Fast detection of near-regular deformed image patterns

Schobben, J.

Award date:
2013

Technische Universiteit Eindhoven

Department of Mathematics and Computer Science
and Department of Electrical Engineering

Master's Thesis

Fast detection of near-regular deformed image patterns

By
Jesse Schobben

Supervisor: Gerard de Haan
Tutor: Chris Damkat

August 2013

Many real-life objects have a regular and repetitive texture, such as wall tilings or pavement stones. Being able to detect this structure in images enables several interesting applications. However, detection can be quite challenging; the pattern is often deformed and/or imperfect, requiring a robust detector. On the other hand, for practical applications speed is often very important. Therefore, we want to find a detector that is both fast and robust.

We present a new approach for texture regularity estimation, building upon the Recursive Search Regularity Estimation (RSRE) algorithm which offers very fast detection rates and is robust to a certain degree of spatial deformation, but is sensitive to selection of appropriate parameter values. We discuss optimal settings for a few of RSRE's parameters, and then proceed to suggest a few improvements.

We continue with describing how to interpret the raw vector fields found by RSRE, clean them up and use them to segment the image into individual textures. Afterwards we explain how robustness against contrast and brightness variations can be achieved. Using the correspondence fields, we also investigate how noise reduction can be done, and how spatial deformations can be removed. Finally, we explain how a suitable texture element for each texture can be extracted.

A comparison with the current state-of-the-art approach is made, and it is found that a MATLAB implementation of our method performs at around the same speed, while there seems to be potential for further processing to give results with comparable quality. A fast native implementation could potentially provide an interesting speed improvement.

The detected regular structures are described by a vector field with confidence values. This result can be used for applications such as noise reduction, compression, foreground subtraction, image segmentation, shape reconstruction or geolocation.

Acknowledgements

I would like to extend my gratitude to all the people who have helped me in completing this thesis. Firstly to my supervisors Gerard de Haan and Chris Damkat, for their support and for letting me complete the thesis. Secondly to my friends and family, for several useful suggestions and for their continued support. Furthermore, I would like to thank Minwoo Park for sharing software.

Table of Contents

List of Figures

Acronyms

3DRS 3-Dimensional Recursive Search

BP Belief Propagation

JCF joint compatibility function

KLT Kanade-Lucas-Tomasi (feature tracker)

MRF Markov random field

MSBP Mean-Shift Belief Propagation

MSER maximally stable extremal regions

NCC normalized cross-correlation

NRT near-regular texture

PCF pairwise compatibility function

ROI region of interest

RSRE Recursive Search Regularity Estimation

SAD sum of absolute differences

SSD sum of squared differences

TPS thin plate spline

Chapter 1

Introduction

Real-life objects often have a repetitive appearance in their texture, such as bricks in a wall, windows in a skyscraper or honeycombs. Finding the structure of this pattern in an arbitrary image enables several interesting applications, like efficiently exploiting the repetitiveness for compression, or recovering a regularly textured object's original 3-dimensional shape from a photograph. However, automatically detecting these repeating patterns and their structure is a challenging task. Images commonly contain several different textures with varying amounts of spatial distortion, imperfect repetitions or no repeating structures at all. Furthermore it is not always clear what the texture period is, as in some cases ambiguities exist which require a more global view of the pattern to be resolved. In this thesis we aim to improve upon existing methods for texture regularity estimation, focusing on high speed.

1.1 Problem description

Roughly speaking, images can be divided in two main categories. One of these categories consists of images having mainly stochastic contents, such as a grass field or clouds in the sky. In these kinds of images no repeating structure can be discerned, and in this thesis we are not interested in them. The other category however, contains images that are regular to a varying degree. The boundary between these two categories is fuzzy at best, and in most cases an image can be said to lie between the extremes of perfect regularity and chaotic texture. When we are given an arbitrary image with the task of determining whether it contains any repeating structures, and finding them if so, we need to be capable of dealing with images that are not exactly regular. These near-regular textures can arise due to deformations in the image, of which perspective foreshortening is a very common example. Additionally, part of the texture might be occluded. Furthermore, the actual object itself can have imperfect repetitions; for instance, in an image of building windows at first glance all windows seem to look similar, but usually on closer inspection each individual window

has its own unique detailed look. Finally, the repeating element often varies across the image due to local differences in lighting. Figures 1.1 and 1.2 illustrate these issues.

Figure 1.1: A deformed, near-regular texture.

Figure 1.2: Image of building windows, displaying varying lighting conditions and perspective deformation. The arrows show some of the correspondences that we want to find.

The repeating element in a regular texture is often called a *texture element*, or *texel* [MBSL99]; we will use the same terminology. In a regular, non-deformed image, the most generic shape of this texel is a parallelogram [Sch78]. Therefore, at a specific pixel we can represent a texel using two vectors: the pixel forms one corner of the texel parallelogram, and each vector —representing an adjacent side— relates the pixel to a similar pixel in a neighboring texel. This fully defines the texel, provided that the two vectors are linearly independent. Thus our goal of finding texels in an image, if they exist, can be accomplished by finding these vector pairs for certain pixel positions. If these positions are chosen such that each pixel lies at the endpoint of another vector, the texels form a *lattice* where no texels overlap and the entire texture is covered by texels. Alternatively, the pixels for which to find vectors can be chosen to lie on a regular grid, as shown in Figure 1.2.

A commonly used method to determine whether two regions in an image are similar is to perform block-matching. Rectangular areas around both centers of interest are extracted and their pixels are compared, using a metric such as sum of absolute differences (SAD), sum of squared differences (SSD) or normalized cross-correlation (NCC). Insensitivity to linear changes in brightness and contrast is achieved by first normalizing the blocks, subtracting their mean and dividing by their standard deviation. The block matching at each possible candidate vector gives rise to a score surface, where for SAD and SSD a minimum indicates similarity while in the case of NCC this is a maximum. Unfortunately this score surface often contains local extrema, making it more difficult to find the desired global ones. Furthermore

the issue remains of selecting an appropriate block size, which depends on the texture in question. A small block size increases sensitivity to noise and is unreliable in textures containing uniform areas, due to ambiguities. Large block sizes on the other hand can result in less precise compensation of lighting variations and they can particularly increase sensitivity to spatial deformations.

1.1.1 Recursive Search Regularity Estimation

In video processing, estimating motion vectors that describe the inter-frame motion in a video sequence is often of practical value. An efficient block-based method for doing so is 3-Dimensional Recursive Search (3DRS) [dHBHO93]. It is based on the assumption that moving objects are typically larger than one block and possess inertia; in which case, motion vectors often are similar in spatio-temporally neighboring blocks. 3DRS exploits this similarity by using neighboring vectors to generate a small amount of candidate vectors for each block, yielding good performance, consistency and accuracy.

