DCSH - Matching Patches in RGBD Images

Yaron Eshet
Tel-Aviv University

Simon Korman
Tel-Aviv University

Eyal Ofek
Microsoft Research

Shai Avidan
Tel-Aviv University

Abstract

We extend patch based methods to work on patches in 3D space. We start with Coherency Sensitive Hashing [12] (CSH), which is an algorithm for matching patches between two RGB images, and extend it to work with RGBD images. This is done by warping all 3D patches to a common virtual plane in which CSH is performed. To avoid noise due to warping of patches of various normals and depths, we estimate a group of dominant planes and compute CSH on each plane separately, before merging the matching patches. The result is DCSH - an algorithm that matches world (3D) patches in order to guide the search for image plane matches. An independent contribution is an extension of CSH, which we term Social-CSH. It allows a major speedup of the k nearest neighbor (kNN) version of CSH - its runtime growing linearly, rather than quadratically, in k. Social-CSH is used as a subcomponent of DCSH when many NNs are required, as in the case of image denoising. We show the benefits of using depth information to image reconstruction and image denoising, demonstrated on several RGBD images.

1. Introduction

Patch based methods rely on the observation that local image patches occur frequently within an image. This observation led to great progress in various applications such as texture synthesis and image denoising.

Virtually all patch based methods use square patches and measure similarity between patches using the Sum-of-Squared-Distances (SSD), no doubt for computational efficiency. But these image patches are the deformed projections of patches in 3D. We therefore propose to use patches in 3D space, in order to increase the quantity and the quality of similar patches. In particular we propose to extend patch based methods to work on RGBD images.

Clearly, patch matching in 3D induces patch matches in the 2D image plane, which are defined by homographies (projective transformations). This richer search space defines a better pool of potential matches, but requires a highly efficient search scheme. As a core technology we develop DCSH, a method for matching patches between two RGBD images that extends Coherency Sensitive Hashing [12]. CSH is a method for computing an Approximate Nearest Neighbor Field between two RGB images, or between an image and itself. Specifically, given an RGBD image we use the depth values to compute the depth and normal of every patch and warp the patches to some virtual reference plane. CSH is carried out in this virtual plane. Experiments show that this depth information considerably improves the quality of patch matching.

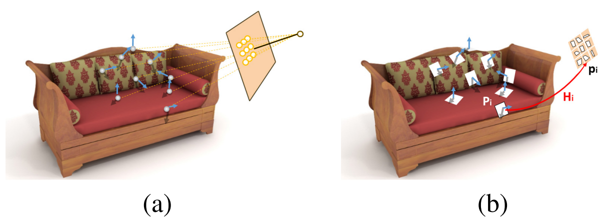


Figure 1. **DCSH:** (a) Each image pixel represents a point in 3D space and a normal direction according to the depth map. (b) Each world patch (P_i) is projected to some quadrilateral (p_i) on the image plane, by some homography (H_i). The fact that world patches are repetitive in the scene is used to guide the search of similar (projected) patches in the image plane.

Still, a single virtual plane might introduce strong warping and resampling artifacts that will affect matching, especially for patches with orientation that is perpendicular to that of the virtual plane. DCSH minimizes these artifacts by estimating several dominant planes in the 3D space and repeating CSH on each plane separately. Matching results are then merged and sorted to produce a list of unique matching patches.

So far we focused on the ability to retrieve the best match but in practice one is often interested in retrieving the k best matches. Unfortunately, the running time of CSH grows quadratically with k. To mitigate that, we propose Social-CSH, whose runtime grows only linearly with k. Social-CSH finds only a small number of matches for each patch and enriches the list of matching candidates by incorporating their own candidate matches. We observe a considerable speedup of about $\times 20$, accompanied with an *improvement* in accuracy, compared to standard CSH. An accuracy/efficiency tradeoff can be controlled, allowing for instance, a $\times 40$ speed up at the cost of a slight degradation in accuracy.

We show the advantages of our contributions for image reconstruction, which is a common building block in texture synthesis algorithms as well as for image denoising. Obtaining the RGBD images is not the focus of our work and we rather aimed at getting the highest quality ground data. To do so, we used high quality color images, aligned using a multi-view stereo algorithm, resulting in a single RGBD image with reliable depth and high quality RGB components[1]. We expect that the rise of range sensors, such as Kinect, will increase the availability of high quality RGBD images in the future.

2. Background

At their core, patch-based methods require efficient Approximate Nearest Neighbor (ANN) algorithms to find similar patches to each query patch. Traditional approaches rely on ANN methods such as KD-trees [3] or LSH [2]. Alternatively, one might consider more recent algorithms such as Barnes *et al's.* PatchMatch [4], which is an extremely efficient algorithm that works by randomly finding possible patch candidates and propagating good matches across the image plane. Korman and Avidan [12] proposed Coherency Sensitive Hashing (CSH) that combines the strengths of LSH and PatchMatch.

Such algorithms consider matching patches under 2D translations only, though some recent works have considered wider classes of transformations. The higher dimensional search space has a significant impact on the algorithms' runtime to approximation-accuracy tradeoff. One example, is the Generalized PatchMatch algorithm [5] that extends [4] to consider also rotations and uniform scales. Such a generalized Nearest Neighbor Field (NNF) has been later shown to be useful in image enhancement applications [10].

