2026 IEEE International Conference on Robotics and Automation (ICRA 2026)
June 1-5, 2026. Vienna, Austria

Robust and Real-time Surface Normal Estimation from Stereo Disparities using Affine Transformations

Muhammad Rafi Faisal, Csongor Csanád Karikó and Levente Hajder

***Abstract*— This work introduces a novel method for surface normal estimation from rectified stereo image pairs, leveraging affine transformations derived from disparity values to achieve fast and accurate results. We demonstrate how the rectification of stereo image pairs simplifies the process of surface normal estimation by reducing computational complexity. To address noise reduction, we develop a custom algorithm inspired by convolutional operations, tailored to process disparity data efficiently. We also introduce adaptive heuristic techniques for efficiently detecting connected surface components within the images, further improving the robustness of the method. By integrating these methods, we construct a surface normal estimator that is both fast and accurate, producing a dense, oriented point cloud as the final output. Our method is validated using both simulated environments and real-world stereo images from the Middlebury[1] and Cityscapes datasets, demonstrating significant improvements in real-time performance and accuracy when implemented on a GPU. The source code is available at `https://github.com/mrafifaisal/Surface-Normal-Estimation/`.**

I. INTRODUCTION

Obtaining oriented point clouds is a crucial challenge in both computer vision and robotics, with numerous real-world applications exploiting the advantages of the additional information they offer. These applications encompass tasks such as 3D reconstruction [1] or simultaneous localization and mapping [2] (SLAM), where the orientation of 3D surface points offers valuable insights into the underlying structure. See figure 1 for an example. Image-based visual localization methods [3] can also leverage knowledge of surface normals. In the context of autonomous vehicles, understanding surface orientation can enhance tasks like ground or facade segmentation within visual odometry [4]. Furthermore, the use of oriented point clouds greatly simplifies geometric model and multi-model estimation [5], [6] by enabling the development of solvers that require fewer data points than their point-based counterparts. Additionally, surface normals play a significant role in the numerical refinement of the reconstructed oriented 3D point cloud [7].

Nowadays, 3D reconstruction pipelines usually serve the result as 3D point clouds, and the surface orientation is computed from spatial points. This paper principally addresses the rapid and accurate estimation of surface normals based on affine correspondences.

*The author gratefully acknowledges the financial support provided by Robert Bosch Group through their scholarship program.

Authors are with Geometric Computer Vision Group, Faculty of Informatics, Eötvös Loránd University, Budapest, Hungary {goo2im,jpoiwr,hajder}@inf.elte.hu

[1]For the Middlebury datasets, disparity values are published.

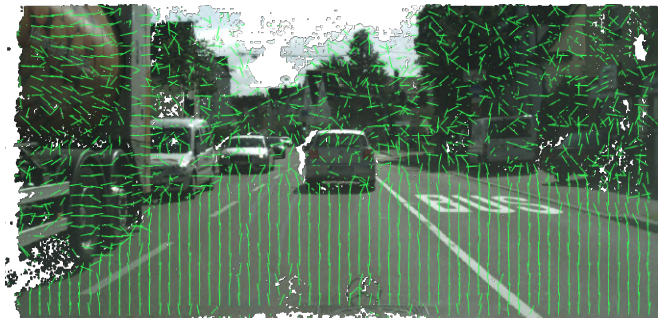


Fig. 1: Estimated surface normals in a vehicle-mounted camera image from the Cityscapes dataset [8]. Normals for the road are vertical, aiding segmentation from other objects.

A. Surface Normal Estimation

There are several different techniques to estimate surface normals from stereo images, they are overviewed here.

Surface normal from point clouds: Upon applying state-of-the-art sparse [9] or dense [10] reconstruction pipelines, the resultant point cloud can be readily endowed with point orientations. In this scenario, it is common practice to estimate normals on a per-point basis, employing the widely accepted Principal Component Analysis (PCA) [11] to deduce tangent planes from the immediate local vicinity surrounding each point. Nan Ming et al. [12] propose minimizing the depth Laplacian through a dynamic programming approach. Higher-order surfaces [13] can also be reconstructed from point clouds. Nevertheless, this approach has two significant drawbacks. First, the choice of neighborhood size significantly impacts accuracy and is often uncertain in the absence of a metric reconstruction. Alternatively, using the k-nearest-neighbors algorithm to determine local neighborhoods is a feasible solution, but the estimation is heavily influenced by the sparsity of the point cloud. Second, independently fitting local surfaces to the neighborhood of each 3D point in the reconstruction is an exceedingly resource-intensive operation.

From affine transformations: Recently, the mainstream point-based reconstruction systems have been extended by the application of affine transformations [7]. An affine transformation connects two small corresponding regions between two images. They are the linear local approximation of differentiable $\mathbb{R}^2 \longrightarrow \mathbb{R}^2$ functions. The pioneering work in this area was the PhD thesis of Kevin Koser [14] in 2008. Molnar and Chetverikov introduced a general formula [15] that shows how surface normals, projected coordinates, and

camera parameters determine the related affine transformation. Later on, Barath et al. [16] showed how homographies can be estimated from fewer correspondences if affine parameters are also available. Remark that the normal can be retrieved from a homography [1] if the camera intrinsic parameters are known.

As far as we know, the first normal estimator for a general pin-hole camera pair was proposed by Molnar et al. [17] in 2014. Since their work, faster and more accurate methods [18], [19], [20] have appeared in the literature.

Surface normals via machine learning: Nowadays, machine learning has become a frequently used method in the computer vision community. There are specific networks [21], [22] for normal estimation as well. However, their results highly depend on the training data, and one of our main purposes is to develop a deterministic approach as surface normal can be computed from the disparities.