Recently Damkat and Hofman have proposed a method for texture regularity estimation [DH08], based on 3DRS. This method, Recursive Search Regularity Estimation (RSRE), works by repeatedly scanning the image at regularly spaced grid positions. Its output is a vector field, where for every grid position a vector pair is found; these two vectors describe the local two-dimensional texture period. As with 3DRS, the assumption that vectors in a spatial neighborhood are similar is used. For each position, the vector pairs found earlier in the neighborhood are used as candidates for the current position, as well as slightly perturbed versions of these neighboring pairs. For each candidate vector a block match is performed. The match scores are augmented with penalties, based on vector length and the angle between the pair's vectors. The preferred length is such that the vector points to a neighboring texel, but not to the same texel or a distant one. The angle penalty discourages vector pairs to be collinear. Finally, the best scoring candidate pair is assigned to the current position. This approach inherits the efficiency of 3DRS: it achieves improved accuracy while still permitting real-time application by exploiting the local similarity. Additionally it is capable of handling images in which different textures are present, due to the perturbation of candidates which causes them to converge to a local optimum.

However, RSRE requires certain parameters to be specified. These include things such as block size, perturbation strength and the weight of penalties, for which a balanced value needs to be given; too little as well as too big a value adversely affects the accuracy of the results. On the other hand there are parameters such as grid spacing, which form a trade-off between performance and accuracy. In practice, the quality of RSRE's output strongly depends on good values for these parameters; appropriate values depend on the texture's scale and the amount of deformation, so no parameter set that performs well in general exists. Especially if the image contains several textures that differ in scale it is necessary to change the parameters during processing. For fully automated use, it is thus essential to automatically derive good values for these parameters based on the image itself. Therefore

we need a method that has the benefits of RSRE, but is not dependent on a good choice of parameters. Furthermore, while RSRE adapts well to moderate spatial distortions it is less suitable for strong ones. In real images perspective distortion occurs quite often, as well as distortion caused by the shape of the underlying object not being perfectly flat.

Hence, it is desirable to have a certain robustness to semi-regular spatial distortions in a texture regularity estimation algorithm. Additionally the textured object is not always uniformly lit, having changes in brightness and contrast. Also, when the texture contains large uniform regions it is possible for ambiguities to arise while finding similar regions in the image. Finally it is common for parts of the repeating texture to be occluded or otherwise deviate locally from the pattern.

In summary, we are looking for a method that can quickly detect repetitive patterns in an arbitrary image while being as robust as possible to certain spatial deformations, lighting variations, ambiguities, pattern imperfections and noise.

1.2 Outline of the thesis

We begin by taking a look at existing work in Chapter 2, before our approach is explained in Chapter 3. Chapter 4 contains an evaluation of our approach, based both on real-life images and generated textures. Finally, we make our conclusions and discuss potential future work in Chapter 5. In the appendices, Appendix A illustrates several possible applications of texture lattice detection in general, and how our method can be of use.

Chapter 2

Previous work

In this Chapter we take a look at existing work done in the area of texture regularity estimation. We begin with reviewing the history of this problem, while briefly discussing various existing methods and roughly categorizing them. Next we take a more detailed look at two of the current state-of-the-art approaches. We conclude with a short introduction of our proposed technique, discussing where it fits in among the existing methods.

2.1 An overview of texture regularity estimation

Research on regular textures has been ongoing for several decades already. Initially this was mainly in the field of psychology; it is only relatively recently that this is being done in the context of computer vision. Among the first researchers was James Gibson [Gib50]. Another was Béla Julesz, who researched the ability of humans to distinguish between two similar textures [Jul62], and who introduced the notion of a texton or texel, a texture element. More recently, Leung and Malik proposed a method for detecting texels, by first detecting interesting points, then estimating an affine transform, followed by growing and grouping of the texel(s) [LM96]. Most approaches developed since then can be split in local and global ones. The first category requires texels to be highly regular but is robust to spatial variations between texels, while the second category allows for near-regular texels but makes strict assumptions about the spatial distortion the texels have undergone. In recent years, however, this has changed, and researchers are now directing their attention to methods that take both local and global structure in account.

2.2 Hays *et al*

Hays *et al* [HLEL06] propose an algorithm which unlike earlier methods, ensures that detected lattices have both good local correspondence between texels as well as a globally

consistent geometric structure; it only requires that the underlying texture is a near-regular texture (NRT). The approach is iterative; starting with a set of candidate texels, immediate neighbors are first assigned, followed by global refinement of the found lattice and finally warping each resulting texel to make the lattice spatially regular. The result is used to propose new candidate texels for the next iteration; initial candidates are found using maximally stable extremal regions (MSER) and peaks in the NCC.

Lower-order affinity associated with a vector representing two texel candidates is based on the NCC of the two image patches, while penalties are applied based on the vector's length and the angle with the texel's other neighbor vector. Higher-order affinity between two vector candidates of different texels is determined by the vectors' difference divided by their mean length, having a lower affinity with increasing difference and capped to be nonnegative. Next an approach introduced by Leordeanu and Hebert [LH05] is used to find a satisfying assignment based on these affinities. The global affinity between a vector pair is multiplied with each vector's local affinity to form a confidence value, which is used to fill a sparse matrix M, containing an entry for each possible vector pair. A subset of vectors for which the elements in M have a maximal sum corresponds to the optimal texel assignment, provided that it satisfies certain constraints that disallow the corresponding lattice to have self-assignments, cycles or many-to-one assignments. A binary vector x is used to denote which vectors are part of the lattice; the optimum occurs when $x^T Mx$ is maximal. The binary constraint on x is relaxed to allow approximation of the optimum. From the Rayleigh quotient, it follows that if x is the principal eigenvector of M and has unit norm, then $x^T Mx$ is maximal. This eigenvector is used to find the final binary x, by iteratively zeroing out values that violate the constraints and finding the eigenvector's maximal value.

In order to enforce higher-order regularity, the lattice proposals are refined by discarding low-confidence suggestions and by requiring that a lattice candidate consists only of quadrilaterals. The maximal connected component among the texels is found to obtain the final lattice.

The texels are then warped using thin plate spline warping in order to assist finding new candidates along the border of the lattice, in the next iteration. For each resulting lattice, a score is computed based on the standard deviation among vectors. Due to the warping step, eventually all vector pairs will point in the same two directions if a successful lattice is found.

2.3 Park *et al*

Park *et al* [PBCL09] have proposed a method that addresses some shortcomings in the approach by Hays *et al*. In particular, these shortcomings are that the older method is computationally expensive, that its last-resort region of interest (ROI) finding method is noise-sensitive and that the two vectors in a texel pair are not jointly assigned, making it

less robust. To address these issues, the improved method enforces high-order constraints early on to make it more accurate, it uses inference for increased robustness, and finally it has a much-reduced time complexity (linear in the number of found texels).