Nevertheless, these additional degrees of freedom (e.g. scale, rotation) do not suffice to capture the repetitive nature of patches in the 3D world. An alternative method could search for matches on a dense SIFT [14] field, and since SIFT is a stable descriptor under affine transformations this can be seen as a proxy to patch matching in 3D space. However, RGBD images contain depth information, so we can model the 3D geometry explicitly and obtain better matching results.

Image reconstruction has become a customary application for evaluating the quality of NNFs. Given only a target image B and an NNF from a source image A to B, the goal is to reconstruct image A using the patches of B. This is a standard building block in many patch based methods for image enhancement, such as denoising, super-resolution and retargetting. The most recent state-of-the-art ANN matching algorithms, TreeCann [15] and Propagation-Assisted KD-Trees [11], evaluate their reconstruction capabilities, both visually and in terms of RMS error.

Image denoising made tremendous progress in the last decade using various different techniques. We consider patch-based methods such as Non-Local-Means [7] and its many extensions, as reviewed in Buades *et al.* [8], including the popular BM3D algorithm [9] that produces state-of-the-art results. These algorithms work on a single noisy image, and recent theoretical analysis [13] suggests we are approaching the limits of patch-based techniques.

One way to improve results is to add information and Zhang *et al.* [17] proposed a Multi-View Image Denoising algorithm where the goal is to collect similar patches across multiple views with independent noise, where depth is treated as a latent variable to be estimated from the data. In recent years, active IR sensors such as Kinect and other Time of Flight sensors have become wide spread. They are today an important source for RGBD images. With RGBD images there are no multiple views available to aid the denoising process, but we show that single image denoising can still benefit from the use of depth information.

There is also some research on denoising Kinect images, where the goal is to denoise the depth map produced by the sensor. For example, Park *et al.* [16] produce a high quality depth map by upsampling the original depth map using the high quality RGB image. We deal with a different setting of denoising the RGB component of an RGBD image, using depth as a cue.

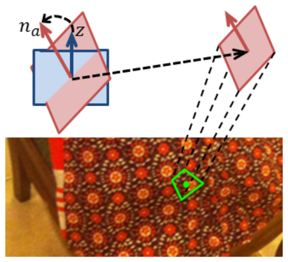
3. DCSH (Depth CSH) matching

DCSH extends Coherency Sensitive Hashing (CSH) [12] to work with RGBD images. The general idea is to use the better matchings that occur between real-world patches in order to find correspondences between their projected image-patches. This can be done by mapping the area around each image pixel by a very particular warp, simulating a camera scanning the surface at a fixed distance from a fixed direction. In Section 3.1 we will first be interested in a particular case, where each surface location will be normalized as if it was viewed from the direction of the surface normal, at a fixed depth. In Section 3.2 we present a more general patch normalization scheme, which will be used in the final DCSH matching algorithm, which is described in Algorithm 1.

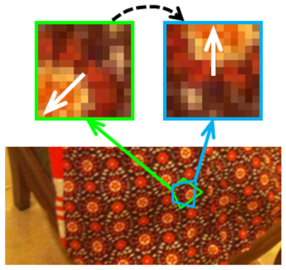
3.1. Simulating a (per pixel) fronto-parallel view

Given the 3D world coordinates $r_a = (X_a, Y_a, Z_a)$ associated with each image pixel a, we first use a standard robust estimation of the normal direction n_a at the 3D point. That is, we take n_a to be the least-squares solution to the stack of 49 equations of the form $r_b \cdot n_a = 1$ for each pixel b in the 7×7 neighborhood of a.

[1]Figure 5 shows a sample of the RGBD images we use in the paper.



(a)



(b)

Figure 2. **Simulating a (per pixel) fronto-parallel view: (a)** For each (green) pixel in the image, we simulate its local appearance, as if the surface was captured from a fronto-parallel view at a distance of z_{ref}. For doing so, consider an origin based world-patch (blue, on left), which faces the z axis. We rotate it in 3D to face the associated normal n_a, translate it to the associated 3D location r_a, and finally - project it to a quadrilateral on the image plane. The combined transformation defines a homography h_a. **(b)** The inverse homography h_a^{-1} can be used to sample a normalized patch (green) around the pixel. We then (in-plane) rotate the normalized patch (by a rotation matrix R_a) such that it faces its dominant RGB texture orientation (white arrow). A new normalized patch can be sampled using $R_a^{-1} \cdot h_a^{-1}$ (further details in text).

Once we know the 3D location r_a and normal n_a, we turn to compute the homography H_a that will enable sampling the surface at r_a from the direction of n_a at a fixed distance of z_{ref}. The main steps of the homography construction are illustrated in Figure 2(a). A world-patch, located at the origin and facing direction of $z = (0, 0, 1)$, is first rotated in 3D so that its normal coincides with the surface normal direction n_a. This is done using a rotation matrix $R = I + [\hat{w}]_\times \sin\theta + (1 - \cos\theta)[\hat{w}]^2_\times$, where $w = \hat{n} \times \hat{z}$, $\theta = \cos^{-1}(\hat{n} \cdot \hat{z})$ [2] and $[\cdot]_\times$ denotes the skew symmetric matrix of a vector. Then, it is shifted from the origin to $r_a = (X_a, Y_a, Z_a)$ and finally - projected to a quadrilateral on the image plane, using the intrinsic matrix K. The combined transformation h_a can be formalized by:

$$h_a = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} R^a_{11} & R^a_{12} & X_a \\ R^a_{21} & R^a_{22} & Y_a \\ R^a_{31} & R^a_{23} & Z_a \end{bmatrix} \cdot \begin{bmatrix} 1/f & 0 & 0 \\ 0 & 1/f & 0 \\ 0 & 0 & 1/z_{ref} \end{bmatrix} \quad (1)$$

One remaining degree of freedom of the warp is the surface in-plane rotation (perpendicular to the normal) which determines the orientation of the normalized patch. We use the orientation normalization technique of SIFT [14], where a prominent orientation is chosen according to a weighted voting scheme, based on orientations and magnitudes of grayscale intensity gradients in a neighborhood of the central pixel. The recovered in-plane rotation R_a is used to produce the final homography $H_a = h_a \cdot R_a$.

