), \quad (4)$$
$$s^A_k(q_0) + \boldsymbol{i} a^A_k(q_0) = \big(s_k(q_0) - s'_k(q_0 - d)\big) + \\ + \boldsymbol{i}\big(a_k(q_0) - a'_k(q_0 - d)\big). \quad (5)$$

With the image being symmetric about the pixel location q_0, the real components s^S and s^A typically take high values and the imaginary components a^S and a^A assume small values [38]. Taking this into account, the symmetry (E^S) and anti-symmetry (E^A) energies can be established for the N wavelet scale responses

$$E^S(q_0) = \frac{\sum_{k=0}^{N-1} |s^S_k(q_0)| - |a^S_k(q_0)|}{\sum_k \sqrt{(s^S_k(q_0))^2 + (a^S_k(q_0))^2}}, \quad (6)$$
$$E^A(q_0) = \frac{\sum_{k=0}^{N-1} |a^A_k(q_0)| - |s^A_k(q_0)|}{\sum_k \sqrt{(s^A_k(q_0))^2 + (a^A_k(q_0))^2}}. \quad (7)$$

The joint energy E comes as a combination of the symmetry and anti-symmetry energies

$$E = E^S.E^A. \quad (8)$$

In Fig.3 it is possible to see the various stages of the *logN* metric.

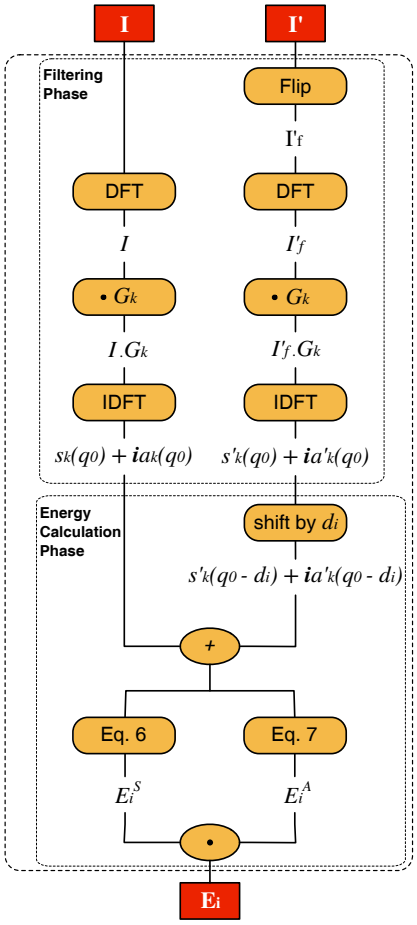


Fig. 3: Depiction of the multiple steps comprising *logN*.

Wavelet scales: To understand the wavelet scales of the log-Gabor filters, we must address the role of each parameter. Fig. 4 illustrates the relation of the parameters with the space-frequency response of the filter.

There are four parameters that define the wavelets: the shape-factor Ω, the center frequency of the mother wavelet ω_0, the scaling step s, and the total number N of wavelets. The shape-factor is related with the bandwidth of the filter and defines a contour in the (ω, σ) domain. The first wavelet scale G_0 is uniquely defined by the center frequency ω_0 and the shape-factor Ω and s sets the distance between center frequencies of successive wavelet scales along the contour. More details on the design of the log-Gabor filters can be found in [39].

3.2 Cost aggregation and disparity computation

There are two types of stereo algorithms: local and global. Local algorithms rely on a window-based approach for aggregation while global algorithms tend to solve a global optimization problem by finding the best disparity that mini-

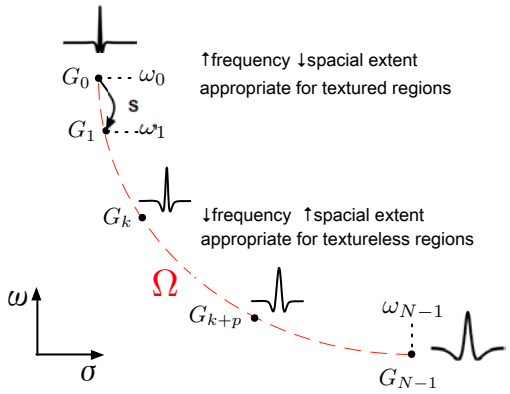


Fig. 4: (Qualitative) space-frequency behaviour of the log-Gabor wavelets G_k . The horizontal axis refers to the spatial support σ of the filter kernel, while the vertical axis concerns the response frequency ω.

mizes a global cost function that is composed by data and smoothness terms. A window-based aggregation step was implemented alongside *logN* to aggregate the determined matching costs, thus making the SymStereo-based pipeline a local method to the calculation of disparity maps.

To estimate the correct disparity for a pixel, the sum of the matching costs is calculated over a square window A_w. This is done for every disparity value. The final disparity chosen will be the disparity associated with the smallest sum of values calculated over the square window.

3.3 Disparity refinement

The disparity refinement stage is used to correct some disparities that were wrongly computed by filling occluded pixels on the depth map. Occluded pixels are only visible in one of the original images. As seen in Fig. 5, this step can be divided in two sub-stages: left-right consistency check and disparity enhancement.

Left-Right Consistency Check: Two disparity maps are necessary for this stage, one computed using the left image as reference and the other with the right image. By subtracting the disparities of corresponding pixels on each depth map, it is verifiable that if the absolute value is superior than a given threshold T, then the pixel is considered occluded.

Disparity enhancement: This stage fills the occluded pixels in the disparity map. The algorithm starts by finding the first occluded pixel, beginning in the top left corner of the disparity map. Then, a 4-way search is performed to find the first non-occluded pixel in each way, left, right, up and down. The disparity selected is the median between the four values that were found. If no value is found in one of the ways, the median is calculated between the other three values and so on. This is done for all occluded pixels in the disparity map.

The pixels that have already been filled are considered in the calculation of the disparities for the next occluded pixels. Therefore, due to these dependencies between pixels, this stage is confined to the CPU.

3.4 From disparity maps to 3D views

To calculate the 3D volume, we map 2D coordinates into 3D:

$$Z = (f \times b)/D; \tag{9}$$
$$X = ((x - c_x) \times Z)/f; \tag{10}$$
$$Y = ((y - c_y) \times Z)/f; \tag{11}$$

where f is the focal length (in pixels), b represents the distance between the two cameras (in meters), c_x and c_y are the coordinates of the principal point (in pixels) and D is the disparity of the pixel.

4 Parallelizing the pipeline

The parallelization of the pipeline aims to compute 3D volumes in real-time. For that purpose, we exploit a hybrid architecture, using a CPU and two Nvidia GTX Titans GPUs. The Compute Unified Device Architecture (CUDA) parallel programming framework [4] is used since it is optimized for extracting superior levels of performance from Nvidia's GPUs. The parallel pipeline is depicted in Fig. 5.

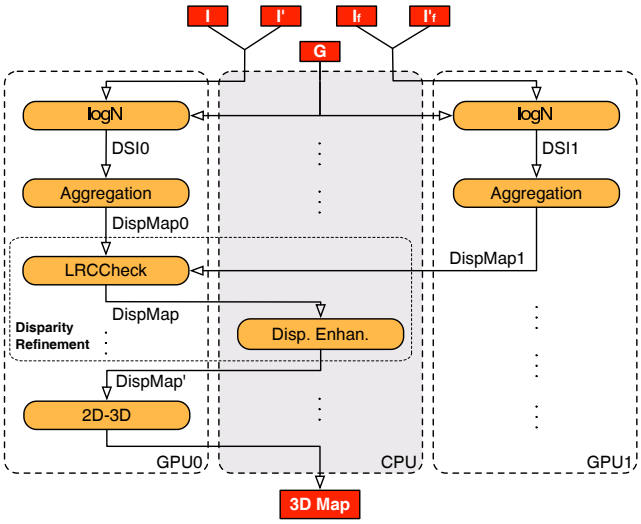


Fig. 5: 3D parallel pipeline representation, where I and I' are the left and right images, I_f and I'_f are the left and right images flipped and G represents the Gabor coefficients.