B. Goals

Although there are general estimators [18], [19], [20] to compute the surface normals from an affine correspondence if the cameras are calibrated, there are many open problems: (i) The *speed is crucial* especially *in real-time applications* if a very dense oriented point cloud is required as a result. (ii) The accuracy of surface normal estimation is *very sensitive* to the quality of the affine transformations, sophisticated filtering methods have not been proposed before, to the best of our knowledge. (iii) Affine transformations are the first-order approximations of the projection between corresponding image patches. However, theoretically, the approximation is valid only if the pixels are related to the same surfaces. At their borders, multiple surfaces might contribute pixels to the estimation of the same normal vector. A *major* task is therefore to detect the borders of these surfaces. We address all these three listed goals in our paper.

C. Contribution

The main theoretical novelty here is that rectified images are applied for affine transformation estimation for the sake of lower processing time. This special stereo image setup can be achieved from a general image pair if the epipoles are not on the image. There are efficient methods to rectify image pairs [23]. Therefore, it is not a strict restriction.

To the best of our knowledge, only Megyesi et al. [24] dealt with the application of affine transformations for rectified images. Their method is not complete as they only proposed a simple formula to determine the normal from the affine transformation and that formula does not consider the camera intrinsic parameters. Their work highly motivated us, but we replace their formula with an alternative one, including camera intrinsics, and we also deal with the accurate estimation of affine transformations and border detection of planar regions.

As speed is also a critical aspect, the application of GPUs is preferred. Modern GPU architectures are massively parallel and have a certain set of limitations that need to be taken into account when designing and implementing algorithms for them. The SIMD architecture employed by modern GPUs executes the same instruction in parallel over multiple sets of data. Therefore it is usually costly to do a lot of branching if the code is expected to be executed differently on neighboring pixels. It is also generally advisable to use as little extra memory as possible per pixel. If the required memory is too much to be entirely stored in local registers, performance degrades significantly. We designed our algorithms considering these limitations.

The key **contribution** of our work **is fourfold**: **(i)** It is shown how the normal estimation is simplified for a rectified image stereo pair. **(ii)** We propose a novel method here that computes the normal vector from affine transformations determined between rectified images. **(iii)** A novel algorithm is also introduced to retrieve the affine parameters from disparity values. It works for both the minimal and over-determined cases. **(iv)** An adaptive method for border detection of regions corresponding to flat surfaces is proposed.

Our approach is purely geometric, without any machine learning components. Therefore, learning data is not required, the results are **deterministic**, it depends only on the disparities. Nevertheless, the proposed methods are straightforwardly differentiable, with the exception of the adaptive estimation, thus the approach can be built into a complex system as a component for which end-to-end training is possible. Moreover, our geometry-based solution contains a convolution, this fact suggests that there can are similarities in the operations of Convolutional Neural Networks (CNNs) and our geometric approach.

II. Theoretical Background

The relationship between inter-region affine transformations of stereo images and the surface normal has been studied previously by Barath et al. [18]. Here, we deduce the related formulas for a rectified stereo pair.

In general, the affine transformation is derived from the surface normal as follows:

$$\mathbf{A} = \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} = \frac{1}{\mathbf{n}^T \mathbf{w_5}} \begin{bmatrix} \mathbf{n}^T \mathbf{w_1} & \mathbf{n}^T \mathbf{w_2} \\ \mathbf{n}^T \mathbf{w_3} & \mathbf{n}^T \mathbf{w_4} \end{bmatrix}, \tag{1}$$

where $\mathbf{n}$ denotes the surface normal and $\mathbf{w_1}$ through $\mathbf{w_5}$ are calculated from the gradients of the projection matrices. These can be written as follows:

$\mathbf{w_1} = \nabla\Pi_{1v} \times \nabla\Pi_{2u}$, $\mathbf{w_2} = \nabla\Pi_{2u} \times \nabla\Pi_{1u}$, $\mathbf{w_3} = \nabla\Pi_{1v} \times \nabla\Pi_{2v}$, $\mathbf{w_4} = \nabla\Pi_{2v} \times \nabla\Pi_{1u}$ and $\mathbf{w_5} = \nabla\Pi_{1v} \times \nabla\Pi_{1u}$.

In the formulas, $\Pi_i(x,y,z)$, $i \in \{1,2\}$, are the projection functions transforming world to image coordinates as

$$\Pi_i(x,y,z) = \begin{bmatrix} \Pi_{iu}(x,y,z) \\ \Pi_{iv}(x,y,z) \end{bmatrix}. \tag{2}$$

Rectified stereo images. Figure 2 shows the special case of a rectified stereo setup with pin-hole cameras when the intrinsic parameters are the same for both cameras. In this case the projection functions can be written as

$$\Pi_1(x,y,z) = \frac{1}{z} \begin{bmatrix} f_x x + u_0 z \\ f_y y + v_0 z \end{bmatrix}, \tag{3}$$

$$\Pi_2(x,y,z) = \frac{1}{z} \begin{bmatrix} f_x (x-b) + u_0 z \\ f_y y + v_0 z \end{bmatrix}, \tag{4}$$

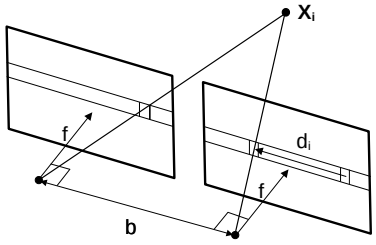


Fig. 2: Geometric setup for rectified stereo. It is represented by baseline b and common focal length f. The inputs are the disparity values d_i coming from the projection of spatial points $\mathbf{X}_i$.

where x, y, z are the world coordinates, f_x and f_y are the horizontal and vertical focal lengths in pixels, b is the baseline distance between the cameras and $[u_0, v_0]^T$ is the principal point in which the optical axis intersects the image plane.