The problem of lattice detection is modeled as a degree-4 Markov random field (MRF), where each node represents one texel's position. Spatial constraints are applied between neighboring nodes to enforce their texels to have a similar shape. Furthermore, the texel's appearance is also constrained to be similar to a predetermined template texel.

The approach can be divided in three phases. In phase 1 an initial lattice proposal is determined, based on the Kanade-Lucas-Tomasi (KLT) feature tracker [ST94], applied block-wise. The found interest points are clustered based on appearance, using mean-shift clustering. From each cluster three points which are near each other are randomly picked. From this triplet an affine transform is computed which maps the points to orthonormal unit vectors; this transform is then used to transform all other points in the cluster, and it is checked how many transformed points lie at integer positions. If this number is big, the initial random selection was a good one. This process is repeated several times, and the best result is used as initial lattice proposal for each cluster. Also, based on the various cluster proposals a global projective transform is derived and applied to the whole image, to assist the next phase.

In phase 2, the existing lattice is extended via spatial tracking. This is done using a method called Mean-Shift Belief Propagation (MSBP) [PLC08], which is a heuristic simplification of Belief Propagation (BP) [YFW03]. As with standard BP, a joint compatibility function (JCF) and a pairwise compatibility function (PCF) are used. In this case, the JCF compares the template texel with the current candidate, using the NCC of both the texels themselves and their edge magnitudes. The PCF is based on how consistent neighboring vector pairs are. After MSBP converges the found texels are verified, by checking if the found maxima are locally significant; the ones that are found to be so are selected as texel locations.

In the last phase, phase 3, thin plate spline (TPS) warping is performed: the texels found in phase 2 are warped such that they form a regular lattice.

Phases 2 and 3 are repeated as long as phase 2 finds new texels.

When compared to Hays *et al*, experimental data indicates that processing time is reduced by nearly a factor 10, bringing it in the order of minutes. For the used test set, the number of detected good texels is over 65% of the number of ground-truth texels, which is almost double that of Hays *et al* on the same test set.

2.4 Our proposed method

As we shall see in the next Chapter our method tries to find a balance between speed and accuracy, with the focus lying on speed. It allows for significant spatial distortions, while

texels are allowed to be imperfect copies subject to noise and lighting variations.

Chapter 3

Approach

In this Chapter we introduce a pixel-based approach for texture regularity estimation, extending the concepts used in Recursive Search Regularity Estimation (RSRE). First we use a one-dimensional image to illustrate the basics of our method. In Section 3.1 we begin with explaining how our pixel-based method derives from regular block matching. Next we describe what happens when lighting conditions vary, in Section 3.2. After this we present the overall approach in Section 3.3. Next, we describe ways to cluster the image based on found textures, in Section 3.4. We then proceed to describe in detail how we handle lighting variations in Section 3.5, followed by a discussion on noise removal in Section 3.6. After that, we explain how we can extract the deformation field based on the correspondence fields in Section 3.7. Finally, in Section 3.8, we describe how a good texel can be extracted for each individual texture.

3.1 A look at 1-dimensional block matching

Block matching is a basic method for finding similar regions in an image. However, if the template to match is not known, it is necessary to somehow decide on a block size beforehand. Additionally, block-based methods for template matching can fail to detect a similar region if significant spatial deformations exist relative to the block size. In order to be robust against these deformations, a block-based approach cannot be used without taking into account at the very least affine deformations between blocks. In this section we take a different view at what is actually done when matching blocks.

Figure 3.1 shows a one-dimensional image in row one, containing a perfectly regular pattern. This type of image is well-suited for block matching, due to the lack of distortions or lighting variations. In the figure, the image is put alongside a shifted copy of itself in row two such that the pattern exactly matches. The amount by which the copy is shifted is what metrics such as normalized cross-correlation (NCC), sum of absolute differences (SAD) and sum of

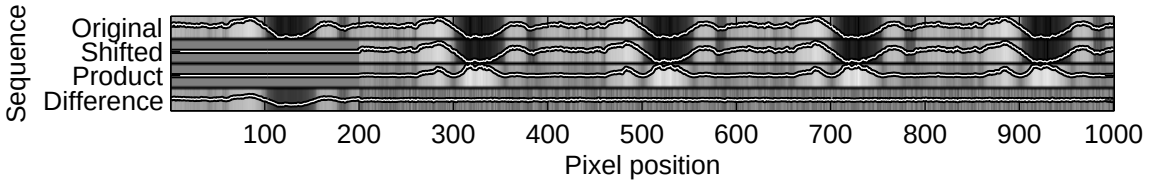


Figure 3.1: The first row shows a one-dimensional image containing a regular pattern, with pixel values ranging from -1 to 1 and having zero mean. Row two is a shifted version of the former, such that the pattern is perfectly aligned with its own repetition. The third row contains the product of the first two rows, while row four holds their halved difference. The brightness mapping is nonlinear, to emphasize zero-crossings.

squared differences (SSD) are meant to find. In the case of NCC the alignment causes a maximum in the sum of products, due to all individual products being nonnegative and peaks having a maximal value since the extrema in the image and the copy are aligned. In the figure the per-pixel products are plotted in row three. SAD and SSD on the other hand are derived from the differences, which are shown in row four. The absolute value of these, squared in the case of SSD, is low if the images are aligned, resulting in their sum being minimal as well.