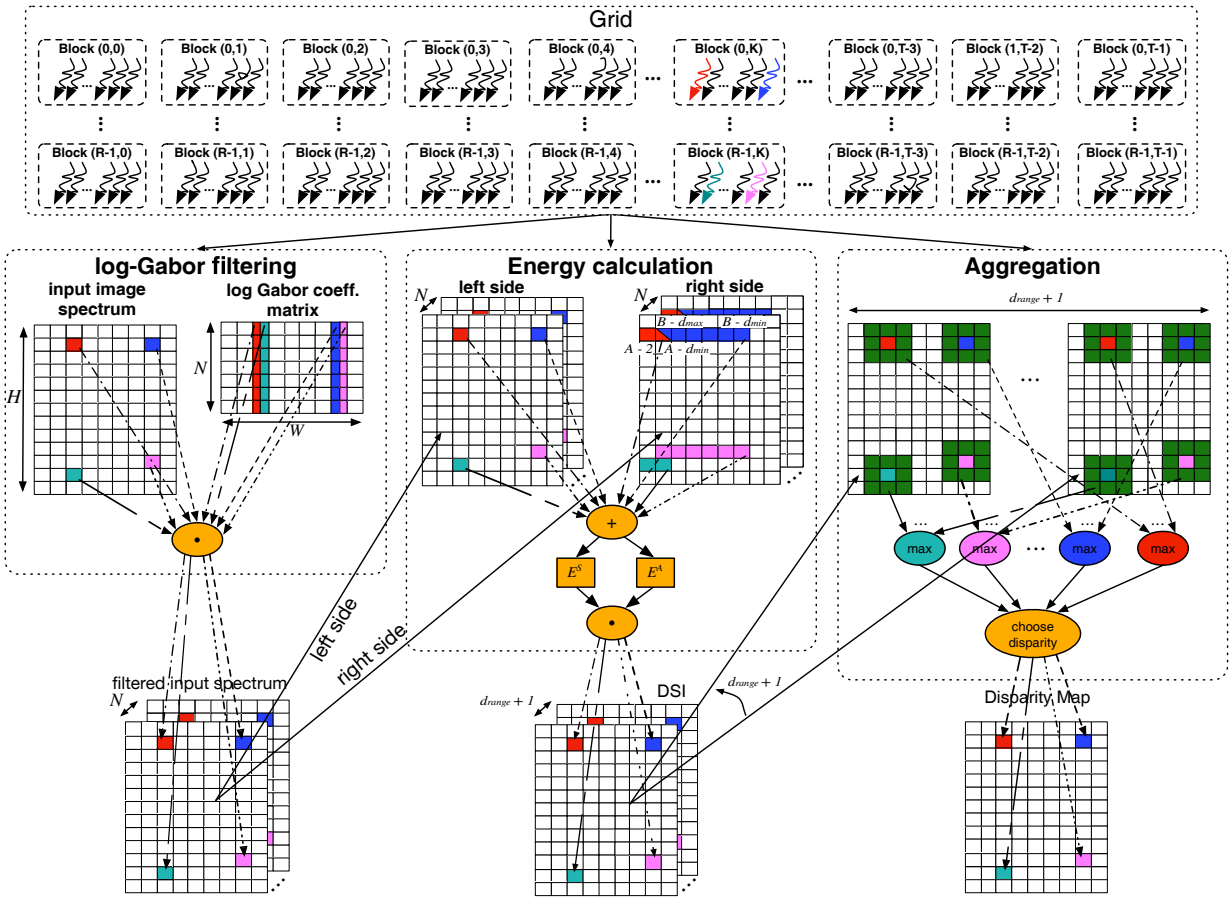


Fig. 6: Representation of the log-Gabor filtering, energy calculation and aggregation kernels. (T,R) are the number of blocks in the (x,y) directions, W and H are the width and height of the input images, respectively, N is the number of wavelets, *dmin* and *dmax* are the minimum and maximum disparities (in this example $dmin = 1$ and $dmax = 6$), respectively, A and B correspond to the red and blue pixels, respectively, and *drange* is the difference between *dmax* and *dmin*.

4.1 Memory performance analysis

The Nvidia GeForce TITAN GPU has a complex memory hierarchy [40] that can be exploited in order to take full advantage of the GPU's capabilities. Besides the GPU main memory, the proposed implementation uses the L1 and L2 level cache of the device to increase memory throughput and enhance the pipeline's performance. Regarding shared memory, since the data block size is higher than the available shared memory, which at the finest level of thread-granularity forces several memory swaps to occur, the use of this technique decreases efficiency due to a high number of transaction overheads accessing the global memory. Therefore, this option was discarded from the optimized solution.

Data transfers between CPU and GPU can have a significant impact on the final processing time. In order to minimize this effect and achieve higher throughput performance, we changed the way data was allocated in the CPU's main memory. By default, when allocating data in the host memory, the system performs a pageable allocation. However, the device is not able to transfer data directly from pageable memory. To perform this transfer, a temporary pinned array is created on the host memory so that data can be transferred from the pageable section to the array and only then transferred to the device memory [4]. To avoid this intermediate step, we perform pinned allocations in the host, minimizing data transfer latency.

4.2 LogN parallelization

Fig. 3 shows the LogN algorithm. To parallelize it, three kernels were created: the flip, filtering and energy calculation kernels. To perform the discrete-time Fourier transform (DTFT) and inverse discrete-time Fourier transform (IDTFT) on the GPU, the optimized CUFFT API [41] is used. Single-precision floating-point variables are employed

to store the FFT values since double-precision variables have no visible impact on the final results and have the drawback of allocating more GPU memory than single-precision ones.

Filtering phase on the GPU: In this phase, two of the three kernels mentioned previously are used, where each thread processes one image pixel. The flip kernel has half the threads of the remaining kernels. Since the objective is to flip the input matrix horizontally, only half of the matrix width needs to be swept. Regarding the filtering kernel, each line of the input spectrum has to be multiplied element-wise by every line of the log-Gabor coefficients matrix. This way, the output of the kernel consists of N matrices used in the posterior energy calculation phase. This is illustrated in the left side of Fig. 6 and the kernel is shown in Algorithm 1. The computational workload of this kernel depends on the number of wavelets chosen for the filter. The larger the number of wavelets, the larger the filter becomes.

Algorithm 1 Log-Gabor filtering GPU parallel kernel

```
// j and i are the thread values in x and y,
 ↪  respectively
if( j < W && i < H )
{
   for( int z = 0; z < N; z++ )
   {
      C[j][i][z].x = A[j][i].x * B[j][z].x;
      C[j][i][z].y = A[j][i].y * B[j][z].y;
   }
}
```

Energy calculation on the GPU: This operation represents a time consuming phase that depends on the image dimensions, the disparity range and the number of scales chosen. The inputs are the images after being filtered by the log-Gabor wavelets and the output is the DSI. Each thread processes the joint energy (8) corresponding to every disparity for one pixel. For the left side pixel, the real (s_k) and imaginary (a_k) components are always the same, for any disparity. For the right side pixel, these components (s'_k and a'_k) depend on the disparity value ((4) and (5)) since the disparity value defines the pixel of the right side to be evaluated. For example, the middle of Fig. 6 depicts a thread calculating every joint energy for pixel B of the left side associated with every pixel from $B-dmin$ to $B-dmax$ of the right side. For pixel A, since $drange+1$ exceeds the number of pixels left for the edge of the matrix, only the joint energies associated with the two pixels on the edge are calculated. The DSI has $drange+1$ matrices containing the costs of each pixel related to each disparity. Algorithm 2 illustrates our application's kernel.