The gradients of the projective functions for the pin-hole camera can be written as follows:

$$\nabla\Pi_{1u} = \frac{1}{z^2}\begin{bmatrix} f_x z & 0 & -f_x x \end{bmatrix}^T, \tag{5}$$

$$\nabla\Pi_{1v} = \frac{1}{z^2}\begin{bmatrix} 0 & f_y z & -f_y y \end{bmatrix}^T, \tag{6}$$

$$\nabla\Pi_{2u} = \frac{1}{z^2}\begin{bmatrix} f_x z & 0 & f_x(b-x) \end{bmatrix}^T, \tag{7}$$

$$\nabla\Pi_{2v} = \nabla\Pi_{1v}. \tag{8}$$

After calculating the cross product of the respective gradients, one can get for $\mathbf{w_1} \dots \mathbf{w_5}$ that

$$\mathbf{w_1} = \frac{f_x f_y}{z^3}\begin{bmatrix} b-x & -y & -z \end{bmatrix}^T, \tag{9}$$

$$\mathbf{w_2} = \frac{f_x^2 b}{z^3}\begin{bmatrix} 0 & 1 & 0 \end{bmatrix}^T, \tag{10}$$

$$\mathbf{w_3} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T, \tag{11}$$

$$\mathbf{w_4} = \mathbf{w_5} = -\frac{f_x f_y}{z^3}\begin{bmatrix} x & y & z \end{bmatrix}^T. \tag{12}$$

Substituting (9), (10), (11), (12) into (1) yields

$$\mathbf{A} = \begin{bmatrix} \frac{\mathbf{n}^T\mathbf{w_1}}{\mathbf{n}^T\mathbf{w_4}} & \frac{\mathbf{n}^T\mathbf{w_2}}{\mathbf{n}^T\mathbf{w_4}} \\ \frac{\mathbf{n}^T\mathbf{0}}{\mathbf{n}^T\mathbf{w_4}} & \frac{\mathbf{n}^T\mathbf{w_4}}{\mathbf{n}^T\mathbf{w_4}} \end{bmatrix} = \begin{bmatrix} a_1 & a_2 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{n_x(x-b)+n_y y+n_z z}{n_x x+n_y y+n_z z} & \frac{-b f_x n_y}{f_y(n_x x+n_y y+n_z z)} \\ 0 & 1 \end{bmatrix}. \tag{13}$$

III. Proposed Method

In this section, it is shown how a robust surface normal estimator for a rectified stereo image pair can be formed. First, it is deduced how the normal can be determined using the affine parameters. Then it is derived how these affine parameters can be computed from disparity values. Finally, an adaptive method is proposed to robustly detect the area from which disparities are selected.

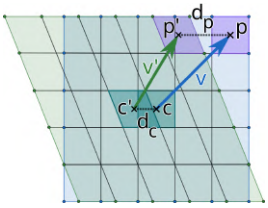


Fig. 3: Two corresponding regions (squares) of the stereo image pair. The region in the left image is estimated by applying an affine transformation to its corresponding pair in the right image. This figure shows the original and the estimated regions overlapped. $(\mathbf{c}, \mathbf{c}')$ and $(\mathbf{p}, \mathbf{p}')$ denote two correspondences between the two regions with respective disparities d_c and d_p. $\mathbf{c}, \mathbf{c}'$ are central pixels of the two regions, the pixels at where the affine transformation is estimated and normal calculated. $\mathbf{v} = \mathbf{p} - \mathbf{c}$, $\mathbf{v}' = \mathbf{p}' - \mathbf{c}'$. Note that $\mathbf{v}' = \mathbf{A}\mathbf{v}$.

A. Normal Calculation

If we apply the Fast Normal Estimation (FNE) formulas from [18] in order to calculate the normal, this special case always yields a zero vector as $a_3 = 0$ is a scalar multiplier. For this reason, we derived a new method for calculating the normal in this special case.

Suppose we know the affine parameters a_1 and a_2. This yields two equations from Eq.(13):

$$a_1(n_x x + n_y y + n_z z) = n_x(x-b) + n_y y + n_z z, \tag{14}$$

$$a_2 f_y(n_x x + n_y y + n_z z) = -b f_x n_y \tag{15}$$

Let us introduce the following vectors if $\mathbf{X} = [x, y, z]^T$:

$$\mathbf{v_1} = \mathbf{X} \qquad \mathbf{v_2} = [x-b, y, z], \tag{16}$$

$$\mathbf{v_3} = f_y\mathbf{X} \qquad \mathbf{v_4} = [0, -b f_x, 0]. \tag{17}$$

If they are substituted to Eqs.(14) and (15), after elementary modifications, then two new formulas can be written as:

$$\mathbf{n}^T(a_1\mathbf{v_1} - \mathbf{v_2}) = 0, \quad \mathbf{n}^T(a_2\mathbf{v_3} - \mathbf{v_4}) = 0.$$

Since $\mathbf{n}$ is perpendicular to both $(a_1\mathbf{v_1} - \mathbf{v_2})$ and $(a_2\mathbf{v_3} - \mathbf{v_4})$, one can calculate $\mathbf{n}$ up to a scalar multiple by computing their cross product. Expressed with the original parameters, this results in the following formula:

$$\mathbf{n} = \begin{bmatrix} k_h z(a_1 - 1) \\ a_2 k_v z \\ -a_1 k_h x - a_2 k_v y + b k_h + k_h x \end{bmatrix}. \tag{18}$$

where k_h and k_v denote the horizontal and vertical pixel densities of the camera sensor. Alternatively f_x may be used instead of k_h and f_y instead of k_v.

B. Calculating the Affine Parameters

In order to use the formula presented above, we need to know the affine transformation between two corresponding image patches. Here, we present a novel robust method for the efficient calculation of the required two affine parameters from disparity maps [25].

Let us examine a pair of corresponding points on the two images, and their surrounding pixels. One of the patches transformed by the respective affine matrix approximately matches the corresponding area on the other camera image. The matching problem is visualized in figure 3. If the disparity map is known, the following equation can be written:

$$\mathbf{v} + [d_p, 0]^T = [d_c, 0]^T + \mathbf{v}'. \tag{19}$$

Note that d_p and d_c can be negative when going from the left image to the right one.