3.2. Simulating (several) general views

In the previous section we normalized each pixel location to a patch, representing a canonical fronto-parallel view. Since pixel resolution in the image is limited, the normalization process for areas in the surface whose natural camera viewpoint is very different from being both fronto-parallel and at depth z_{ref}, will introduce warping and resampling errors that affect the quality of the matching. For example, areas that are close to the camera (with depth smaller than z_{ref}) will be compared based on their normalized versions, which lack relevant information, e.g., due to downsampling. In general, any possible reference plane will work well for areas that undergo only a mild deformation under the normalization. The normalization to different viewpoints results in a rich variety of candidate patches which will enable improved patch matching.

[2]For any vector v, $\hat{v}$ is the its unit normalized version.

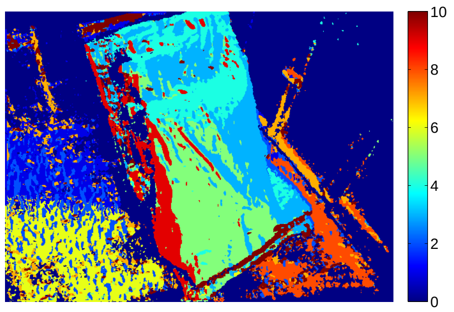


Figure 3. **Example of DCSH viewpoint clustering.** The Dress image clustered to 10 prototypical viewpoints, color-coded from 1 to 10 (Areas with invalid depth are coded with 0, dark blue).

The gold standard, in this sense, would be to normalize all of the image patches to every single viewing point of each of the image's patches (leaving the target image patch unchanged under the normalization). This of course is infeasible, and we therefore compromise between speed and accuracy by selecting a set of L prototypical viewpoints, which are found by a clustering process on the surface normals and depths (these determine the natural viewpoint). Specifically, each pixel is represented by a 5D vector $[x, y, z, n_x, n_y]$, where $[x, y, z]$ are its associated 3D coordinates and $[n_x, n_y]$ are the x and y components of the normal at the $3D$ point. We normalize all values to the range $[0, 1]$, except for x, y that are normalized to the range $[0, 0.5]$. This serves to encourage nearby pixels in the image plane to behave similarly. We then run k-means in this 5D space and each cluster center induces a different reference plane.

Each representative cluster center $[x^i, y^i, z^i, n^i_x, n^i_y]$ represents a specific prototypical viewpoint and induces a homography H_i (following the exact formulation in the Section 3.1). See Figure 3 for an example of clustering an image to L=10 areas with prototypical viewpoints. The normalization of any patch a to the i'th viewpoint can now be synthesized by resampling through the concatenated homography: $N_a = H_i \cdot H_a^{-1}$.

3.3. Nearest neighbor search

The procedure above produces a (square) normalized patch for each location in images A and B. The CSH al-

Algorithm 1 ***DCSH: Matching Patches in RGBD Images***

```
Input: RGBD images $A$ and $B$, intrinsic matrix $K$
Output: $k$ patch mappings (homographies) per pixel $a \in A$

Step 1: Homographies to a fronto-parallel plane $z_{ref}$
  1. For each pixel $a \in A$ (and $b \in B$):
     (a) Estimate the normal $n_a$, using LS (see Sec. 3.1).
     (b) Compute the homography $h_a$ (Eqn. 1).
     (c) Find the (orientation normalizing) in-plane rotation $R_a$ (see Sec. 3.1).
     (d) Compute the final homography: $H_a = h_a \cdot R_a$.
Step 2: Prototypical Viewpoint Fitting
  1. Using K-means (see Sec. 3.2) - Fit $L$ homographies $\{H_i\}_{i=1}^{L}$ (each induced by a prototypical viewpoint).
Step 3: Nearest Neighbor Search
  1. For each prototypical homography $H_i$:
     (a) Create a normalized patch per location $a \in A$ (and $b \in B$) by sampling: $N_a = H_i \cdot H_a^{-1} \cdot p_a$ .
     (b) Run a kNN CSH search between the arrays of normalized patches: $\{N_a\}_{a \in A}$ and $\{N_b\}_{b \in B}$.
     (c) For each match $N_a \Rightarrow N_b$ ($k$ per location), compute the direct mapping $H_{ab} =: H_a^{-1} \cdot H_b$ and the associated error $\|H_{ab} \cdot p_a - p_b\|_2$ (see Sec. 3.3).
  2. Return the $k$ lowest-error transformations per $a \in A$ (out of the $kL$ possible candidates).
```

gorithm [12] is then naturally used to match these normalized patches, but it requires two simple adaptations. While it usually works with the entire set of overlapping square patches of an image, here, one patch per location is given to the algorithm, but these patches do not overlap in the regular sense. This fact required the preprocessing stage of Walsh-Hadamard-Kernel patch projections to be computed directly (rather than using the more advanced Grey-Code Kernel method [6], which requires true overlapping). In addition, since neighboring patches underwent different homographies and orientation corrections, patch matches in the image plane are propagated in all four directions (up/down/left/right) rather than in the single expected direction.