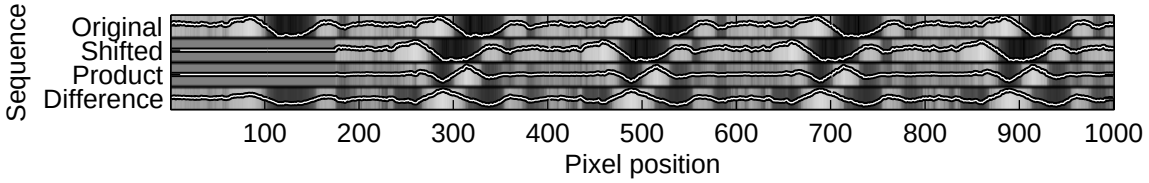
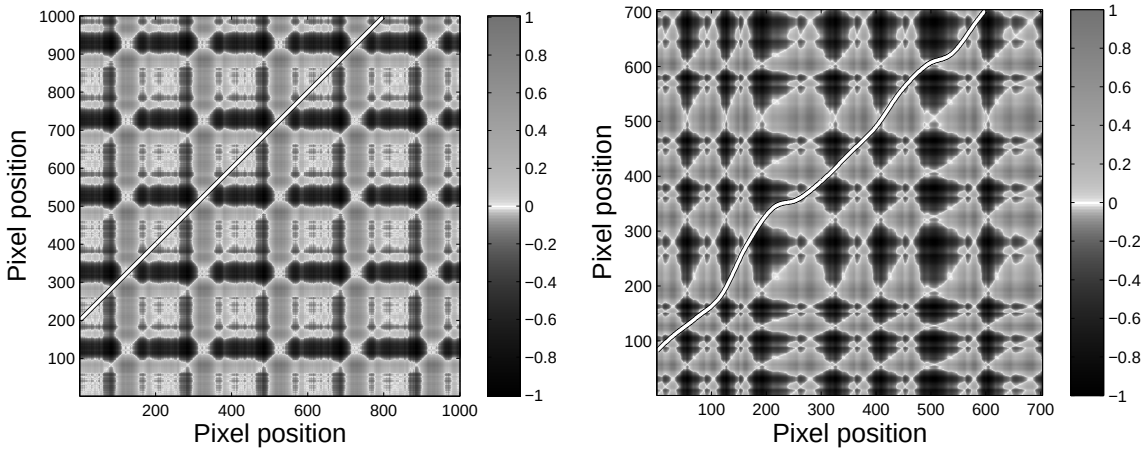


Figure 3.2: The same image as in Figure 3.1, but now misaligned.

Figure 3.2 contains the same image, yet misaligned. Here the Difference clearly deviates significantly from the zero value, giving rise to a higher valued sum of the absolute differences. Likewise, the Product contains a high amount of negative values that lower the sum of the products. From these two sequences, we can thus conclude an alignment exists if the Difference is near-zero or if the Product is mainly positive. This means that on a local scale the Difference gives a more direct indication of alignment than the Product, since it is near-zero everywhere. Additionally, for NCC the image values need to be mean-centered, requiring knowledge of the mean at a local scale. Therefore we focus on the Difference in this section.

If we compute all the Difference sequences for each possible shift amount, we obtain the matrix in Figure 3.3a. The brightness of each element in this skew-symmetric matrix indicates the difference of the image's corresponding pixel pair, with white implying equality and light gray/dark gray representing positive/negative values, respectively. Here the main diagonal corresponds to a shift amount of zero. It contains only white elements, indicating



(a) Difference matrix for Figures 3.1 and 3.2

(b) Difference matrix for Figure 3.4

Figure 3.3: Difference matrices of all pixel value pairs in each figure, where dark gray represents a negative value, light gray indicates a positive value and white stands for zero. This brightness mapping highlights the boundary between positive and negative values. The black-outlined white line marks an *alignment line*, representing the shift amount used in the source figure. Due to the spatial deformation in Figure 3.4, the alignment line in Figure (b) is not straight.

that each pixel is equal to itself. Less unsurprising information can be gathered from the other elements in the matrix. Each diagonal parallel to the main diagonal is also a Difference sequence, having a shift amount defined by the vertical distance to the main diagonal. In the displayed matrix a few of these diagonals are entirely white; these *alignment lines* indicate that for their corresponding shift amount an alignment exists at every pixel. The alignment line which corresponds to Figure 3.1's shift amount is marked; our eventual goal is to automatically find such an alignment line.

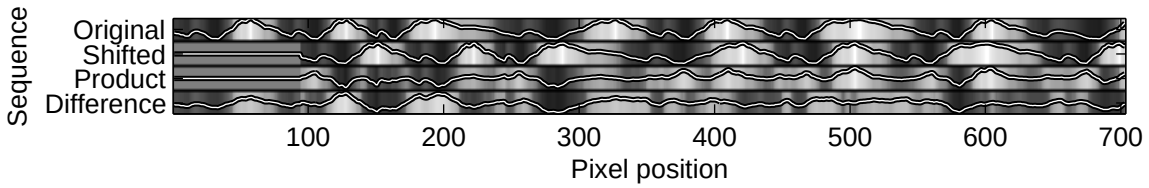


Figure 3.4: Another one-dimensional regular pattern, spatially deformed.

If the pattern is free of spatial deformations such as in Figure 3.1, all of the alignment lines are diagonals. In one-dimensional block matching, normally only elements on the same diagonal are summed together, which succeeds in detecting an alignment line if it coincides with a diagonal. However, if the image contains a pattern that is spatially deformed the

alignment lines are not parallel to the main diagonal, as the shift amount varies between pixels. To illustrate this situation, consider the deformed pattern in Figure 3.4 and its difference matrix in Figure 3.3b. The alignment lines seen in the matrix have a wavy shape, due to the irregular deformation. To detect them, normal block-matching cannot be used. Instead, we use a pixel-based method.

In order to achieve robustness to arbitrary spatial deformations, we exploit the fact that these deformations are typically low-frequency, i.e. the displacement vectors are locally very similar. This allows us to make a prediction of the deformation at a particular pixel, based on previously found deformation vectors of its surrounding pixels. The mentioned one-dimensional image in Figure 3.4 consists of a repeating pattern which is spatially deformed. This is an otherwise ideal image, having texture and uniform lighting everywhere, thereby avoiding ambiguities; consequently, the alignment lines in Figure 3.3b's difference matrix are also obvious, aside from their deformations.

In the difference matrix, we can see that the change in spatial deformation can locally be approximated by a line tangent to one of the alignment lines. The slope of this tangent line varies only slightly from pixel to pixel, giving rise to the idea of tracking this line when given an initial point lying on it.

However, real-life images are less ideal; if we just use the difference between pixels to determine similarity, we will not find a value of zero if lighting conditions vary across the image, which is often the case. What we will find, however, is a more or less constant value.

3.2 The effect of contrast and brightness variations

The images in which we want to detect regular textures often have variations in contrast and brightness, for instance due to shadows or highlights. To decide whether two areas in an image are similar despite different lighting conditions, we need to find a method of comparing them with disregard for these differences. In block-based matching, this is commonly handled via normalization of the block's pixels by subtracting their mean and dividing by their standard deviation. In our case we will use the same technique; however, as we do not use blocks in the traditional sense, we need to find some other way to decide which pixels to consider for normalization. Section 3.5 describes the technique we finally use, but first let us take a closer look at the one-dimensional case.

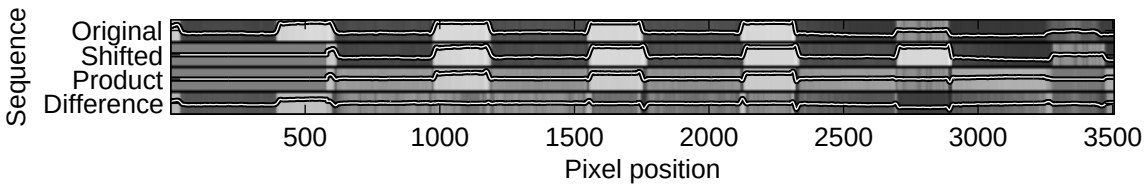


Figure 3.5: A one-dimensional repetitive pattern, containing shadow.