Furthermore, $\mathbf{v}'$ may be expressed using the affine transformation ($\mathbf{v}' = \mathbf{A}\mathbf{v}$), which yields

$$\begin{bmatrix} d_p - d_c \\ 0 \end{bmatrix} = (\mathbf{A} - \mathbf{I})\mathbf{v} = \begin{bmatrix} a_1 - 1 & a_2 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix}. \tag{20}$$

The upper row gives a linear equation for each surrounding pixel. With N selected pixels, one can get an over-determined linear system of equations as follows:

$$\underbrace{\begin{bmatrix} v_{1x} & v_{1y} \\ v_{2x} & v_{2y} \\ \vdots & \vdots \\ v_{Nx} & v_{Ny} \end{bmatrix}}_{\mathbf{V}} \underbrace{\begin{bmatrix} a_1 - 1 \\ a_2 \end{bmatrix}}_{\mathbf{a}} = \underbrace{\begin{bmatrix} d_1 - d_c \\ d_2 - d_c \\ \vdots \\ d_N - d_c \end{bmatrix}}_{\mathbf{d}}. \tag{21}$$

Using the well-known least squares estimation of an inhomogeneous linear system of equations, we get the affine parameters as $\mathbf{a} = \left(\mathbf{V}^T\mathbf{V}\right)^{-1}\mathbf{V}^T\mathbf{d}$.

C. Affine Parameter Calculation by Convolution

Suppose the relative position of selected surrounding pixels is the same for all observed pixels. For example we always select an $n \times n$ square centered at the observed pixel. In this case, the entire $\mathbf{S} = \left(\mathbf{V}^T\mathbf{V}\right)^{-1}\mathbf{V}^T$ matrix can be precalculated as it only depends on the shape of the selected area, its disparity values do not influence the elements in $\mathbf{S}$. *Precalculation of* $\mathbf{S}$ *can significantly reduce the time demand of the algorithm.*

By expanding $\mathbf{S}$, the following formula is obtained for the convolution kernels:

$$\mathbf{V}^T\mathbf{V} = \begin{bmatrix} \underbrace{\sum_{i=1}^{N} v_{ix}^2}_{\alpha} & \underbrace{\sum_{i=1}^{N} v_{ix}v_{iy}}_{\beta} \\ \underbrace{\sum_{i=1}^{N} v_{ix}v_{iy}}_{\beta} & \underbrace{\sum_{i=1}^{N} v_{iy}^2}_{\gamma} \end{bmatrix}. \tag{22}$$

Therefore,

$$\mathbf{S} = \frac{1}{\alpha\gamma - \beta^2} \begin{bmatrix} \gamma v_{1x} - \beta v_{1y} & \dots & \gamma v_{Nx} - \beta v_{Ny} \\ -\beta v_{1x} + \alpha v_{1y} & \dots & -\beta v_{Nx} + \alpha v_{Ny} \end{bmatrix}. \tag{23}$$

When calculating the affine parameters, we just have to construct $\mathbf{d}$ for each observed pixel, then it multiplies by the precalculated elements of $\mathbf{S}$ from the left as

$s_{1,1}$ $[-1,-1]$	$s_{1,2}$ $[-1,0]$	$s_{1,3}$ $[-1,1]$
$s_{1,4}$ $[0,-1]$	$s_{1,5}$ $[0,0]$	$s_{1,6}$ $[0,1]$
$s_{1,7}$ $[1,-1]$	$s_{1,8}$ $[1,0]$	$s_{1,9}$ $[1,1]$

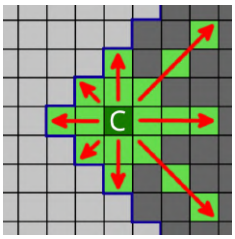


Fig. 4: **Left.** An example of a 3×3 kernel for computing a_1. The listed coefficients s_i and their corresponding vectors $\mathbf{v_i}^T$ are in the same cell. **Right.** Visualization of the selected (green) pixels by performing star-fill from C in eight directions.

$$a_1 - 1 = \sum_{i=1}^{N} \underbrace{\frac{1}{\alpha\gamma - \beta^2}(\gamma v_{ix} - \beta v_{iy})}_{s_{1i}} \underbrace{(d_i - d_c)}_{\mathbf{d}_i}, \tag{24}$$

$$a_2 = \sum_{i=1}^{N} \underbrace{\frac{1}{\alpha\gamma - \beta^2}(-\beta v_{ix} + \alpha v_{iy})}_{s_{2i}} \underbrace{(d_i - d_c)}_{\mathbf{d}_i}. \tag{25}$$

Note that other than the precalculated α, β and γ constants, s_{1i} and s_{2i} are both only dependent on the relative position vector $\mathbf{v_i}$ of the ith pixel.

With some rearrangement, we can decompose the calculation of each parameter into two steps. A discrete 2D convolution of the disparity map with a special precalculated kernel, and a second step consisting of a multiplication by a precalculated constant and a subtraction:

$$a_1 = \sum_{i=1}^{N} s_{1i}(d_i - d_c) = \underbrace{\sum_{i=1}^{N} s_{1i}d_i}_{\text{Step 1}} - d_c \underbrace{\overbrace{\sum_{i=1}^{N} s_{1i}}^{\delta_1} + 1}_{\text{Step 2}}, \tag{26}$$

$$a_2 = \sum_{i=1}^{N} s_{2i}(d_i - d_c) = \underbrace{\sum_{i=1}^{N} s_{2i}d_i}_{\text{Step 1}} - d_c \underbrace{\overbrace{\sum_{i=1}^{N} s_{2i}}^{\delta_2}}_{\text{Step 2}}.$$

Inside the summations of equation (26), s_{1i} and s_{2i} are only multiplied with the disparity of the pixel at $\mathbf{v_i}$. Therefore, this is a 2D convolution with the kernel containing $s_{1,1}, s_{1,2}, \dots, s_{1,N}$ for the computation of a_1, and $s_{2,1}, s_{2,2}, \dots, s_{2,N}$ for a_2.