Once matches have been found, we no longer need the 'bridging' normalized patches and we turn back to the original image patches and construct direct mappings between them, avoiding excess interpolation and resampling. Namely, if the normalized version $H_b^{-1}(p_b)$ of the patch $p_b \in B$ was matched to $H_a^{-1}(p_a)$ (a normalized version of a patch $p_a \in A$), we directly link the corresponding locations using the combined homography: $H_{ab} =: H_a^{-1} \cdot H_b$.

4. Social-CSH

The main computational effort of the DCSH algorithm is spent on its NN search stage. The burden of this stage becomes even worse, when we use many simulated viewpoints or are required to find a large number, k, of NNs. Although CSH is a fast algorithm for approximate k-ANN Field estimation, it inherently scales *quadratically* with k. Simply put, this happens since each patch evaluates the k NNs of its k NNs. This deems the algorithm impractical in cases where k is large (e.g. in the order of hundreds). In order to overcome this obstacle, we introduce Social-CSH, which is a standalone improvement of kNN CSH, independent of the DCSH framework.

Social-CSH runs standard CSH as a subroutine, and it is based on sharing matches across NNs. Social-CSH is configured by two parameters - k (the desired number of NNs) and $\tilde{k}$ (the number of NNs to be computed by the CSH subroutine). The process is as follows: standard CSH is used to get $\tilde{k}$ seed NNs per patch. This core list is extended by taking the NNs of NNs, forming a merged list of $\tilde{k}^2 + \tilde{k}$ NNs. The merged list is scanned to remove duplicates and sorted by RMSE to take the best k candidates as a final output.

We chose $\tilde{k}$ to be $2\sqrt{k}$, as it results in an expanded list of $\sim 4k$ patches, which is a large enough set from which a high quality set of k distinct NNs may be obtained. As a result of this selection, Social-CSH grows *linearly*, rather than quadratically, in k. Our experiments validate this speed up (being 10 to 50 times faster for k in the range of 100 and 200) and show that Social-CSH typically improves on accuracy as well. In addition, we will demonstrate how the choice of $\tilde{k}$ trades-off between speed and accuracy.

4.1. Social-CSH experiments

We evaluated Social-CSH on three RGB images (Dress, Cup and Friends) from our RGBD image collection, ignoring the depth component. In the first setup, we seek to get $k = 100$ nearest neighbors (NNs). we run the standard CSH kNN with $k = 100$ and compare it to Social-CSH with $\tilde{k} = 20$ (which is $2\sqrt{k}$, our default choice). In the second and third setups, we seek to get $k = 200$ NNs. we run the standard CSH kNN with $k = 200$ and compare it to Social -CSH with 2 different choices of $\tilde{k} = 30$ and $\tilde{k} = 20$ (above and below $2\sqrt{k}$, respectively), which result in the choice of the best 200 out of 900 or 400 NNs. These two configurations of Social-CSH will demonstrate its speed-accuracy tradeoff.

The results are shown in the top (RMSE gain in mapping) and bottom (runtime speedups) tables of Figure 4.

Considering first the comparisons for the first setup ($k = 100$), which appears in the left side of both tables, we can see an average improvement in RMSE of 2.59% along with a major average speedup factor of 13.2.

In the right hand side of each of the tables, where $k = 200$, the results in the parentheses are those of the second configuration (with $\tilde{k} = 20$) of Social-CSH. Comparing CSH to the first configuration of Social-CSH (using $\tilde{k} = 30$) the average gain of 3% in RMSE is similar to that of the previous setup, however, as expected - the average runtime speedup jumps to 22.6. The second configuration (with $\tilde{k} = 20$) gives a boost in speed to a factor of 41.2, but results in accuracy loss of -1.2%. This clearly shows the trade-off between speed and accuracy, which can be controlled through the $\tilde{k}$ parameter of Social-CSH.

	$k = 100$			$k = 200$		
image	**CSH**	**S-CSH**	**gain**	**CSH**	**S-CSH**	**gain**
Dress	35.3	34.4	2.6%	35.6	34.7 (35.9)	2.5% (-0.8%)
Cup	36.8	36.0	2.1%	36.9	36.2 (37.3)	1.8% (-1.3%)
Friends	40.7	39.4	3.0%	40.9	39.9 (41.5)	2.5% (-1.5%)
avg.:	37.6	36.6	**2.6%**	37.8	36.9 (38.2)	**2.3% (-1.2%)**

	$k = 100$			$k = 200$		
image	**CSH**	**S-CSH**	**speedup**	**CSH**	**S-CSH**	**speedup**
Dress	927	69.6	13.3	2986	133 (69.6)	22.5 (43.0)
Cup	894	68.1	13.1	2988	133 (68.1)	22.7 (42.6)
Friends	1021	77.4	13.2	2899	128 (77.4)	22.5 (38.3)
avg.:	947	71.7	**13.2**	2959	131 (71.7)	**22.6 (41.3)**

Figure 4. **kNN performance: CSH versus Social-CSH. Top:** RMSEs (in graylevels) and gain factors. **Bottom:** Runtimes (seconds) and speedup factors. See text for full details.