Algorithm 2 Energy calculation GPU parallel kernel

```
// dmin must be superior to 0
// j and i are the thread values in x and y,
 ↪  respectively
if(j < W && i < H)
{
   Complex C1;
   Complex C2;

   for(int d = 0; d < d_range ; d++)
   {
      float sum_num1 = 0.0f;
      float sum_den1 = 0.0f;
      float sum_num2 = 0.0f;
      float sum_den2 = 0.0f;

      if(j >= d + dmin)
      {
         for(int z = 0; z < N ; z++)
         {
// C1 and C2 represent the symmetry and anti-symmetry
 ↪  coefficients
            C1.x = D[j][i][z].x + E[j-(d+dmin)][i][z].x;
            C1.y = D[j][i][z].y + E[j-(d+dmin)][i][z].y;

            C2.x = D[j][i][z].x - E[j-(d+dmin)][i][z].x;
            C2.y = D[j][i][z].y - E[j-(d+dmin)][i][z].y;

// Factors associated to the symmetry energy
            sum_num1 += fabsf(C1.x) - fabsf(C1.y);
            sum_den1 += hypotf(C1.x , C1.y);

// Factors associated to the anti-symmetry energy
            sum_num2 += fabsf (C2.y) - fabsf(C2.x);
            sum_den2 += hypotf(C2.x , C2.y);
         }
         DSI[j][i][d] = ((sum_num1 + sum_den1) *
 ↪  (sum_num2 + sum_den2)) / (sum_den2 * sum_den1);
      }
   }
}
```

4.3 Aggregation and refinement parallelization

This section encompasses four algorithms. The disparity enhancement algorithm is not parallelizable, which means that it runs on the host side. The remaining three kernels, namely aggregation, consistency check and 3D conversion run on the device, which implies that data needs to be transferred from the GPU to the CPU and then back to the GPU. This is depicted in Fig. 5.

Aggregation on the GPU: Here, each thread calculates the sum of the matching costs over the defined square window, for each disparity, and chooses the disparity with the lowest sum of costs (right side of Fig. 6). Like in previous kernels, each thread is associated to a single pixel. This kernel can be computationally costly, depending on the window size and disparity range chosen. With a large window size, the pro-

cessing time of the pipeline can soar, damaging processing time performance. The kernel can be seen in Algorithm 3.

Algorithm 3 Aggregation GPU parallel kernel

```
// j and i are the thread values in x and y,
 ↪  respectively
int Win = floor(0.5 * Aw);
if (j >= Win && j < W - Win && i >= Win && i < H - Win)
{
   float temp = 0.0f;
   float dsi_sum;
   float dsi_disp;
   int count = 0;
// The DSI sum for the minimum disparity must be
 ↪  calculated separately. The DSI values within the
 ↪  aggregation window are added
   for(int z = i - Win; z <= i + Win; z++)
      for(int p = j - Win; p <= j + Win; p++)
         temp += DSI[p][z][count];
// The values associated with the minimum disparity are
 ↪  stored
   dsi_sum = temp;
   dsi_disp = dmin;
   count = 1;
// The other DSI sums area calculated for each disparity
   for (int d = dmin + 1; d <= dmin + drange; d++)
   {
      temp = 0;
      for(z = i - Win; z <= i + Win; z++)
         for(p = j  - Win; p<= j + Win; p++)
            temp += DSI[p][z][count];
// If the new sum is smaller than the old one, a new
 ↪  disparity is saved
      if (dsi_sum > temp)
      {
         dsi_sum = temp;
         dsi_disp = d;
      }
      count++;
   }
// The disparity associated with the minimum DSI sum is
 ↪  saved
   DispMap[j][i] = dsi_disp;
}
```

Refinement on the GPU: The consistency check phase involves two GPUs working in parallel, as we need two disparity maps to be generated and compared. With two GPUs, the maps are calculated concurrently, saving considerable processing time. Again, each thread verifies the consistency of the disparity associated to a single pixel. Algorithm 4 highlights this kernel.

As mentioned before, the algorithm used to fill occluded pixels should not be performed on the GPU, hence data must be transferred to the host's memory in order to be processed by the CPU. When all calculations are over, the data can be transferred back to the device's memory.

Algorithm 4 LRCCheck GPU parallel kernel

```
// j and i are the thread values in x and y,
 ↪  respectively
if(j < W && i < H)
{
   int xr, xlr;
   xr = j - DispMapL[j][i];
// If the displacement leads outside the matrix
↪  boundaries, the pixel is considered occluded
   if(xr < 1)
      DispMapLRC[j][i] = 0;
   else
   {
      xlr = xr + DispMapR[xr][i] ;
// If the absolute difference is superior to the
↪  threshold, the pixel is considered occluded
      if (abs( j - xlr ) < T)
         DispMapLRC[j][i] = DispMapL[j][i];
      else
         DispMapLRC[j][i] = 0;
   }
}
```

3D Reconstruction on the GPU: At the end of the pipeline, the final disparity map is used to calculate the 3D coordinates for the 3D scene reconstruction. Each thread, corresponding to one pixel, is responsible for calculating the three coordinates necessary to generate the 3D map, according to (9), (10) and (11). Algorithm 5 depicts this kernel. With all pixels processed, data can be transferred from the device to the host.

Algorithm 5 3D reconstruction GPU parallel kernel

```
// j and i are the thread values in x and y,
 ↪  respectively
if(j < W && i < H)
{
   float Z = (f * b) / DispMapF[j][i];
   3DMap[j][i][0] = ((j - cx) * Z) / f;
   3DMap[j][i][1] = ((i - cy) * Z) / f;
   3DMap[j][i][2] = Z;
}
```

5 Experimental Results

To present the results, this section has been split into three parts: the first one focusses on the tests performed to tune the pipeline parameters; the second one discusses the visual results of the 3D volumes generated; and the third one emphasizes the processing times of the method.

The pipeline presented here was developed using CUDA 6.5 and uses a GeForce GTX Titan dual-GPU workstation with an i7 4790k @ 4 GHz, 32 GB of RAM running Ubuntu 14.04.1 and GCC 4.8.2.

To calculate the percentage of error in non-occluded areas, the absolute difference between the calculated disparity map and the matching ground truth image is evaluated. For corresponding pixels in each map, if the difference is greater than one, then the disparity of the processed map's pixel is wrong.

The only parameter that maintains the same value throughout the tests is the consistency check threshold T since its variation has a small impact on the final results. It was manually set to 3.

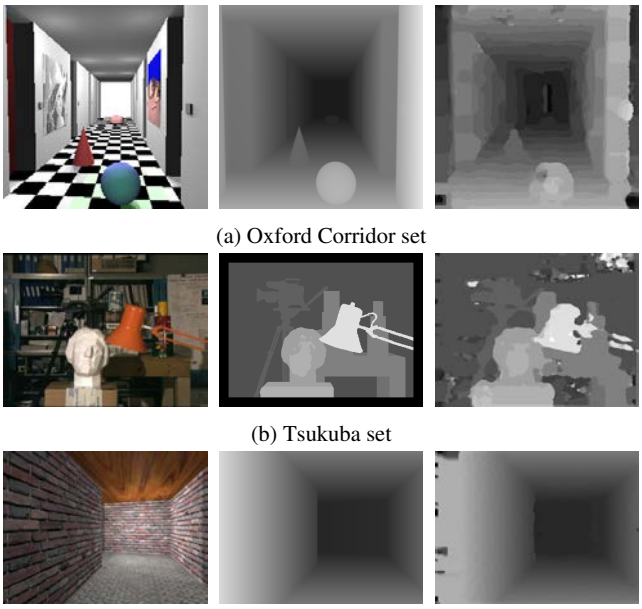
To test the pipeline presented in this paper, the KITTI stereo dataset [42] was chosen since it is mostly composed of urban scenes with slanted surfaces. The KITTI stereo benchmarking [43] was utilized in order to evaluate the pipeline relatively to other modern methods capable of computing disparity maps of urban datasets. For the benchmarking a disparity range of 0 to 109 was used and the error calculation has a 3 pixel margin. Also, all tested images are rectified. This means that a geometric transformation is performed to each image pair in order for them to be horizontally aligned. This is extremely important since *logN* only searches for corresponding pixels on matching horizontal lines.

This benchmark compares CPU and GPU implementations. It is important to note that different stereo vision algorithms require distinct parallelization efforts. Usually, local stereo or semi-global matching algorithms suit better parallelization while global matching algorithms are less likely to be implemented on GPUs. This benchmark has a large variety of algorithms and while some can be more efficiently implemented on GPUs, the final map results are addressed and compared in Table 2.