In both cases, the first step is a 2D convolution of the disparity map with two kernels. This is followed by the other operations involving the precomputed δ_1 and δ_2, defined in equation (26). An example kernel layout is shown in the left plot of figure 4.

The bulk of the computation is in the calculation of the 2D convolutions. In practice, this is efficiently sped up using the Fast Fourier Transform (FFT) algorithm either on CPU or GPU [26], [27]. However, according to our testing, on

modern GPUs and with reasonable kernel sizes, the method is already very fast without the additional performance gain of FFT.

D. Adaptive Region Estimation

A very important challenge for robust surface normal estimators is maintaining accuracy around corners and edges. The main problem here is that including points from a neighboring surface will yield less accurate normals. This usually appears as a smoothing of normals around edges, which is especially problematic when the results are used for segmentation tasks. Ming et al. [12] propose minimizing the depth Laplacian via dynamic programming. Feng et al. [28] also propose a post processing step for correcting the smoothing effect around edges.

We introduce a dynamically shaped kernel for our method described above. The main idea is to iteratively extend the patch of included points, starting from the observed pixel. As the same procedure should be run for all pixels, it can be strongly parallelized in GPU cores.

A natural method of choice would be to use flood filling for the selection of included pixels. However, in our case, we have to run this selection algorithm for each pixel. Instead, we introduce a simplified technique, for the sake of a more efficient GPU implementation. Our algorithm starts from the observed pixel at the center and traverses several lines in M uniformly distributed directions. The traversal either stops when a maximum length is reached, or when a stopping condition is met. All visited pixels are included in the final normal estimation. The selected pixels will be in star-like shape as it is visualized in the right image of figure 4.

Once we have a list of included pixels, we need to do two passes. During the first pass, parameters α, β and γ are computed. In the second pass, the affine parameters are estimated using equation (24). and equation (25).

Stopping at edges on the depthmap. (Simple thresholding, ST) As a first step, an edge detection filter of choice is run over the depth map. It stops when an edge pixel surpassing a threshold is reached. We use the depth Laplacian as a measure of discontinuity.

Stopping based on covered depth. (CD) Let us assume that the central pixel has depth d, the surrounding pixels on the same flat surface should be in a range of kd, where k is an adaptive threshold parameter.

With this method, we keep track of the maximum and minimum encountered depth along our traversal. If their difference surpasses kd, then the traversal is stopped. This method adapts much better to wider ranges. Unfortunately, it usually misses depth-continuous edges, like the edge detection based method.

IV. Experimental Results

The dataset used for evaluation is the 3F2N dataset [29], which contains 24 synthetic images with ground truth normals and depth values. We calculated the disparity maps from the depth values, with a unit length baseline (1 meter). Then we added noise with a Gaussian distribution to generate our tests. The dataset is divided into three difficulty levels (Easy, Medium, Hard). Scenes with a higher difficulty contain more discontinuities and fine detail.

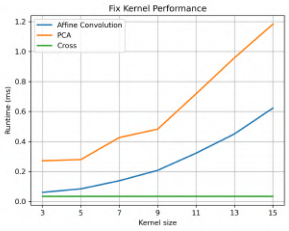


Fig. 5: Comparison of execution time depending on kernel size. The naive cross product execution time is also displayed for reference. Average over 100 frames at resolution 480x620.

The proposed method has been implemented in an OpenGL shader, along with some other methods for reference. Tests have been conducted on a laptop equipped with an Nvidia RTX 4060 (mobile) GPU and an AMD Ryzen 7 7840HS CPU.

Implemented algorithms. We have developed all variants of the proposed approach. Additionally, a PCA-based solution and a naive implementation are also examined in the tests. The naive solution is denoted by 'Cross' in the tests, when the normal is calculated from two tangent vectors. These vectors are determined by triangulation of the processed point and the horizontal and vertical neighbouring pixels.

The PCA-based normal estimation is also our own implementation, we set its parameters by an exhaustive search to tune it for the best results. The essence of the estimation is the eigenvalue/eigenvector computation of a 3×3 matrix, we applied closed-form formulas for the sake of speed.

Moreover, we tested the SDA-SNE [12] method as well, which is claimed to be robust to noise. Unfortunately, a GPU implementation for the technique is not available at the moment, therefore we compared with the original CPU implementation, mainly for comparing accuracy.

Runtime. To avoid overhead from unrelated rendering tasks, we exclusively measured the GPU execution time of the shaders. This was accomplished utilizing OpenGL query objects to get the elapsed GPU time. A comparison between the adaptive methods, PCA, and the fixed kernel version is shown in table I and table II.

The provided SDA-SNE implementation [30] is for the CPU. To compensate for at least the additional time taken by the python interpreter, we ran the original python script through cython, compiled it and ran all tests natively.

Additionally, we have compared the execution time of the methods depending on the kernel size. The affine convolution method shows about twice the performance of PCA. Time demands are reported in figure 5.

It is clearly seen that adaptive methods are slower, as it is expected, but all the values are below $10ms$, therefore real-time operation can be reached even if large kernel size or adaptive disparity selection are applied.

Method	Avg	Min	Max	Std
Cross	0.03	0.03	0.03	0.00
PCA 9x9	0.48	0.48	0.48	0.00
Affine 9x9	0.21	0.21	0.21	0.00
ST s10 d8 t0.1	0.19	0.12	0.30	0.04
CD s10 d8 t0.5	0.24	0.17	0.50	0.07
SDA-SNE	190.35	162.48	220.53	15.25

TABLE I: Statistics of frametimes over 100 frames at a resolution of 480x640. 's' denotes the maximal number of steps taken in each direction, 'd' denotes the number of uniformly distributed directions. The threshold for both conditions is denoted with 't'. All measurements are in milliseconds.