5. DCSH Applications

The *iPhone* data-set The iPhone data-set consists of 8 RGBD images[3], five of which can be seen in rows 1 (RGB), 2 (depth) and 3 (estimated normals) of Figure 5. The images include a variety of indoor and outdoor scenes, taken under different conditions, where most are characterized by large amounts of detailed texture, which appear across different depths or surfaces in the scene.

5.1. Image Reconstruction

In this experiment, we demonstrate the added value of using real-world patches, as opposed to standard square image patches, in the clean process of image reconstruction. In this process, given only a target image B and an NNF from a source image A to B, it is required to reconstruct image A using only the patches of B.

We use Coherency-Sensitive-Hashing (CSH) [12] as the NNF search engine in our reconstruction alternatives. The baseline method computes an ordinary NNF, using CSH,

[3]The full set of 8 images appears in the project webpage [1].

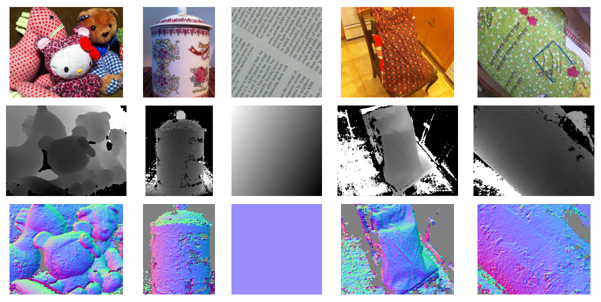


Figure 5. **Five representative images of the iPhone dataset.** Rows 1-2: input RGBD images. Row 3: Estimated normals maps, where gray areas are invalid due to noisy or missing depth values.

between square 8×8 image patches. In the reconstruction process, each patch is replaced by its nearest neighbor patch and since the NNF is dense, each final pixel will be an average of the 64 pixel values it receives through the 64 patches that contain it. A standard improvement (following Non-Local-Means (NLM) [7]) is achieved by introducing a spatial gaussian weight kernel G (with a spatial σ of 5) over each NN patch, giving higher weight to pixels closer to the center of the patch. The final result is a weighted average, per pixel, of 64 color samples. We term this baseline method CSH-NLM. On the other hand, replacing CSH by DCSH, we obtain an equivalent reconstruction pipeline, DCSH-NLM. A delicate adjustment is needed, regarding the NLM gaussian weighting scheme. This is due to the fact that the matches were retrieved in a normalized plane, so the gaussian weighing should be done in the normalized plane itself, rather than on the image plane. We therefore use at each patch p centered at pixel a, the same Gaussian weight kernel G after warping it back to image A, using the relevant inverse homography H_a, namely, $H_a^{-1} \cdot G$.

In addition, we were interested in the contribution of the orientation normalization step (see step 1(c) in Algorithm 1). To that end, we run two versions - DCSH-NLM-Orient and DCSH-NLM, with and without the orientation step. Some examples of running both methods (baseline CSH-NLM vs. DCSH-NLM with/without color texture orientation normalization) on several image pairs from the iPhone data-set appear in detail (focusing on specific enlarged areas) in Figure 6. Also, in Table 1, we compare the performance of the 3 methods on the full images.

method/image	Dress	Plane1	Plane2	Building	Rocks
CSH-NLM	13.31	10.74	4.8	10.32	9.76
DCSH-NLM	8.03	7.86	2.09	8.81	6.99
DCSH-NLM-Orient	6.86	7.52	1.68	6.65	6.26

Table 1. **RMSEs** (in graylevels) **of reconstructions.** We compare DCSH (with and without orientation normalization) to the baseline CSH. The introduction of patch normalization reduces the reconstruction error dramatically, while the addition of orientation normalization (as done in SIFT) gives an additional improvement.