5.1 Parameter tuning and benchmarking

The objective is to create fast and high quality 3D reconstructed images with slanted surfaces. With that goal in mind, various combinations of the already mentioned pipeline parameters were applied to different sets of images; one image from a dataset we created (820x1142 pixels); six images from the KITTI dataset [42] (375x1242 pixels); one image from the synthetic Tunnel dataset [44] (300x400 pixels); another image from the Tsukuba set [12] (288x324 pixels); and, finally, an image from the Oxford Corridor set (256x256 pixels). The Tsukuba image does not contain slant but is used for reference in this paper. The images of the KITTI and our dataset were given letters from 'A' to 'E' and are represented in Fig. 8 and Fig. 11 through 14.

The original tests performed in [1] were carried out using specific values for each parameter. That combination is used in this paper and it is called reference combination, where N=20, Ω=0.55, s=1.05, W_0=0.25, A_w=9 and, additionally, T=3. With these values, an equilibrium is reached by not compromising either textured or textureless regions. In fact, this study has been made before [1] and lies outside the scope of this paper. In Fig. 7 the Oxford Corridor, Tsukuba and Tunnel disparity maps calculated with the reference combination are shown, using the proposed parallel processing pipeline.



(a) Oxford Corridor set

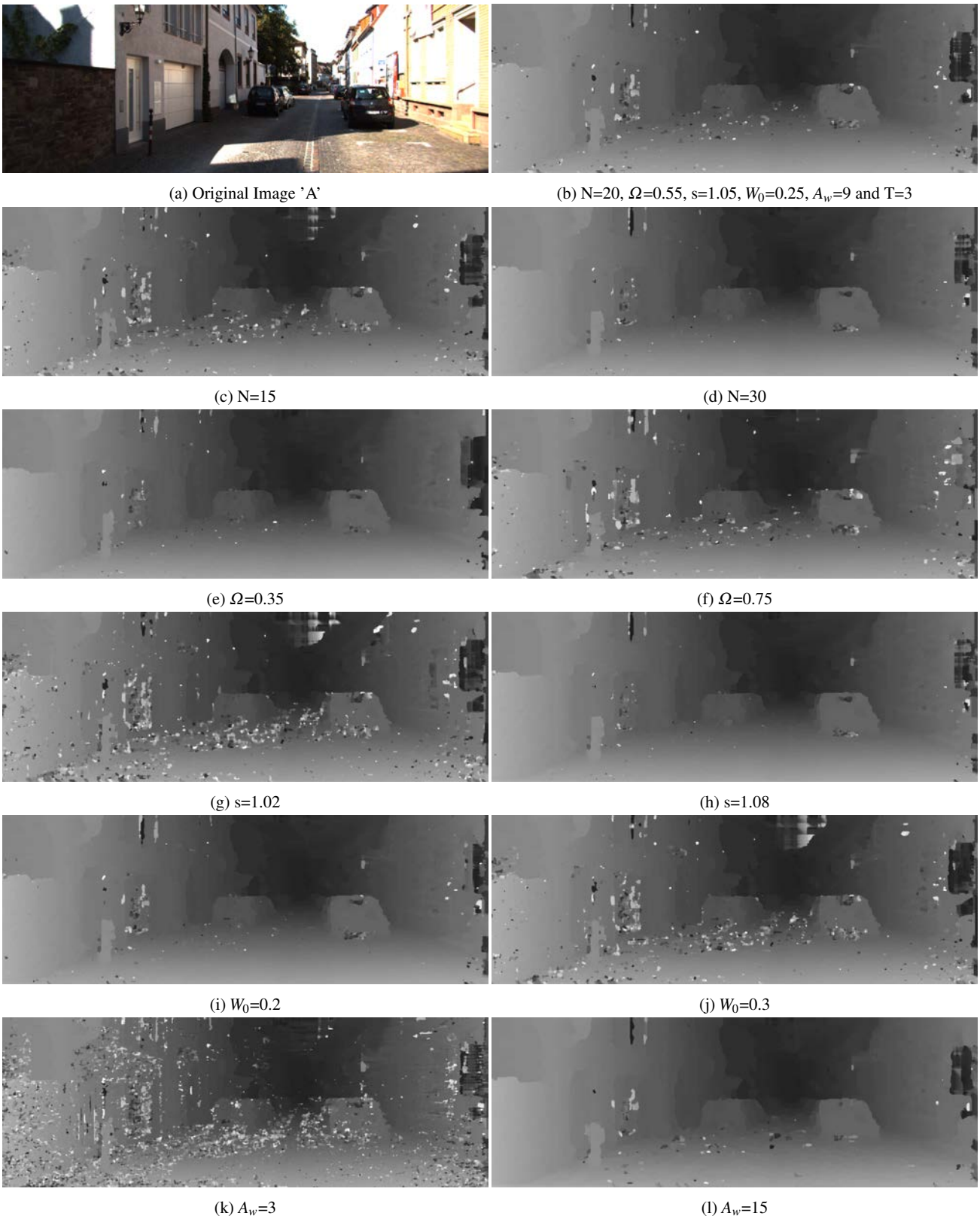
(b) Tsukuba set

(c) Tunnel set

Fig. 7: a) Oxford Corridor; b) Tsukuba; c) Tunnel image; original image, ground truth and disparity map generated using the implemented pipeline.

The tests performed aim to show the impact of the various parameters on the disparity maps and to find the best combination for images with slanted surfaces. Increasing the number of wavelets will let the filter have a larger range, reacting better to images with textureless regions. The shape factor changes the value of the center frequency of each wavelet, giving us different results. The higher the scaling step, the bigger the difference between the center frequencies of each wavelet. Choosing the right center frequency for the first wavelet is crucial, as it can change the results drastically. With a high first frequency, the disparity map yields better results for textured regions (Fig. 4). Finally, a large aggregation window corrects some wrong disparities but affects the definition on the discontinuities. The changes that occur in image 'A' by changing the parameters can be observed in Fig. 8. The alteration of these parameters has a large impact on the final disparity maps. This is why it is crucial to reach an optimal combination for our group of images.

By processing disparity maps (trying different combinations of parameters that can be found in Table 1) for



(a) Original Image 'A'

(b) N=20, Ω=0.55, s=1.05, W_0=0.25, A_w=9 and T=3

(c) N=15

(d) N=30

(e) Ω=0.35

(f) Ω=0.75

(g) s=1.02

(h) s=1.08

(i) W_0=0.2

(j) W_0=0.3

(k) A_w=3

(l) A_w=15

Fig. 8: Example of parameter variation effects for the KITTI dataset image 'A'. The aim is to show the effects of changing the value of different parameters and the elevated degree of freedom disposable for parameter variation. Only one parameter is changed in each row. The others maintain its reference values, shown in b).

Table 1: Combinations used in the tests with ground truth verification for the Oxford Corridor, Tsukuba, Tunnel (1, 2, 3, 4) and KITTI (5, 6, ..., 15) datasets.

	Parameters					
Combination	N	Ω	s	W_o	A_w	T
1	20	0.55	1.05	0.25	15	3
2	30	0.35	1.08	0.20	15	3
3	15	0.55	1.05	0.20	15	3
4	15	0.35	1.02	0.30	15	3
5	30	0.55	1.08	0.25	15	3
6	30	0.55	1.08	0.20	15	3
7	30	0.55	1.08	0.30	15	3
8	20	0.55	1.08	0.20	15	3
9	20	0.35	1.08	0.20	15	3
10	30	0.35	1.05	0.20	15	3
11	30	0.35	1.08	0.30	15	3
12	30	0.55	1.05	0.02	15	3
13	30	0.35	1.08	0.25	15	3
14	20	0.35	1.08	0.25	15	3
15	30	0.35	1.08	0.20	15	3
16	30	0.35	1.05	0.25	15	3
17	15	0.35	1.08	0.25	15	3

three images and comparing them with the ground truth provided, the accuracy of the pipeline was tested. The results are shown in Fig. 9, with the Tunnel image being the most accurate. Combinations '2', '3' and '4' provide the best results for the Oxford Corridor, Tsukuba and Tunnel images, respectively.