Method	Avg	Min	Max	Med	Std
Affine 9x9	**0.65**	**0.62**	**0.72**	**0.65**	**0.01**
PCA 9x9	1.62	1.55	1.73	1.62	0.03
ST[s:10 d:8]	1.80	1.73	2.07	1.79	0.03
ST[s:30 d:16]	8.15	7.73	8.89	8.14	0.21
CD[s:10 d:8]	1.71	1.66	1.86	1.71	0.03
CD[s:30 d:16]	6.85	6.47	8.01	6.84	0.21

TABLE II: Statistics of frametimes over 1000 frames at a resolution of 1024x720.

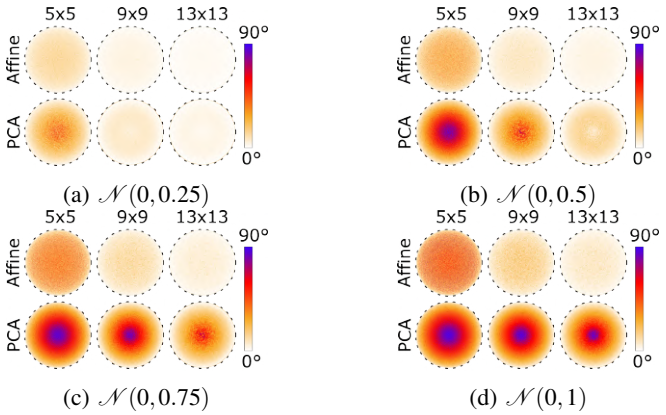
Robustness to Noise. To measure the effect of disparity noise at different viewing angles, we rendered a sphere placed directly in front of the camera. For better accuracy, the sphere has been rendered via raycasting [31], the true normals were calculated per pixel. Disparities were calculated by projecting the ray-sphere intersection to two virtual pinhole cameras and taking the difference of the projected coordinates. Their noise is assumed to have a Gaussian distribution with zero mean. Therefore, we add this $\mathcal{N}(0,\sigma)$ noise to the calculated disparities.

Table III shows our results for different kernel sizes and for a scenario with $\sigma = 1$ px and a more reasonable case with $\sigma = 0.2$ px. For reference, the Middlebury dataset's "Adirondack" scene has an average standard deviation of 0.12 over the samples. In our measurements, the affine convolutional estimator exhibited superior accuracy and consistency over PCA, especially for larger kernel sizes.

As shown in figure 6a and figure 6d, the sensitivity to noise

Method	$\sigma = 0.2$	$\sigma = 1$
Affine 3x3	19.15	51.74
Affine 5x5	6.92	30.52
Affine 9x9	2.22	10.47
Affine 15x15	**0.97**	**3.94**
PCA 3x3	27.44	50.36
PCA 5x5	11.22	42.85
PCA 9x9	3.62	35.30
PCA 15x15	1.55	24.42

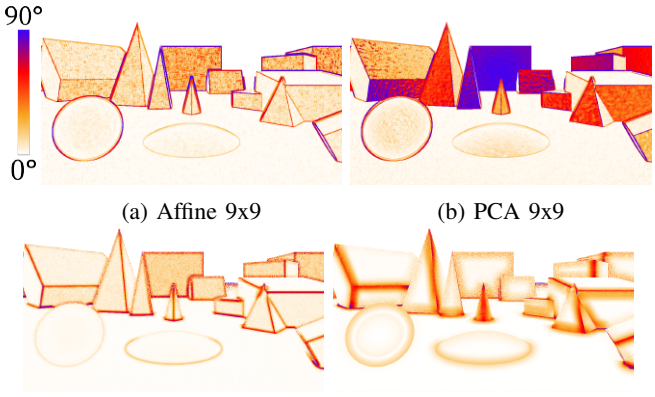
TABLE III: Average angle in degrees between ground truth and estimated normal vectors for a sphere with $\mathcal{N}(0,0.2)$ and $\mathcal{N}(0,1)$ disparity noise (Resolution: 1024x1024). Note that with the same kernel size the affine-based method performs significantly better.



(a) $\mathcal{N}(0,0.25)$ (b) $\mathcal{N}(0,0.5)$

(c) $\mathcal{N}(0,0.75)$ (d) $\mathcal{N}(0,1)$

Fig. 6: Angular normal error of sphere. Radius: 1.4, distance: 3 units from center, baseline: 0.3, resolution: 1024x1024.



(a) Affine 9x9 (b) PCA 9x9

(c) CD s:10 d:8 (d) CD s:30 d:16

Fig. 7: Angular error of estimated normals on synthesized scene. Angles in degrees. $\mathcal{N}(0,0.2)$ noise. The furthest box is of size $4 \times 3 \times 4$ and its center is 15.5 units from the camera focal point. baseline: 0.3, resolution: 1024x720.

tends to depend on the viewing angle. The reasons of it is that the noise in the disparities does not affect all directions uniformly, yielding a distortion to the general shape of the point-cloud. An example for this effect is shown in figure 9. We think that this non-uniform scaling is the reason behind PCA performing worse especially in heavy noise circumstances. Therefore, *the use of affine transformations has theoretical advantages over PCA-based solution(s).*

Additionally, figure 8 shows the accuracy of all methods depending on noise level. SDA-SNE did not show any noise filtering capability, opposed to the claim in the original paper [12].

a) Discontinuities and the adaptive version: Figure 7 shows a comparison of the discussed methods on a test scene, consisting of multiple simple shapes. In this scene, the furthest cube is 15 units away from the cameras. The used baseline is 0.3 and the applied noise is of the distribution $\mathcal{N}(0,0.2)$. Table IV contains the numerical measurements excluding background pixels. The runtime performance of calculating the normals for this scene is displayed in table II.

For the same amount of included pixels, the adaptive

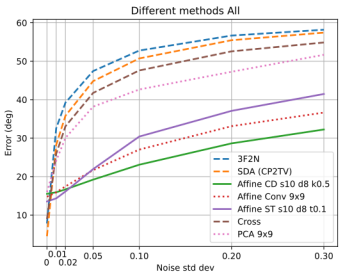


Fig. 8: Comparison of average angular error depending on disparity noise. Complete dataset, images of 480x640 resolution.