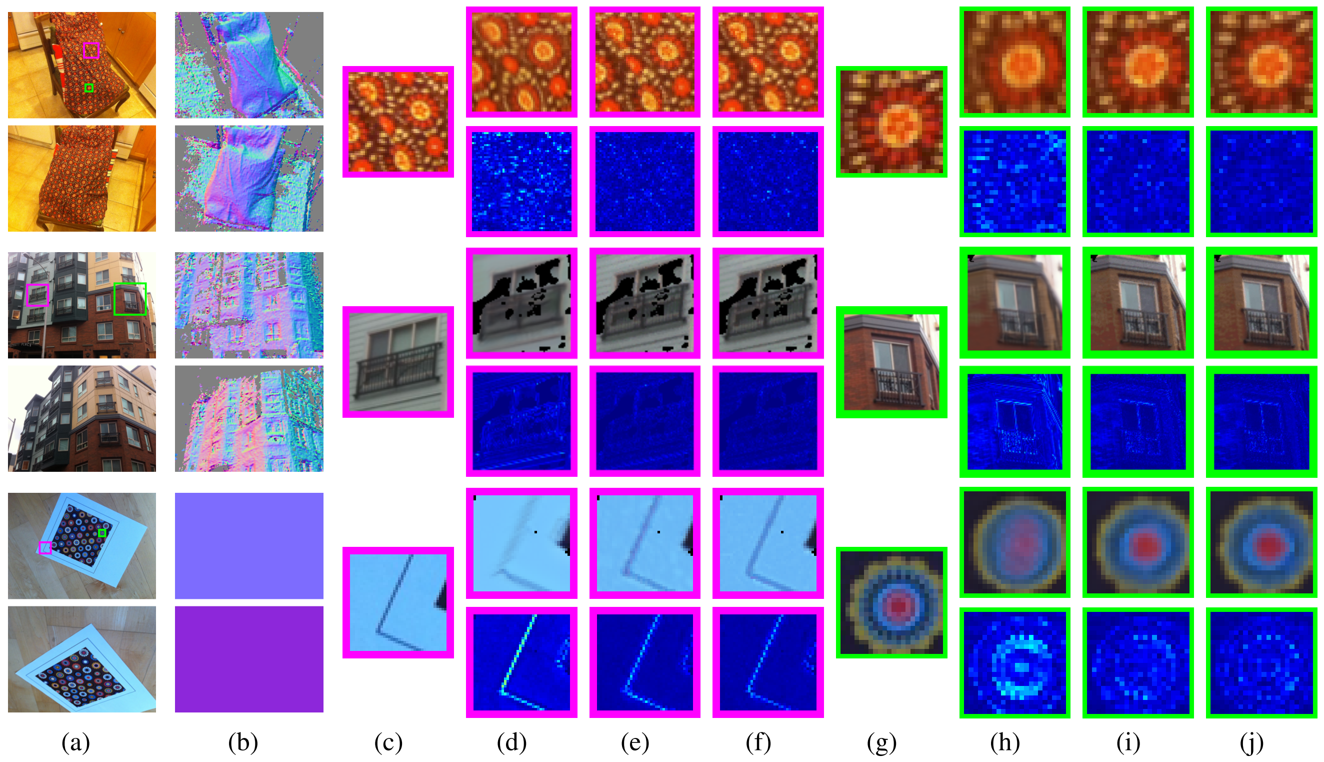


Figure 6. **Three reconstruction examples.** For *each* example: (a) RGB images - source (top), target (bottom) (b) normal maps (c) original subimage, (d) CSH-NLM reconstruction (top) and RMSE (bottom), (e) DCSH-NLM reconstruction (top) and RMSE (bottom), (f) DCSH-NLM-Orient reconstruction (top) and RMSE (bottom), (g)-(j): another subimage reconstruction example.

5.2. RGBD image denoising

We choose to demonstrate the added value of using world 3D patch matches through a simple denoising pipeline. It first finds, for each patch, a set of similar patches. These patches are then used to construct a low-rank PCA space, and denoising the original patch amounts to projecting it on the subspace and storing the denoised patch back in the image. Since each pixel is covered by multiple patches, averaging is used to obtain the denoised image. And optionally - a Bilateral Filter is applied as post processing step. We give further details on each of the steps.

Step 1: Find K NNs per patch In this stage we find k=200 NNs per image patch. In the first alternative, we compute the 200 NNs directly with CSH and in the second - we run both CSH and DCSH to find 200 NNs each and choose the best 200 (with lowest SSD errors) of the two sets of patches. The reason we use the combination is that the 2D CSH patches have the advantage of not undergoing any kind of warping.[4]

Step 2: Denoise each image patch using PCA Formally, let p be the query RGB patch (represented as a flat vector of length L) and let $\mathcal{P} = \{p^i\}_{i=1}^m$ be the set of m matching patches found in step 1. We then create the matrix $A = [p-u, p_1-u, \cdots, p_m-u]$, where $u = \frac{1}{m+1}(p+\sum_{i=1}^m p^i)$ is the patch mean. We take an SVD decomposition of $A = UDV^T$, and project p on the eigenvectors of the top c eigenvalues to obtain the denoised patch p':

$$p' = u + UD_cV^T(p-u) \tag{2}$$

where D_c is a diagonal matrix (with A's singular values $\{\lambda_d\}_{d=1}^L$ on its diagonal) where the singular values beyond c are set to zero. We use the method of Zhang *et al.* [17] to automatically find the preferred dimensionality c, separately for each set of matching patches

Step 3: Integrate the denoised patches to form the denoised image Using 8×8 patches, 64 values are assigned to each pixel and are averaged to obtain the denoised image.

Step 4: Bilateral filtering This is an optional step, which isn't needed for comparing the denoising scheme with and without the usage of depth. It is intended to handle the smooth areas of the image, which aren't handled specifically by our pipeline. This is required in order to compare against dedicated algorithms, such as BM3D.

We experiment with three versions of this pipeline:

- **CSH-PCA:** Steps 1-3, where in step 1 we use 200 2D patches, found by CSH.
- **DCSH-PCA:** Steps 1-3, where in step 1 we use the best 200 patches out of: 200 2D patches (found by CSH) in addition to 200 3D patches found by DCSH.

[4]Both CSH and DCSH are run in their 'Social' version (see Section 4)

- **DCSH-PCA-BI:** Like DCSH-PCA, with the addition of Bilateral filtering (step 4)

5.2.1 RGBD image denoising experiments

In this section, we experiment with the 3 denoising versions. Throughout the experiments, we compare our method with the BM3D [9] algorithm, which is considered a state of the art method in the field. The input to our algorithm is an RGBD image with synthetic noise added to the RGB image, while assuming the depth image is left intact. In all experiments we create the noisy image, by adding white gaussian noise with a standard deviation of $\sigma = 25$ graylevels to each color channel independently.