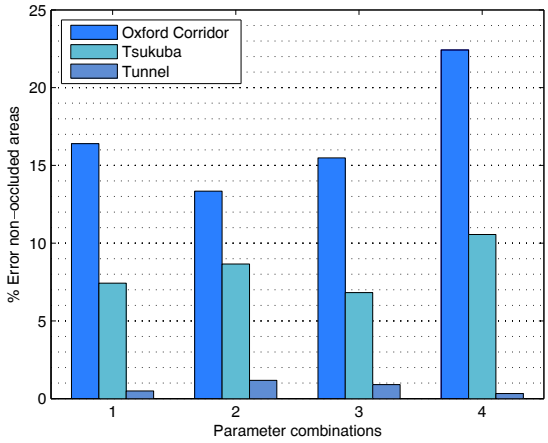


Fig. 9: Test results for the Oxford Corridor, Tsukuba and Tunnel images with ground truth verification, using parameter combination from Table 1.

In order to find the combination that would produce the best 3D map, the tests taken encompassed the creation of several disparity maps of twenty training images from the KITTI dataset by combining all parameters. Fig. 10 shows the pipeline accuracy for images 'B' and 'C' (the only ones presented in this paper with ground truth, images 'A', 'D' and 'E' don't have a ground truth image) of the KITTI dataset. The pipeline produces results with a low error percentage. The combinations used were the best performing combinations of the tests taken with the KITTI training dataset (where the error variation from the best combination '5' to combination '17' is less than 1%) and can be found in Table 1. The pipeline parameters affect visual results as well as processing times. Therefore, instead of choosing combination '5' to compute the disparity maps, we opted for combination '8' because it significantly diminishes the final pipeline times (time analysis in section 5.3) and the error percentage difference is only of 0.32%.

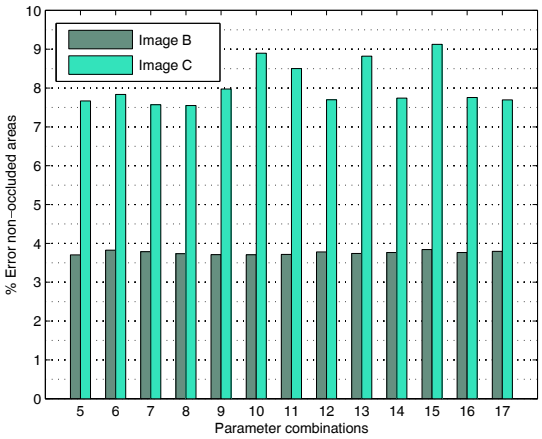


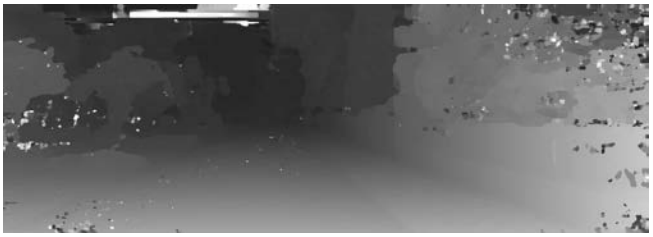
Fig. 10: Test results for images 'B' and 'C' (Figures 11 and 12) of the KITTI dataset with ground truth verification, using parameter combination from Table 1.

Disparity maps of tested images 'B' to 'E' can be seen in Fig. 11 through 14. The maps computed with combination '5' show a significant improvement when compared with the ones calculated with the reference combination.

With the chosen parameter combination, our algorithm was benchmarked [43] on the 29th October 2015, achieving positive results, with an error of less than 10% for non-occluded areas, less than 12% for occluded areas and the 8th best processing time (considering only better performing methods), which is an important highlight for a computationally heavy dense stereo algorithm when compared to sparse stereo algorithms. The results are presented in Table 2.

The ELAS algorithm is a sparse stereo method since it resorts to support points (image points that due to their uniqueness can be robustly matched) for stereo matching. This makes the algorithm capable of handling object discontinuities much better than the SymStereo-based pipeline.

(a) Original Image 'B'



(b) N=20, Ω=0.55, s=1.05, W_0=0.25, A_w=9 and T=3

(c) N=20, Ω=0.55, s=1.08, W_0=0.2, A_w=15 and T=3

Fig. 11: Test results for the KITTI dataset image 'B'.

(a) Original Image 'C'

(b) N=20, Ω=0.55, s=1.05, W_0=0.25, A_w=9 and T=3

(c) N=20, Ω=0.55, s=1.08, W_0=0.2, A_w=15 and T=3

Fig. 12: Test results for the KITTI dataset image 'C'.

(a) Original Image 'D'

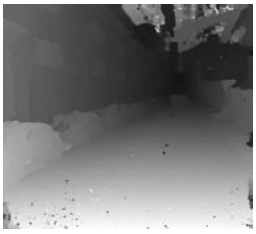
(b) N=20, Ω=0.55, s=1.05, W_0=0.25, A_w=9 and T=3

(c) N=20, Ω=0.55, s=1.08, W_0=0.2, A_w=15 and T=3

Fig. 13: Test results for the KITTI dataset image 'D'.

(a) Original Image 'E'

(b) N=20, Ω=0.55, s=1.05, W_0=0.25, A_w=9 and T=3



(c) N=20, Ω=0.55, s=1.08, W_0=0.2, A_w=15 and T=3

Fig. 14: Test results for the created dataset image 'E'.

Table 2: KITTI dataset benchmarking [43] (`http://www.cvlibs.net/datasets/kitti/eval_stereo_flow.php?benchmark=stereo`), as of 29th October 2015, with pipeline results marked. **Out-Noc** and **Out-All** are the percentage of erroneous pixels for non-occludded zones and in all, respectively, **Avg-Noc** and **Avg-All** are the average pixel errors for non-occludded zones and in all, respectively, and **Density** is the percentage of pixels provided for evaluation by the method.

Rank	Method	Out-Noc	Out-All	Avg-Noc	Avg-All	Density	Runtime	Environment
1	MC-CNN-acrt	2.43 %	3.63 %	0.7 px	0.9 px	100.00 %	67 s	Nvidia GTX Titan X (CUDA, Lua/Torch7)
2	Displets	2.47 %	3.27 %	0.7 px	0.9 px	100.00 %	265 s	> 8 cores @ 3.0 Ghz (Matlab + C/C++)
3	MC-CNN	2.61 %	3.84 %	0.8 px	1.0 px	100.00 %	100 s	Nvidia GTX Titan (CUDA, Lua/Torch7)
						...		
20	MBM	4.35 %	5.43 %	1.0 px	1.1 px	100.00 %	0.20 s	1 core @ 3.0 Ghz (C/C++)
24	AARBM	4.86 %	5.94 %	1.0 px	1.2 px	100.00 %	0.25 s	1 core @ 3.0 Ghz (C/C++)
27	AABM	4.97 %	6.04 %	1.0 px	1.2 px	100.00 %	0.12 s	1 core @ 3.1 Ghz (C/C++)
29	rSGM	5.03 %	6.60 %	1.1 px	1.5 px	97.22 %	0.20 s	4 cores @ 2.6 Ghz (C/C++)
31	RBM	5.18 %	6.21 %	1.1 px	1.3 px	100.00 %	0.20 s	1 core @ 3.0 Ghz (C/C++)
38	SNCC	5.40 %	6.44 %	1.2 px	1.3 px	100.00 %	0.11 s	1 core @ 3.1 Ghz (C/C++)
46	Toast2	6.16 %	7.42 %	1.2 px	1.4 px	95.39 %	0.03 s	4 cores @ 3.5 Ghz (C/C++)
						...		
60	ELAS	8.24 %	9.96 %	1.4 px	1.6 px	94.55 %	0.30 s	1 core @ 2.5 Ghz (C/C++)
						...		
67	SymST-GP	9.79 %	11.66 %	2.5 px	3.3 px	100.00 %	0.254 s	Dual - Nvidia GTX Titan (CUDA)
						...		
86	ALE-Stereo	50.48 %	51.19 %	13.0 px	13.5 px	100.00 %	50 m	1 core @ 3.0 Ghz (C/C++)
87	MEDIAN	52.61 %	53.67 %	7.7 px	8.2 px	99.95 %	0.01 s	1 core @ 2.5 Ghz (C/C++)
88	AVERAGE	61.62 %	62.49 %	8.0 px	8.6 px	99.95 %	0.01 s	1 core @ 2.5 Ghz (C/C++)

Additionally, it also uses plane fitting, that limits the search range for corresponding pixels, saving processing time, and has left-right consistency check and post-processing phases to correct pixel disparities [45].