Method	Avg	Min	Max	Std
Affine 9x9	9.79	**0.0**	**89.99**	16.27
PCA 9x9	22.07	0.02	90.00	28.07
ST[s:10 d:8]	8.27	**0.0**	**89.99**	13.80
ST[s:30 d:16]	**8.08**	**0.0**	90.00	15.22
CD[s:10 d:8]	6.62	**0.0**	90.00	**12.11**
CD[s:30 d:16]	7.52	**0.0**	90.00	12.13

TABLE IV: Angle in degrees between ground truth and estimated normal vectors with $\mathcal{N}(0,0.2)$ disparity noise. (Scene shown on Fig. 7)

version is much slower than the convolutional method. Using star fill with 8 directions and maximum 10 steps takes about thrice the amount of time it takes to run the convolutional method with a 9x9 kernel, while processing a similar number of pixels. Despite being slower, star fill came close to the performance of 9x9 PCA in this case, with better accuracy.

Most of the errors come from areas close to edges. There are two reasons for this. There are cases when the stopping condition is not triggered. This occurs mostly around edges where the depth is exactly C^0 continuous. In other cases, the stopping condition is rightly triggered, reducing the number of included points, but there are not enough remaining points for an accurate estimation. This is usually observable around strong discontinuities, such as around the edges of the furthest cube in the center of figure 7b.

Additional Real-World Evaluation.

We have also collected data using the car, called ELTECar, of our university. Two cameras are mounted on the top, for which the baseline is about one meter. The reconstruction

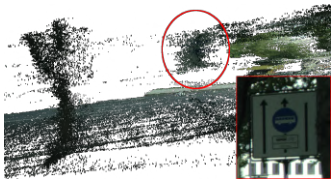


Fig. 9: Side view of pointcloud generated from disparities (Cityscapes dataset). The points of the sign in the bottom right corner are circled in the center. As errors affect the depth of the points more, the pointcloud is distorted here with a very high signal to noise ratio.

Fig. 10: Real-world examples with estimated normals from the Cityscapes [8] (left and center), and the Middlebury [32] dataset (right).

setup is visualized in figure 12. As the fixation of the cameras is not perfect, a slight rectification is required. We use chessboard images for this purpose.

Some images of real world results are provided in figure 11. In addition, we conducted experiments on our own stereo car dataset, collected using a stereo camera rig, the images are not fully rectified as the optical axes of the cameras are only quasi parallel. The camera calibration parameters (intrinsic and extrinsic) were first estimated from chessboard images, and the stereo pairs were rectified. Disparity maps were then generated using BGNet [33], and the adaptive method was applied to compute the surface normals. The corresponding 3D coordinates are calculated by the widely-used triangulation algorithm for standard/rectified stereo [34].

The estimated normals demonstrate that the method can be applied reliably to real driving scenarios, even when the disparity is obtained from a deep stereo network. This highlights both the versatility of our approach and its applicability beyond synthetic or benchmark datasets.

Remark that more results are included in the supplementary video.

V. CONCLUSIONS

It is shown here how affine correspondences provide an effective way of estimating surface normals from rectified stereo images. GPU implementations guarantee efficient and rapid real-time estimation. Our solution is purely geometric, without any machine learning-based component.

For fixed sized and shaped kernels, the proposed method outperforms PCA-based normal estimation both in terms of runtime and accuracy. The experiments show double speed-up w.r.t. an optimized PCA implementation while offering about $30\% - 40\%$ improvement over the average angular error. Moreover, we introduced heuristic methods for adapting our affine estimator to dynamic neighbour selection, yielding $20 - 25\%$ improvement in accuracy.

Distant camera facing surfaces are challenging for normal estimation as these scenarios exhibit a very high signal to noise ratio. Affine methods are more efficient in filtering the noise in such extreme conditions.

Future work. While the proposed normal estimation framework is computationally efficient, its real-time integration is currently bottlenecked by the disparity estimation stage, which accounts for the bulk of the processing latency. Furthermore, because the framework's spatial mapping relies on the fidelity of input disparity data, any artifacts from occlusions or reflective surfaces directly impact the final output

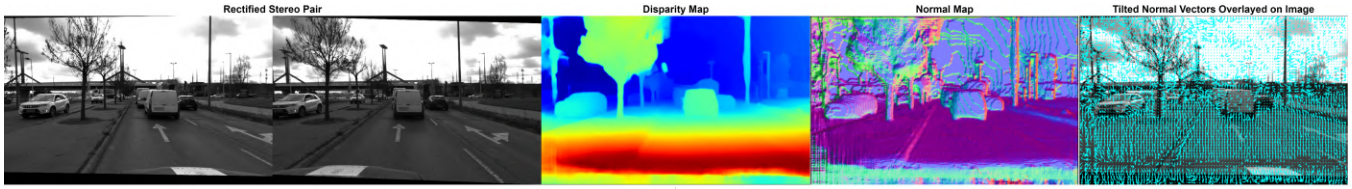


Fig. 11: Real-world result for ELTECar. Two images in the left shows the rectified images. It is clearly seen that rectification is mandatory. In the middle, coloured disparity image is pictured. The full normal map and regularly sampled normal directions are plotted in the right.

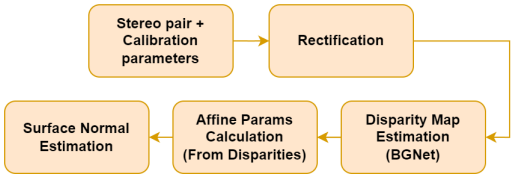


Fig. 12: Reconstruction pipeline for surface normal estimation. The point clouds can also be reconstructed from disparity values, but this process is not visualized.

quality. Consequently, future work will focus on accelerating and enhancing state-of-the-art disparity estimators to ensure both the speed and accuracy required for a fully realized real-time pipeline.

REFERENCES

[1] C. Raposo and J. P. Barreto, "Accurate reconstruction of oriented 3d points using affine correspondences," in *ECCV*, vol. 12373. Springer, 2020, pp. 545–560.