The first sequence in Figure 3.5 shows a one-dimensional slice from a photograph of building windows, affected by a shadow. A shifted version of this texture is displayed also, such that the two sequences are aligned around pixel 1000. The difference between them is visible as well. Because the spatial deformation is low in this case, the textures are nearly aligned at every pixel. Note how the sequences differ between pixels 2500 and 3000 due to the shadow.

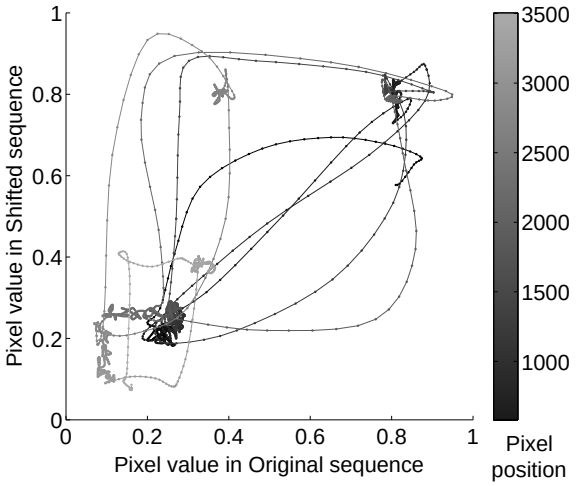


Figure 3.6: Scatter plot visualizing the correlation between the original and shifted sequences of Figure 3.5. Dots are connected if they are neighbors in the texture; the line segments are shaded according to spatial position in the texture.

In order to achieve invariance to brightness and contrast variations, it is good to first get a better understanding of their effect on the pixel values. Figure 3.6 shows a scatter plot visualizing the correlation between the Original and Shifted sequences, where the horizontal coordinates of the points are determined by the pixel values in the Original sequence and the vertical coordinates by pixel values in the Shifted sequence. If the texture would be perfectly regular and the sequences exactly aligned, this plot should show all points lying on the main diagonal. In this case, however, a slight spatial deformation does still exist. This causes only two edges in the texture to be reasonably aligned, visualized in the scatter plot by the two nearly diagonal line segments. Areas in the sequences with little variation result in tight clusters of points; however, if one sequence varies significantly without the other doing so, a mainly horizontal or vertical line can be seen, indicating misalignment for that part in the sequences.

Figure 3.7 shows a different texture, artificially generated to have strong variation in brightness and contrast. Figure 3.8a shows its associated scatter plot. In this plot, we can see that the points lie mostly along diagonal lines, indicating that the sequence is properly aligned at every pixel. Due to the big variations in contrast the slope of these lines varies, while the changes in brightness cause a variation in the intercept. For comparison, Figures 3.9 and 3.8b show the same texture with a slightly different shift amount, resulting in a radically different scatter plot.

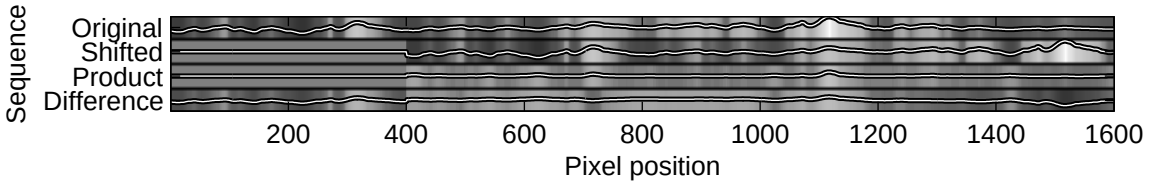
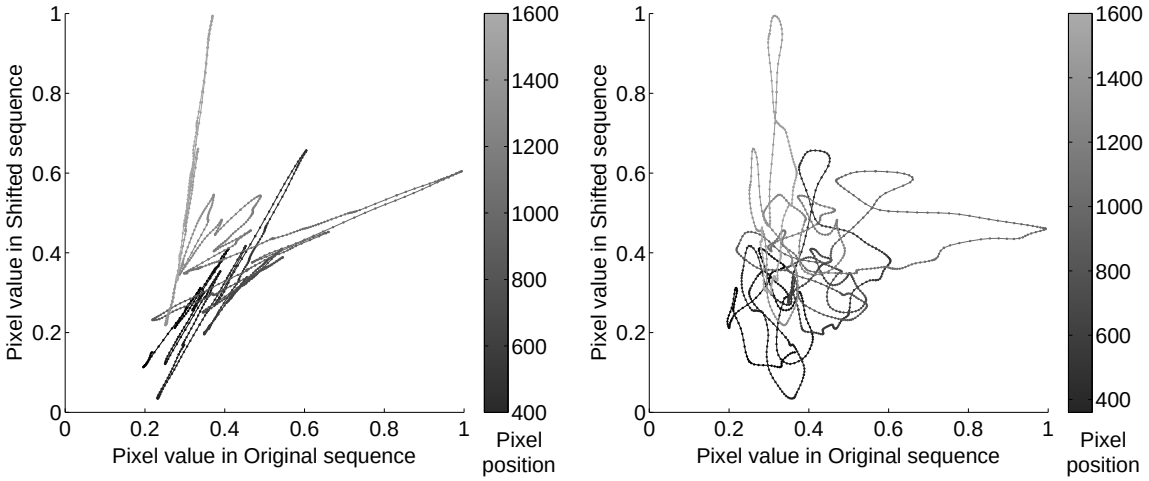


Figure 3.7: Artificial texture, containing significant variations in brightness and contrast



(a) Scatter plot of Figure 3.7's texture

(b) Scatter plot of Figure 3.9's misaligned texture

Figure 3.8: Note how most points are locally collinear in Figure (a), indicating alignment. In Figure (b), the points are clearly less organized.