It is a well known property of natural images, that the higher the textureness (e.g. mean gradient magnitude) of a patch - the lower the density of similar patches in the image (see e.g. [18]). One implication of this fact is that NNs methods tend to under-smooth low texture area, since they fit the noise rather than the signal. These areas dominate an image's area and therefore such methods need to specifically handle non-textured areas. To that end, we use the Bilateral Filter, which further smoothes these areas. Another implication is that textured patches typically have few good NNs throughout the image. For such patches, the introduction of normalized (3D) patches allows to increase the density of similar patches in the image, by searching across general homographies (image plane scales and orientations as well as out-of-image-plane rotations). We therefore expect the main impact of depth normalization to occur in textured areas of the image.

This assumption can be validated, when inspecting in which regions of the images our method outperforms BM3D. These regions are shown, for example, in Figure 7 (c) colored in green (BM3D is superior in red areas), and are evidently highly correlated with the textured areas of the image. An insight into the contribution of 3D patches to the denoising process can be observed by examining the statistics of the patches that manage to get into the final list of 200 patches fed to the PCA process. Over all images, an average of around 80% of the patches originated from the list of normalized (3D) patches. This can be seen visually in Figure 7 (b), where the per-patch ratio (in [0,1]) of 3D patches vs. regular image patches is color-coded [blue to red]. Namely, red areas are those where the vast majority of contributing patches came from the normalized list.

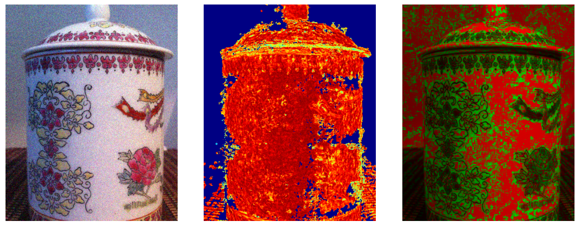

(a) (b) (c)

Figure 7. **A detailed example** (see text for discussion): (a) noisy image (b) fraction of normalized (as opposed to regular) patches automatically chosen by the algorithm (the range [0,1] color coded by [blue,red]), (c) 'winner' areas - DCSH (green) vs. BM3D (red).

in Table 2, we compare the different versions of our pipeline to that of BM3D. Comparing the first and second columns (CSH-PCA vs. DCSH-PCA) the contribution of adding depth normalization is evident across all examples and amounts to an average of $0.5dB$. In order to compare with BM3D, we added the post-processing bilateral filter smoothing (DCSH-PCA-BI). Here as well, our method improves on BM3D with a significant average gain of 0.5 dB[5].

image	CSH-PCA	DCSH-PCA	CSH-PCA-BI	BM3D
Paper	29.70	30.53	31.56	30.32
Friends	28.82	29.01	29.70	28.66
Dress	29.16	29.63	30.70	29.67
Cup	30.16	30.42	32.15	31.97
Book	28.71	29.35	30.09	29.28
Rocks	28.15	28.85	29.49	29.91
Tree	27.80	28.29	28.87	29.19
Mosaic	27.89	28.47	29.19	28.68
Avg.:	**28.80**	**29.32**	**30.22**	**29.71**

Table 2. **PSNR denoising results on iPhone data-set:** The usage of 3D patches results in significantly improved denoising (comparing CSH-PCA to DCSH-PCA) across all images with an average gain of 0.5 dB. Also, our method equipped with the final bilateral filter post-processing improves on BM3D across all images, with an average gain of 0.5 dB.

Figure 8 gives a close look at the qualities of our different denoising results (also compared to BM3D) on several images from our iPhone dataset. Across the different images, the contribution of (3D) patch normalization can be seen by comparing columns '2D' (CSH-PCA) and '3D' (DCSH-PCA) (notice especially the doll's ear, where fine details are revealed, or the cleaner letters in the text). Also, in comparison to BM3D, our full pipeline '3D-BI' better preserves fine details (e.g. in dress flowers) and introduces less hallucinated artifacts (e.g. around text letters).

6. Conclusions

We extended patch based methods to work on patches in 3D space. In particular, we extended the CSH patch matching algorithm to work with RGBD images. The novel algorithm, DCSH, runs CSH on a set of planes, representing prototypical viewpoints in the image. We also developed Social-CSH, an independent contribution, which improves kNN CSH to allow quick computations of ANN fields for large values of k (e.g. in the hundreds), at lower error rates.

We showed the added value of using depth information for improving the quality of patch matching and in partic-

[5]Note that our method uses depth information, which BM3D does not.