Unlike ELAS, the pipeline presented in this paper is a straightforward dense stereo implementation. This means that it does not have any techniques, like plane fitting, to save processing time and speed pixel matching. Despite not excelling in object discontinuities, the algorithm behaves extremely well for low-textured regions and it has some similarities with ELAS like the consistency check and post-processing stages.

5.2 3D reconstruction of urban scenes with slant

The obtained 3D reconstructions can be observed in different views (frontal and lateral) in Fig. 15. The close analysis of the reconstructions shows that around discontinuities there are some wrongly calculated pixels. For example, in Fig. 15(h), the trunk of the tree surrounded by the wall of one of the houses in the background is an error, since the trunk and the wall were calculated as being in the same disparity zone.

When evaluating reconstructions, an important feature is the sky. It is one of the main noise sources existent in most tested images and, since it belongs to the infinite plane, a correct disparity estimation is impossible. Sky perturbation is visible in Fig. 15(j).

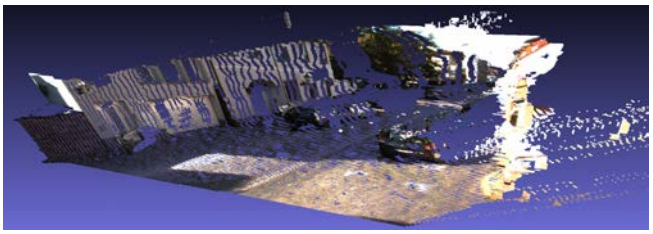
The proposed pipeline is able to robustly handle slanted surfaces. This is observable in every side view of the aforementioned scenes. By using the calculated parameter configuration, the algorithm was optimized to this kind of scenarios. This is where we can see the strength of the *logN* matching cost and the versatility that it provides for different situations.

More examples and videos of the generated 3D reconstructions are available at `http://montecristo.co.it.pt/3DReconstructions/`.

5.3 Numerical results

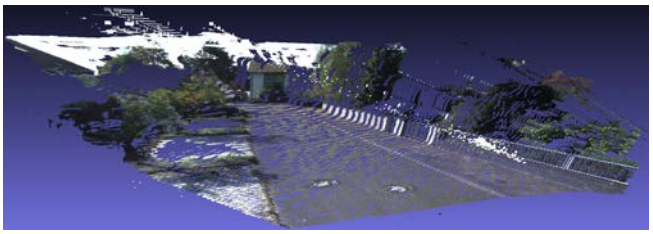
Results seem to indicate that the pipeline parameters are free to change without consequence regarding computational effort. Unfortunately, this is not true as some parameters have a high impact on the processing time of the pipeline. Of the six tuned parameters, N and A_w have visual and processing impact while Ω, s, W_0 and T only have visual impact. Another parameter has a high impact, especially in processing time: the disparity range. Each one of the three parameters has a different impact in processing speed. By increasing the value of N, a larger log-Gabor coefficients matrix is created. This influences the transfer times of the matrix coefficients to the GPU, the log-Gabor filtering and the energy processing kernels that run on the GPU. A_w only influences the aggregation stage but the disparity range impacts not only this stage as the posterior energy calculation phase.

(a) Front 3D Reconstruction



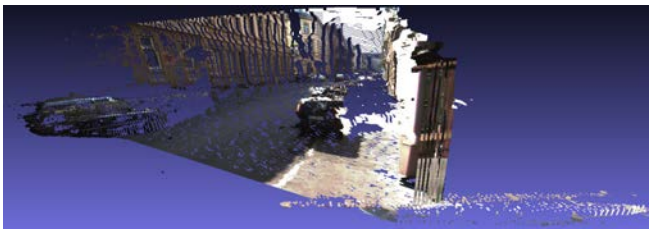
(b) Side 3D Reconstruction

(c) Front 3D Reconstruction



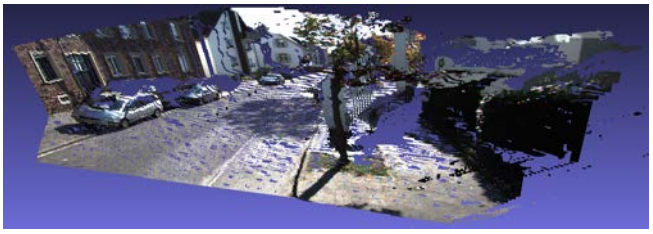
(d) Side 3D Reconstruction

(e) Front 3D Reconstruction



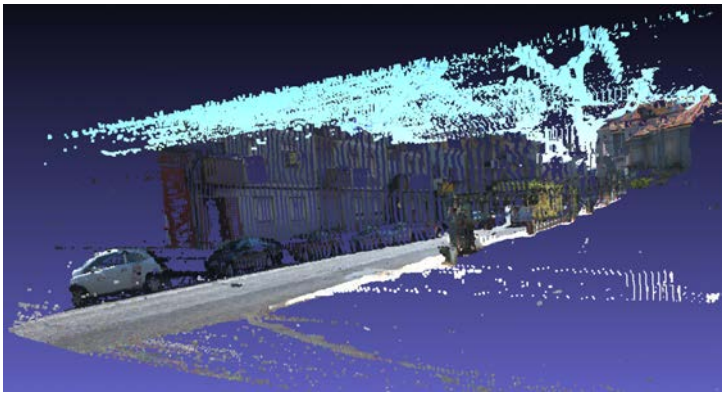
(f) Side 3D Reconstruction

(g) Front 3D Reconstruction



(h) Side 3D Reconstruction

(i) Front 3D reconstruction



(j) Side 3D reconstruction

Fig. 15: KITTI dataset images 3D reconstruction: 'A', 'B', 'C', 'D', 'E' w/ N=20, Ω=0.55, s=1.08, W_0=0.2, A_w=15 and T=3.

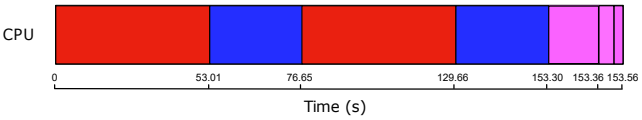
Table 3: Volumes per second and pipeline times (ms) with the variation of the number of scales, the disparity range and the aggregation window for the five resolutions of the tested images. The values highlighted are the processing times with the reference combination and with the combination for the best visual results obtained. The corresponding volumes per second are represented in rows "Volumes p/sec" 1 and "Volumes p/sec" 2, respectively.