[2] D. DeTone, T. Malisiewicz, and A. Rabinovich, "Toward geometric deep SLAM," *CoRR*, 2017.

[3] V. Panek, Z. Kukelova, and T. Sattler, "Meshloc: Mesh-based visual localization," in *ECCV*, 2022.

[4] D. Nistér, O. Naroditsky, and J. R. Bergen, "Visual odometry for ground vehicle applications," *J. Field Robotics*, vol. 23, no. 1, 2006.

[5] H. Isack and Y. Boykov, "Energy-based geometric multi-model fitting," *International Journal of Computer Vision*, vol. 97, no. 2, pp. 123–147, 2012.

[6] D. Barath and J. Matas, "Progressive-x: Efficient, anytime, multi-model fitting algorithm," in *ICCV*, 2019, pp. 3780–3788.

[7] L. Hajder and I. Eichhardt, "Computer vision meets geometric modeling: Multi-view reconstruction of surface points and normals using affine correspondences," in *ICCV Workshops*, 2017, pp. 2427–2435.

[8] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *CVPR*, 2016.

[9] J. L. Schonberger and J.-M. Frahm, "Structure-from-motion revisited," in *CVPR*, 2016, pp. 4104–4113.

[10] R. Chen, S. Han, J. Xu, and H. Su, "Point-based multi-view stereo network," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 1538–1547.

[11] I. T. Jolliffe and J. Cadima, "Principal component analysis: a review and recent developments," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 374, 2016.

[12] N. Ming, Y. Feng, and R. Fan, "Sda-sne: Spatial discontinuity-aware surface normal estimation via multi-directional dynamic programming," in *2022 International Conference on 3D Vision (3DV)*. IEEE, 2022, pp. 486–494.

[13] X. Jiao and D. Wang, "Reconstructing high-order surfaces for meshing," *Eng. Comput.*, vol. 28, no. 4, pp. 361–373, 2012.

[14] K. Köser and R. Koch, "Differential Spatial Resection - Pose Estimation Using a Single Local Image Feature," in *ECCV*, 2008, pp. 312–325.

[15] J. Molnár and D. Chetverikov, "Quadratic transformation for planar mapping of implicit surfaces," *J. Math. Imaging Vis.*, vol. 48, no. 1, pp. 176–184, 2014.

[16] D. Barath and L. Hajder, "A theory of point-wise homography estimation," *Pattern Recognit. Lett.*, vol. 94, pp. 7–14, 2017.

[17] J. Molnár, R. Huang, and Z. Kato, "3d reconstruction of planar surface patches: A direct solution," in *Computer Vision - ACCV 2014 Workshops - Singapore, Singapore, November 1-2, 2014, Revised Selected Papers, Part I*, ser. Lecture Notes in Computer Science, C. V. Jawahar and S. Shan, Eds., vol. 9008. Springer, 2014, pp. 286–300.

[18] D. Barath, J. Molnar, and L. Hajder, "Novel methods for estimating surface normals from affine transformations," in *VISIGRAPP, Selected and Revised Papers*. Springer International Publishing, 2015, pp. 316–337.

[19] N. L. Minh and L. Hajder, "Affine transformation from fundamental matrix and two directions," in *VISAPP*, 2020, pp. 819–826.

[20] L. Hajder, L. Lóczi, and D. Barath, "Fast globally optimal surface normal estimation from an affine correspondence," in *ICCV*, 2023, pp. 3390–3401.

[21] D. Eigen and R. Fergus, "Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture," in *ICCV*, 2015, pp. 2650–2658.

[22] H. Zhan, C. S. Weerasekera, R. Garg, and I. D. Reid, "Self-supervised learning for single view depth and surface normal estimation," in *ICRA*. IEEE, 2019, pp. 4811–4817.

[23] R. I. Hartley, "Theory and practice of projective rectification," *Int. J. Comput. Vis.*, vol. 35, no. 2, pp. 115–127, 1999.

[24] Z. Megyesi, G. Kos, and D. Chetverikov, "Dense 3D Reconstruction from Images by Normal Aided Matching," *Machine Graphics & Vision*, vol. 15, pp. 3–28, 2006.

[25] R. A. Hamzah, H. Ibrahim, *et al.*, "Literature survey on stereo vision disparity map algorithms," *Journal of Sensors*, vol. 2016, 2016.

[26] E. O. Brigham, *The fast Fourier transform and its applications*. Prentice-Hall, Inc., 1988.

[27] K. Moreland and E. Angel, "The fft on a gpu," in *EUROGRAPHICS*, 2003, pp. 112–119.

[28] Yi Feng, Bohuan Xue, Ming Liu, Qijun Chen, and Rui Fan, "D2NT: A High-Performing Depth-to-Normal Translator," in *ICRA*.

[29] R. Fan, H. Wang, B. Xue, H. Huang, Y. Wang, M. Liu, and I. Pitas, "Three-filters-to-normal: An accurate and ultrafast surface normal estimator," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5405–5412, 2021.

[30] ryming2001, "Ryming2001/sda-sne: Sda-sne: Spatial discontinuity-aware surface normal estimation via multi-directional dynamic programming." [Online]. Available: https://github.com/ryming2001/SDA-SNE

[31] S. D. Roth, "Ray casting for modeling solids," *Computer graphics and image processing*, vol. 18, no. 2, pp. 109–144, 1982.

[32] D. Scharstein, H. Hirschmüller, Y. Kitajima, G. Krathwohl, N. Nešić, X. Wang, and P. Westling, "High-resolution stereo datasets with subpixel-accurate ground truth," in *GCPR, Münster, Germany, September 2-5, 2014, Proceedings 36*. Springer, 2014, pp. 31–42.

[33] B. Xu, Y. Xu, X. Yang, W. Jia, and Y. Guo, "Bilateral grid learning for stereo matching networks," 2021. [Online]. Available: https://arxiv.org/abs/2101.01601

[34] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003.