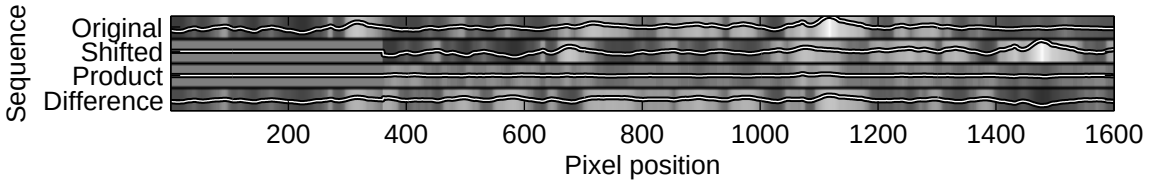


Figure 3.9: The same texture as in Figure 3.7, with a different shift amount

3.3 Finding the correspondence fields

After these one-dimensional visualizations, now it is time to look at the two-dimensional case. The method we use to extract the correspondence vector fields is based on RSRE. First, let us recap how RSRE works: for each vector position in a grid with a certain step size, several candidate vectors are generated, based on neighboring vectors. Randomly perturbed versions of the candidates are added as additional candidates. For each of them, a block match is performed between the candidate's origin and its target position. Furthermore, several penalty values are computed, which are based on things such as the candidate vector's length or the angle it makes with the other vector in its pair. Ultimately, the candidate with the best match score and lowest penalty is chosen. The vector positions are processed in a zig-zag order, which helps the convergence speed (see Figure 3.10). Several such zig-zag passes are done.

Figure 3.10: Zig-zag scanning pattern.

RSRE in its original version depends on a number of parameters, such as block size, spacing between vectors, ideal vector length and noise power. The quality of its output depends heavily on good values for these parameters, so to make the algorithm usable in practice some good values for these parameters have to be determined automatically. It is therefore worthwhile to begin with examining these parameters in more detail, and what effect they have on performance and quality.

Probably the most important parameter of the original RSRE algorithm is the block size used. This does not only affect the block match size, but also the step size in the vector field. Furthermore, the initial lengths of the vectors in the two correspondence fields are initialized to half the block size (their angles are set to respectively fully horizontal and fully vertical). Another important parameter is the kernel size, used to scale the vector length penalty. Noise power is also among the parameters: this affects how strongly candidate vectors are perturbed. All these parameters have an optimal value which depends on the texture scale, so the performance of RSRE depends on how well these parameters match the texture. Other parameters include the number of scans performed and scaling constants for the penalties.

3.3.1 Improvements

We decouple parameters such as step size and initial vector values from the block size, so that we can investigate each parameter independently.

A rather trivial case is the step size parameter. If it is high, RSRE will execute very fast, but the results are not likely to be good since there is barely a chance for convergence to occur. When the step size is set very small result quality is greatly improved, as there are much more vectors, which helps convergence. However, this comes at the expense of computation time.

Self-assignments are not desired, so RSRE uses a penalty for zero-length vectors, scalable using the kernel size parameter and a scaling constant *zpen*: $zpen * e^{-\frac{|v|}{\text{kernelsize}}}$, where v is the candidate vector. We take a simplified approach to this, by using a cutoff: vectors with a length shorter than a certain value are excluded. For this, we define the parameter MINSIZE. The reasoning behind this is that vectors shorter than this amount are considered not relevant, while longer vectors are long enough for block matching and propagation to indicate whether they are correct or not. The downside is of course that structures which have a period shorter than this parameter can not be detected, but the advantage is scale-invariance. A value of 4 has experimentally been found to perform well.

The block size parameter influences the block matching step; setting this parameter too low causes high performance but inaccurate matching, while setting it too high makes the matching slow. Also, a large value does not always result in better quality, especially in the presence of strong spatial distortions.

As a compromise, experiments have indicated that a step size of one pixel combined with a minimal block size gives good results in general. A block size of 3x3 was found to be the best compromise. The performance is still acceptable at this setting, as the smaller cost of block matching compensates for the added overhead of matching at every pixel. As an added bonus, the algorithm is now more robust against spatial deformations. It becomes, however, very important to enforce neighboring vectors to be similar, as matching such small blocks is very noise-sensitive.

To give more opportunities for the algorithm to find good vectors, it is useful to generalize RSRE's two vectors per position. This of course means the vector fields do not directly map to the two correspondence fields we want to find, so they have to be extracted in a post-processing step. Thus, we introduce the number of vectors per position as an additional parameter VECTNUM. Good results have been obtained with four vectors, so from here on, we fix this parameter to four.

The initial vector values influence to which solution RSRE converges. It is therefore important that these are initialized with a sensible value. Setting them all to half the block size is not an option anymore with our 3x3 blocks. Instead we assign a random value to each vector; specifically, we use a normal distribution to assign the angles and an absolute Cauchy distribution to assign the lengths. For the angle distribution, we choose as mean for

each vector the value $\frac{n}{\text{VECTNUM}} * \pi$, where n is the vector number ($0 <= n < \text{VECTNUM}$); as standard deviation, we choose $\frac{0.5*\pi}{\text{VECTNUM}}$. This spreads out the initial vector angles to cover 180 degrees. For the vector lengths, the Cauchy distribution's property of not having a defined mean or variance represents the fact that we don't know our texture's scale up-front. As we are only interested in positive vector lengths of at least MINSIZE, we take a random sample's absolute value and add MINSIZE to it. As scale parameter, the image diagonal's length divided by a constant INITIAL_SCALE is used; the value fifty appears to work well. In summary:

$$\begin{aligned} \text{angle} &= \frac{n}{\text{VECTNUM}} * \pi + X_1 * \frac{0.5 * \pi}{\text{VECTNUM}} \\ \text{length} &= \text{MINSIZE} + |X_2| * |\text{image\_diagonal}| * \text{INITIAL\_SCALE} \end{aligned}$$

with X_1 being sampled from a normal distribution and X_2 from a Cauchy distribution.

Due to the single-pixel step size we have a lot of these random vectors, and there is a reasonable probability that a handful of them will point in a good direction, which will cause them to have a better matching score than their neighbors and a better chance at being selected as winning candidate by a neighbor; thus propagating the few successful ones. This simple random initialization strategy turns out to work surprisingly well, as we will see.

To reduce the impact of ambiguities due to uniform areas in the image, we compute a variability map for the entire image, where each pixel contains the standard deviation of a 3x3 block centered at its position. This can be done efficiently using the approach described in Section 3.5. The formula we use to compare the local block B at (y, x) with a candidate block C at $(y, x) + v$ is: $\frac{\sigma[B]}{1+\text{avg}[|B-C|]}$, provided that $|v| >=$ MINSIZE (if not, the score is zero). A higher score is better.

Similar to RSRE, we then proceed to scan over the image in a zig-zag pattern. At each position, we compose a set of candidates. This set consists of the local set of vectors, as well as those of the immediate neighbors. For each of these candidates we create a version randomly perturbed with two-dimensional Gaussian noise of strength NOISE_STRENGTH$*$ $|v|$, with NOISE_STRENGTH again being a parameter; the value 0.1 seems to work good in practically all cases. For each of the original/perturbed vector pairs, we do a block match and keep the vector with the best score. The candidate set is next checked for (near-)duplicates, which are removed; two vectors are considered the same if their absolute difference is below MINSIZE. Finally, the best VECTNUM candidates are stored as new vectors for the current position.