		Image Resolution														
		256x256			**288x384**			**300x400**			**375x1242**			**820x1142**		
Aw	**N / Range**	d=12	d=22	d=32	d=6	d=16	d=26	d=45	d=65	d=85	d=70	d=110	d=150	d=155	d=185	d=215
3	15	3.67	4.82	6.02	4.28	5.68	7.40	11.21	14.51	17.82	69.10	95.11	119.52	249.28	283.98	329.95
	20	4.20	5.60	7.20	4.85	6.64	8.82	13.78	18.10	22.35	85.55	118.57	150.64	310.45	355.93	397.66
	30	5.21	7.33	9.66	6.05	8.81	12.14	20.13	26.68	33.17	124.31	175.31	220.10	452.10	525.12	590.81
9	15	4.55	6.44	8.23	5.04	7.22	9.87	17.01	**22.98**	28.50	97.96	139.80	181.49	382.45	415.25	482.29
	20	4.98	**7.56**	9.48	5.78	**8.28**	11.35	19.69	**26.59**	33.04	113.92	**163.03**	209.83	431.61	**500.21**	565.89
	30	6.11	9.04	11.99	6.86	10.68	14.82	26.11	35.10	43.37	153.88	218.03	279.48	575.29	668.68	755.79
15	15	6.46	9.82	13.21	6.53	**10.72**	15.34	28.68	39.4	50.05	162.36	234.27	301.25	617.92	719.40	832.10
	20	7.11	10.75	14.51	7.19	11.70	16.93	31.35	42.99	54.63	176.84	**254.00**	335.76	680.37	**794.88**	909.39
	30	8.07	**12.44**	17.24	8.36	14.14	20.27	37.77	51.72	64.74	214.28	313.61	399.12	822.86	972.74	1099.37
Volumes p/sec 1			132.3			120.8			37.6			6.1			2	
Volumes p/sec 2			80.4			93.3			43.5			4			1.25	

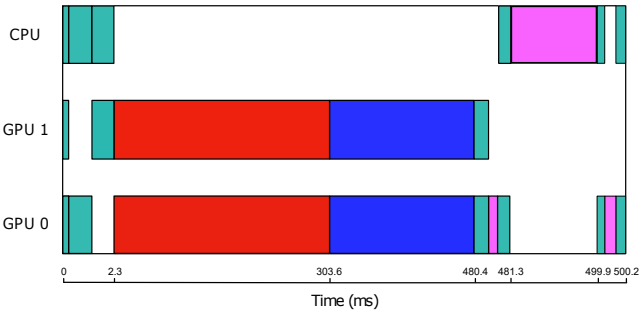
In Table 3 the processing times depending on A_w, N and the disparity range are shown, for five image resolutions. It is noticeable that A_w greatly increases the final processing times, especially in larger images, and N is the parameter that least influences the throughput performance. The disparity range is chosen depending on the different depths of various objects belonging to the scene. For every image, a range of disparities that fits the represented scene is needed. Therefore, it is very important to reach a compromise between the parameters, as their variation has a high impact on the number of maps generated per second.

In the tests performed, disparity ranges of 22, 16, 65, 110 and 185 for the Oxford Corridor, Tsukuba, Tunnel, KITTI and our dataset images, respectively, were used. The analysis of Table 3 concludes that the pipeline is able to process, for each image, 132.3, 120.8, 37.6, 6.1 and 2 volumes per second, respectively, with the reference combination. For the best quality results in each image, the effects of changing the parameters are obvious. The rates of generated volumes diminished (excluding the Tunnel image) being 80.4, 93.3, 43.5, 4 and 1.25 for each of the aforementioned image resolutions.

Fig. 16 compares the CPU implementation with the GPU pipeline for an 820x1142 image resolution, with the reference combination. As predicted, the *logN* and aggregation stages are the most time consuming phases of both implementations but there are differences. In the CPU implementation, the SymStereo and aggregation phases must be run twice sequentially to calculate the two disparity maps for the LRCCheck phase, while in the GPU implementation, each device calculates one disparity map concurrently, diminishing the processing time. Regarding post-processing stages, in the CPU implementation the most time consuming phase is the LRCCheck algorithm, while in the GPU approach the occluded pixel filling algorithm represents the most intensive one since it cannot be parallelized and thus runs on the CPU. Finally, despite having a small impact on the GPU implementation, memory transfers between devices are an important part of the pipeline.



(a) CPU implementation taking approximately 153 seconds



(b) GPU implementation taking approximately 500 milliseconds

Fig. 16: CPU vs GPU pipeline comparison for an 820x1142 resolution image, with the parameters reference combination. The reported speedup for the pipeline is 307×. Please note that the scale in a) represents seconds while in b) milliseconds.

Table 4 shows the speedup obtained for each individual kernel. The LRRCheck kernel presents a speedup of 627x, the highest for all analyzed kernels. Regarding pipeline speedup for each image resolution, a massive speedup is ob-

served in the achieved processing times, in Table 5, as each image resolution registers a boost of 173x, 217x, 252x, 291x and 307x, respectively.

Table 4: Individual kernel speedup for an 820x1142 resolution image, with the parameters reference combination.

Kernel	CPU (s)	GPU (ms)	Speedup
SymStereo	53.01	300.34	**353**×
Aggregation	23.64	176.81	**267**×
LRCCheck	0.06	0.09	**627**×
Disp. Enhan.	0.02	18.58	**[N/A]**
2D-3D	0.01	0.14	**77**×

Table 5: CPU vs GPU and speedup for each image resolution, with the parameters reference combination.

Resolution	CPU (s)	GPU (ms)	Speedup
256x256	1.31	7.56	**173**×
288x384	1.80	8.28	**217**×
300x400	6.70	26.59	**252**×
375x1242	47.44	163.03	**291**×
820x1142	153.56	500.21	**307**×

6 Conclusion

The real-time pipeline presented in this paper is a robust dense stereo estimation method capable of computing two 3D maps per second, for high-resolution images, and 132 volumes per second, for low resolution images. With the tests presented in this paper, we have shown how the pipeline reacts to the alteration of its parameters and how that affects processing times and the final quality of 3D maps.

As stated previously in the literature [1], the logN algorithm performs very well for most kinds of scenarios, especially when dealing with slanted surfaces. We decided to explore it a bit further and concluded that for a specific combination of parameters, slanted surfaces can be accurately and robustly reconstructed. Unfortunately this causes textured zones to be less precise, especially around the discontinuities.

Besides impacting visual results, changing parameters also impacts processing times. Three parameters are highly influential: the number of wavelets, the disparity range and the aggregation window. It is highly important to reach a compromise between these three variables in order to combine good visual results and real-time processing capabilities.

Despite the reported good performance, the proposed pipeline still has room to evolve, especially in the aggregation and post processing stages. Several window-based aggregation schemes can be applied in order to analyze more deeply the output results in the aggregation section of the pipeline. Also, a new disparity enhancement algorithm is needed, one that is parallelizable since memory transfers between the device and host's memory must be avoided whenever possible.

Acknowledgements This work was supported by the Portuguese Foundation for Science and Technology (FCT), with FEDER/COMPETE program funding, under Grants AMS-HMI12: RECI/EEI-AUT/0181/2012, UID/EEA/50008/2013 and also by a Google Faculty Research Award from Google Inc. This research was also carried out at the Multimedia Signal Processing Lab, Instituto de Telecomunicações, an NVIDIA GPU Research Center from the University of Coimbra, Portugal.

References

1. Michel Antunes and João P Barreto. SymStereo: Stereo Matching using Induced Symmetry. *International Journal of Computer Vision*, pages 1–22, 2014.
2. Stan Birchfield and Carlo Tomasi. A Pixel Dissimilarity Measure That Is Insensitive to Image Sampling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20:401–406, 1998.
3. Ramin Zabih and John Woodfill. Non-Parametric Local Transforms for Computing Visual Correspondence. In Jan-Olof Eklundh, editor, *Computer Vision — ECCV '94*, volume 801 of *Lecture Notes in Computer Science*, pages 151–158. Springer Berlin Heidelberg, 1994.
4. NVIDIA Corporation. CUDA Zone. [Online]. Available: `https://developer.nvidia.com/cuda-zone`, 2014.
5. AMD. OpenCL Zone. [Online]. Available: `https://developer.amd.com/tools-and-sdks/opencl-zone/`, 2014.
6. K. Hill, S. Craciun, A. George, and H. Lam. Comparative analysis of opencl vs. hdl with image-processing kernels on stratix-v fpga. In *Application-specific Systems, Architectures and Processors (ASAP), 2015 IEEE 26th International Conference on*, pages 189–193, July 2015.
7. C. Rodriguez-Donate, G. Botella, C. Garcia, E. Cabal-Yepez, and M. Prieto-Matias. Early experiences with opencl on fpgas: Convolution case study. In *Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on*, pages 235–235, May 2015.
8. Joao Maria, Joao Amaro, Gabriel Falcao, and Luís A. Alexandre. Stacked autoencoders using low-power accelerated architectures for object recognition in autonomous systems. *Neural Processing Letters*, pages 1–14, 2015.
9. Biao Wang, M. Alvarez-Mesa, Chi Ching Chi, and B. Juurlink. Parallel h.264/avc motion compensation for gpus using opencl. *Circuits and Systems for Video Technology, IEEE Transactions on*, 25(3):525–531, March 2015.
10. G. Falcao, V. Silva, L. Sousa, and J. Andrade. Portable LDPC Decoding on Multicores using OpenCL [Applications Corner]. *Signal Processing Magazine, IEEE*, 29(4):81–109, July 2012.
11. Zhengyou Zhang and Ying Shan. A Progressive Scheme for Stereo Matching. In Marc Pollefeys, Luc Van Gool, Andrew Zisserman, and Andrew Fitzgibbon, editors, *3D Structure from Images — SMILE 2000*, volume 2018 of *Lecture Notes in Computer Science*, pages 68–85. Springer Berlin Heidelberg, 2001.