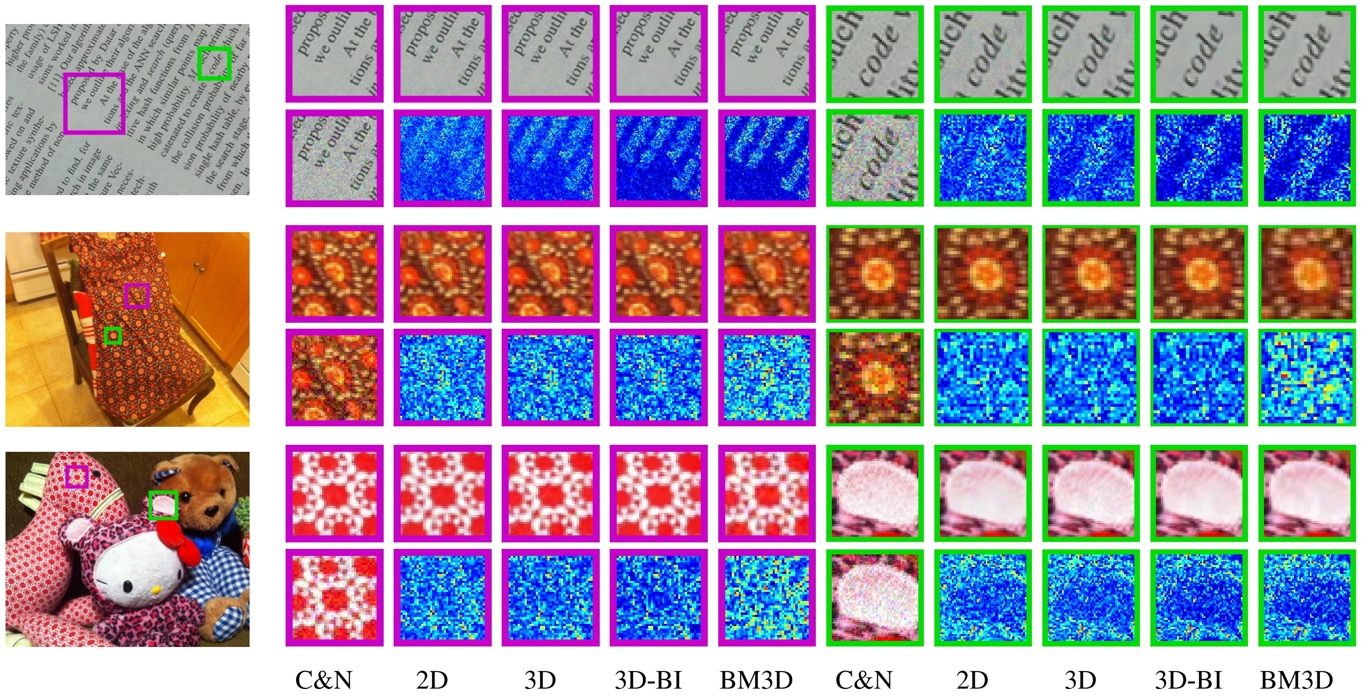


Figure 8. **iPhone dataset denoising examples.** ***Each* example contains:** the clean RGB image and 4 enlarged areas, each with 5 columns. **'C&N'**: Clean patch (top) and Noisy patch (bottom), **'2D'**: CSH-PCA denoised (top) and RMSE (bottom), and similarly for the others: **'3D'**: DCSH-PCA, **'3D-BI'**: DCSH-PCA-BI, **'BM3D'**: BM3D. Additional results appear in the project webpage [1].

ular, we experimented with DCSH in the context of image reconstruction and image denoising. The results point to depth as a new source of information in patch based methods and suggest that DCSH could prove useful in other applications, such as super-resolution, inpainting and retargetting. In addition, the algorithmic improvement used in Social-CSH can be applied to other existing 2D algorithms.

Acknowledgements This work was supported by the Israel Science Foundation (grant No. 873/08, in part) and the Ministry of Science and Technology.

References

[1] DCSH webpage. `www.eng.tau.ac.il/~simonk/DCSH`. 5, 8

[2] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *FOCS*, 2006. 2

[3] S. Arya, D. M. Mount, Na. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6), November 1998. 2

[4] C. Barnes, E. Shechtman, A. Finkelstein, and D. Goldman. Patchmatch: a randomized correspondence algorithm for structural image editing. *ACM TOG*, 28(3), 2009. 2

[5] C. Barnes, E. Shechtman, D. Goldman, and A. Finkelstein. The generalized patchmatch correspondence algorithm. *ECCV*, 2010. 2

[6] G. Ben-Artzi, H. Hel-Or, and Y. Hel-Or. The gray-code filter kernels. *In PAMI*, 2007. 4

[7] A. Buades, B. Coll, and J.M. Morel. A non-local algorithm for image denoising. In *CVPR*, 2005. 2, 5

[8] A. Buades, B. Coll, and J.M. Morel. Self-similarity-based image denoising. *Comm. of the ACM*, 54(5), 2011. 2

[9] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian. Image denoising by sparse 3-d transform-domain collaborative filtering. *Image Processing*, 16(8), 2007. 2, 7

[10] Y. HaCohen, E. Shechtman, D.B. Goldman, and D. Lischinski. Non-rigid dense correspondence with applications for image enhancement. *ACM TOG*, 30(4), 2011. 2

[11] K. He and J. Sun. Computing nearest-neighbor fields via propagation-assisted kd-trees. In *CVPR*, 2012. 2

[12] S. Korman and S. Avidan. Coherency sensitive hashing. In *ICCV*, 2011. 1, 2, 4, 5

[13] A. Levin, B. Nadler, F. Durand, and W. T. Freeman. Patch complexity, finite pixel correlations and optimal denoising. In *ECCV (5)*, 2012. 2

[14] D.G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2), 2004. 2, 3

[15] I. Olonetsky and S. Avidan. Treecann-kd tree coherence approximate nearest neighbor algorithm. *ECCV*, 2012. 2

[16] J. Park, H. Kim, Y. Tai, M. S. Brown, and I. Kweon. High quality depth map upsampling for 3d-tof cameras. In *ICCV*, 2011. 2

[17] L. Zhang, S. Vaddadi, H. Jin, and S. K. Nayar. Multiple view image denoising. In *CVPR*, 2009. 2, 6

[18] M. Zontak and M. Irani. Internal statistics of a single natural image. In *CVPR*, 2011. 7