Note that unlike RSRE, no penalties have been used yet, aside from the MINSIZE cutoff. This means that vectors might still be collinear, or too long. However, because we have more than two vectors, collinearity is less of a problem; in practice, with four vectors there are nearly always two that are not collinear. As for the lengths, it does occur relatively often

that vectors point one texel too far; especially when the texture period is small relative to the image size. There seem to be almost always at least a few correct vector lengths though, which we can make use of in a post-processing step.

We do introduce one penalty: a penalty to encourage a smooth vector field, which is needed due to our small block size, as mentioned earlier. This penalty is defined as follows: it is the absolute difference between the candidate vector and the most similar neighboring vector — both horizontally and vertically — in the direction our scan has just visited. If the current location is (y, x), and the scan is currently moving bottom-to-top, left-to-right, then these neighbors are $(y, x-1)$ and $(y+1, x)$, and the penalty value for candidate v is:

$$\begin{aligned}\text{nonSmoothPenalty}(v) = &\min(i : 0 <= i < \text{VECTNUM} : |v - (y, x-1).v_i|) \\ &+ \min(i : 0 <= i < \text{VECTNUM} : |v - (y+1, x).v_i|)\end{aligned}$$

This penalty is applied by subtracting it from the score.

One thing we can keep track of in the first phase, is which candidate was picked at each position, and in particular where that candidate originated from. If the candidate was obtained from a neighbor (permuted or not), then we can group the chosen candidate and its original vector together. This way, a grouping of vectors can be made, where the size of the group indicates how often the vectors in that group were propagated. This can give an indication of confidence; a vector that was propagated only very few times, is likely not a good vector in a global context. On the other hand, a vector that does propagate a lot is more likely to be a correct representative for the true correspondence field. This can be combined with the image's local standard deviation to obtain a confidence value, based on position and vector.

The above procedure still results in a relatively chaotic vector field, however it does tend to contain a large amount of correct correspondences. See the "raw" vector field in Figure 3.11. For some applications, this raw field can suffice; for others, it is necessary to extract the full correspondence fields, which can be derived from this raw field.

3.4 Clustering individual textures

It is often the case that an image contains multiple (near-) regular textures, such as in a photo of a city skyline where typically each skyscraper has its own texture. Furthermore, it is even more common for an image to also contain non-regular areas, such as a cloudy sky in the aforementioned cityscape example. In order to do anything useful with the raw correspondence fields we have found, it is necessary to segment them into individual textures, and into non-regular (stochastic) areas as well. We are also interested in the approximate centers of each texture, for some of the methods discussed further on.

As we do not know up-front how many clusters we will get, the usual clustering methods such as k-means cannot be used. A method that is commonly used in this scenario is mean-shift clustering.

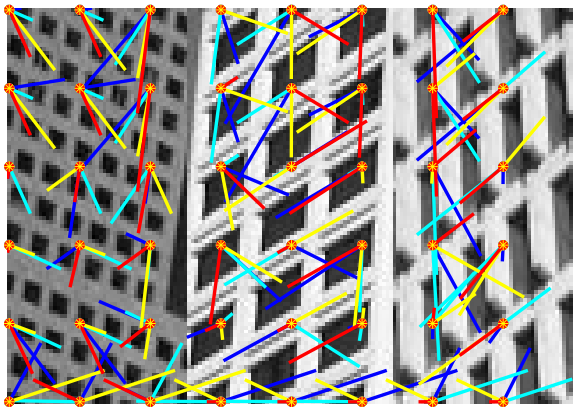


Figure 3.11: This figure shows the "raw" vector field, the result of the first phase of correspondence field detection, for an image with three textures. Best seen in color. Note that only an arbitrary subset of the vector field is shown, to maintain overview. In reality, a quartet of vectors has its origin at each pixel.

Since we have per-pixel correspondence vectors, it is possible to obtain a high-resolution clustering, as for each pixel we can determine whether it belongs to a texture or not. The method we use to determine this is as follows. We start with a few cluster seeds in locations in the image where there is a high confidence in the found correspondence vector field. Neighboring vectors of these locations which are locally similar and also have a high confidence, are added to the cluster. Wherever there is a sudden change in the vector field, accompanied by low confidence, we mark the boundary of our cluster. Stochastic areas, such as sky, will typically have very low confidence values, so they should be relatively easy to detect. After clustering is complete, we select as center points for each cluster the cluster's mean; we shall refer to these resulting clusters as *texture cluster*.

This simple approach has a few shortcomings in practice. It is possible that the detected correspondence field is not accurate enough due to some local texture obstructions, which means that a sudden change in vectors can be a false alarm, and the two areas do actually belong to the same texture. This problem is best handled in the correspondence finder, by taking in account whether a certain popular vector shape reappears elsewhere in great numbers.

3.5 Normalizing brightness and contrast

Typically, a near-regular pattern is subject to shade or highlights, as shown in Figure 3.12. The usual way to handle this when comparing image blocks is to normalize the block's pixel values, such that they have zero mean and a standard deviation of one. For best results, the normalizing block's size should match the texel size.

Figure 3.12: An image with varying brightness and contrast, due to shadows and highlights.

Since we don't know the texel sizes up-front, and moreover since they may vary across the image, this normalization should be done as part of the lattice finding procedure. If we restrict ourselves to normalizing rectangular blocks, an effective technique commonly used to improve performance can be used: the integral image [Cro84]. It makes calculating the mean of an arbitrary rectangular block a constant-time operation, irrespective of the block size.

If we define I as our input image, B as a rectangle bounded by (x_1, y_1) and (x_2, y_2), and $N = (x_2 - x_1 + 1) * (y_2 - y_1 + 1)$ as the number of pixels in B, then we can define the mean of B:

$$E[B] = \frac{1}{N} \sum_{\substack{x_1<=i<=x_2 \\ y_1<=j<=y_2}} I_{j,i}$$

The sum-term can be split as follows:

$$\sum_{\substack{x_1<=i<=x_2 \\ y_1<=j<=y_2}} I_{j,i} = \sum_{\substack{0<=i<=x_2 \\ 0<=j<=y_2}} I_{j,i} - \sum_{\substack{0<=i<x_1 \\ 0<=j<=y_2}} I_{j,i} - \sum_{\substack{0<=i<=x_2 \\ 0<=j<y_1}} I_{j,i} + \sum_{\substack{0<=i<x_1 \\ 0<=j<y_1}} I_{j,i}$$