12. Daniel Scharstein and Richard Szeliski. A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms. *International Journal of Computer Vision*, 47(1-3):7–42, 2002.
13. S.M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A Comparison and Evaluation of Multi-View Stereo Reconstruction Algorithms. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 519–528, June 2006.
14. Jian Sun, Nan-Ning Zheng, and Heung-Yeung Shum. Stereo Matching using Belief Propagation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(7):787–800, July 2003.
15. Y. Boykov, O. Veksler, and R. Zabih. Fast Approximate Energy Minimization Via Graph Cuts. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 23(11):1222–1239, Nov 2001.
16. V. Kolmogorov and R. Zabih. Computing Visual Correspondence with Occlusions using Graph Cuts. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 2, pages 508–515 vol.2, 2001.
17. S. Birchfield and C. Tomasi. Depth Discontinuities by pixel-to-pixel Stereo. In *Computer Vision, 1998. Sixth International Conference on*, pages 1073–1080, Jan 1998.
18. P. Anandan. A Computational Framework and an Algorithm for the Measurement of Visual Motion. *International Journal of Computer Vision*, 2(3):283–310, 1989.
19. Larry H. Matthies, Richard Szeliski, and Takeo Kanade. Kalman Filter-based Algorithms for Estimating Depth from Image Sequences. *Int. J. Computer Vision*, 3(3):209–236, September 1989.
20. T. Kanade and M. Okutomi. A stereo matching algorithm with an adaptive window: theory and experiment. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 16(9):920–932, Sep 1994.
21. Kuk-Jin Yoon and In So Kweon. Adaptive Support-Weight Approach for Correspondence Search. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(4):650–656, April 2006.
22. Hao Tang and Zhigang Zhu. Content-Based 3-D Mosaics for Representing Videos of Dynamic Urban Scenes. *Circuits and Systems for Video Technology, IEEE Transactions on*, 22(2):295–308, Feb 2012.
23. Z.Y. Zhu, S. Zhang, S.C. Chan, and H.Y. Shum. Object-Based Rendering and 3D Reconstruction using a Moveable Image-Based Rendering System. In *Multidimensional (nD) Systems (nDs), 2011 7th International Workshop on*, pages 1–4, Sept 2011.
24. R. Ferrari Pinto, A.G.S. Conceicao, P.C.M.A. Farias, and E.T.F. Santos. A Cost Effective Open-Source Three-Dimensional Reconstruction System and Trajectory Analysis for Mobile Robots. In *Biosignals and Biorobotics Conference (2014): Biosignals and Robotics for Better and Safer Living (BRC), 5th ISSNIP-IEEE*, pages 1–5, May 2014.
25. A. Zia, Jie Liang, Jun Zhou, and Yongsheng Gao. 3d reconstruction from hyperspectral images. In *Applications of Computer Vision (WACV), 2015 IEEE Winter Conference on*, pages 318–325, Jan 2015.
26. S. Yamao, M. Miura, S. Sakai, K. Ito, and T. Aoki. A sequential online 3d reconstruction system using dense stereo matching. In *Applications of Computer Vision (WACV), 2015 IEEE Winter Conference on*, pages 341–348, Jan 2015.
27. K. Ge, H. Hu, J. Feng, and J. Zhou. Depth estimation using a sliding camera. *Image Processing, IEEE Transactions on*, 25(2):726–739, Feb 2016.
28. J. Kowalczuk, E.T. Psota, and L.C. Perez. Real-Time Stereo Matching on CUDA using an Iterative Refinement Method for Adaptive Support-Weight Correspondences. *Circuits and Systems for Video Technology, IEEE Transactions on*, 23(1):94–104, Jan 2013.
29. Sang Hwa Lee and S. Sharma. Real-Time Disparity Estimation Algorithm for Stereo Camera Systems. *Consumer Electronics, IEEE Transactions on*, 57(3):1018–1026, August 2011.
30. Ke Zhang, Jiangbo Lu, Qiong Yang, G. Lafruit, R. Lauwereins, and L. Van Gool. Real-Time and Accurate Stereo: A Scalable Approach with Bitwise Fast Voting on CUDA. *Circuits and Systems for Video Technology, IEEE Transactions on*, 21(7):867–878, July 2011.
31. Jinglin Zhang, J.-F. Nezan, M. Pelcat, and J.-G. Cousin. Real-time gpu-based local stereo matching method. In *Design and Architectures for Signal and Image Processing (DASIP), 2013 Conference on*, pages 209–214, Oct 2013.
32. Ailin Yang, Xiuzhi Li, Songmin Jia, and Baoling Qin. Monocular three dimensional dense surface reconstruction by optical flow feedback. In *Information and Automation, 2015 IEEE International Conference on*, pages 504–509, Aug 2015.
33. Qian Long, Qiwei Xie, S. Mita, K. Ishimaru, and N. Shirai. A real-time dense stereo matching method for critical environment sensing in autonomous driving. In *Intelligent Transportation Systems (ITSC), 2014 IEEE 17th International Conference on*, pages 853–860, Oct 2014.
34. V. Mota, G. Falcao, M. Antunes, J. Barreto, and U. Nunes. Using the gpu for fast symmetry-based dense stereo matching in high resolution images. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 7520–7524, May 2014.
35. R. Ralha, G. Falcao, J. Andrade, M. Antunes, J.P. Barreto, and U. Nunes. Distributed dense stereo matching for 3d reconstruction using parallel-based processing advantages. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 1126–1130, April 2015.
36. R. Szeliski and D. Scharstein. Sampling the Disparity Space Image. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(3):419–425, March 2004.
37. Robert T. Collins. A Space-Sweep Approach to True Multi-Image Matching. In *Proceedings of the 1996 Conference on Computer Vision and Pattern Recognition (CVPR '96)*, CVPR '96, pages 358–, Washington, DC, USA, 1996. IEEE Computer Society.
38. Peter Kovesi. Symmetry and Asymmetry from Local Phase. In *Tenth Australian Joint Conference on Artificial Intelligence*, 1997.
39. Peter Kovesi. Image Features from Phase Congruency. Technical report, Videre: Journal of Computer Vision Research, 1995.
40. NVIDIA Corporation. NVIDIA's Next Generation CUDA Compute Architecture: Kepler GK110. 2012.
41. NVIDIA Corporation. cuFFT. [Online]. Available: `https://developer.nvidia.com/cuFFT`, 2014.
42. Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets Robotics: The KITTI Dataset. *International Journal of Robotics Research (IJRR)*, 2013.
43. Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.
44. C. Richardt, D. A. H. Orr, I. P. Davies, A. Criminisi, and N. A. Dodgson. Real-time Spatiotemporal Stereo Matching using the Dual-Cross-Bilateral Grid. In *European Conference on Computer Vision (ECCV)*. Springer Verlag, 2010.
45. Andreas Geiger, Martin Roser, and Raquel Urtasun. Efficient Large-Scale Stereo Matching. In Ron Kimmel, Reinhard Klette, and Akihiro Sugimoto, editors, *Computer Vision – ACCV 2010*, volume 6492 of *Lecture Notes in Computer Science*, pages 25–38. Springer Berlin Heidelberg, 2011.