We now pre-compute the integral image II such that:

$$II_{y,x} = \sum_{\substack{0<=i<x \\ 0<=j<y}} I_{j,i}$$

This can easily be done in linear time:

$$\begin{aligned}
II_{0,x} &= \sum_{\substack{0<=i<x \\ 0<=j<0}} I_{j,i} = 0 \\
II_{y,0} &= \sum_{\substack{0<=i<0 \\ 0<=j<y}} I_{j,i} = 0 \\
II_{y+1,x+1} &= \sum_{\substack{0<=i<x+1 \\ 0<=j<y+1}} I_{j,i} \\
&= \sum_{\substack{0<=i<x+1 \\ 0<=j<y}} (I_{j,i}) + \sum_{0<=i<x} (I_{y,i}) + I_{y,x} \\
&= \sum_{\substack{0<=i<x+1 \\ 0<=j<y}} (I_{j,i}) + \sum_{\substack{0<=i<x \\ 0<=j<y+1}} (I_{j,i}) - \sum_{\substack{0<=i<x \\ 0<=j<y}} (I_{j,i}) + I_{y,x} \\
&= II_{y,x+1} + II_{y+1,x} - II_{y,x} + I_{y,x}
\end{aligned}$$

This allows us to compute the mean of B using four lookups in II:

$$E[B] = \frac{1}{N} * (II_{y_2+1,x_2+1} - II_{y_2+1,x_1} - II_{y_1,x_2+1} + II_{y_1,x_1})$$

Obtaining the standard deviation can also be sped-up, although it is a bit more involved. If we define:

$$IIS_{y,x} = \sum_{\substack{0<=i<x \\ 0<=j<y}} I_{j,i}^2$$

where IIS is computed similar to II, then:

$$\begin{aligned}
Var[B] &= E[(B - E[B])^2] \\
&= \frac{1}{N-1} \sum (B - E[B])^2 \\
&= \frac{1}{N-1} (\sum (B^2) - 2 * E[B] * \sum (B) + N * E[B]^2) \\
&= \frac{N}{N-1} (\frac{1}{N} \sum (B^2) - 2 * E[B] * \frac{1}{N} \sum (B) + E[B]^2) \\
&= \frac{N}{N-1} (\frac{1}{N} \sum (B^2) - E[B]^2) \\
&= \frac{N}{N-1} (\frac{1}{N} (IIS_{y_2+1,x_2+1} - IIS_{y_1,x_2+1} - IIS_{y_2+1,x_1} + IIS_{y_1,x_1}) - E[B]^2)
\end{aligned}$$

where sums are over all elements of B and the squaring operation is element-wise.

Finally, the standard deviation is $\sqrt{Var[B]}$.

As our texels are typically not axis-aligned, we can derive an axis-aligned rectangle R from a given texel defined by vectors U and V by calculating the texel parallelogram's area: $A = |U_x V_y - U_y V_x|$. The vector component from (U_x, U_y, V_x, V_y) which is closest to $\sqrt{A}$ is taken as one side of the rectangle, and the other side's length is chosen such that the rectangle's area is equal to the parallelogram's. This results in a rectangle which is as close to a square as possible, yet also has a close relation with the true texel's shape. See Figure 3.13.

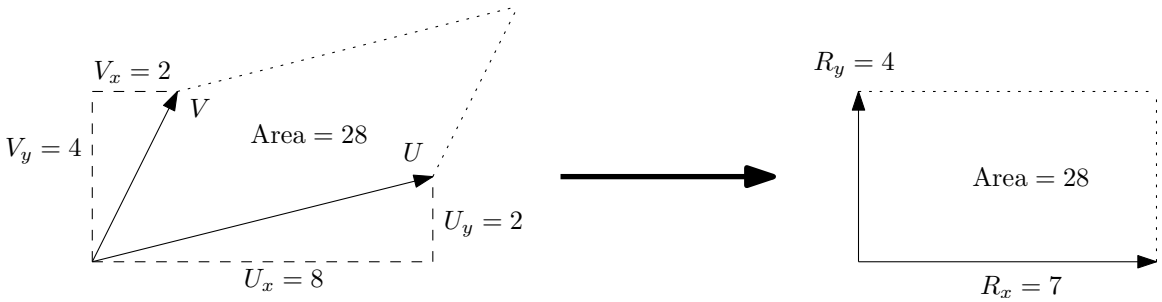


Figure 3.13: Deriving an axis-aligned rectangle R from a texel defined by vectors U and V. Here, V_y is the vector component closest to $\sqrt{28}$, so $R_y = V_y = 4$ and $R_x = \frac{|U_x V_y - U_y V_x|}{V_y} = 7$.

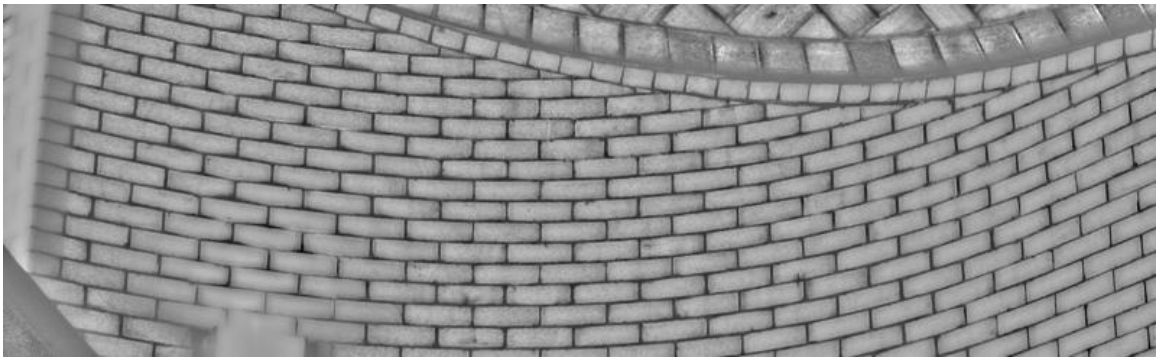


Figure 3.14: The same image as in Figure 3.12, after removing brightness and contrast variations.

We integrate this light normalization into the algorithm of Section 3.3.1 for finding the initial "raw" correspondence fields; as is usually done, the block matching step is adjusted so that both blocks are normalized before matching. However, we obtain the mean and standard deviation using the method described in this section, using two of the vectors at the current match position. To be specific, we take the two vectors with the largest confidence (i.e. the largest group size), provided they are not collinear.

3.6 Adaptive noise reduction

Once we have found the correspondence fields, it becomes easy to jointly perform operations on pixels in different texels. We can define a "pixel group", which is the group of pixels that have the same relative position in each texel. See Figure 3.15. One of the most basic applications is noise reduction. The naive approach is to simply average all the pixels in a pixel group, however this will only give good results if the image is perfectly regular and we have managed to find the exact correspondence fields. When these conditions do not hold, reckless averaging of pixels causes loss of detail as illustrated in Figure 3.16b.

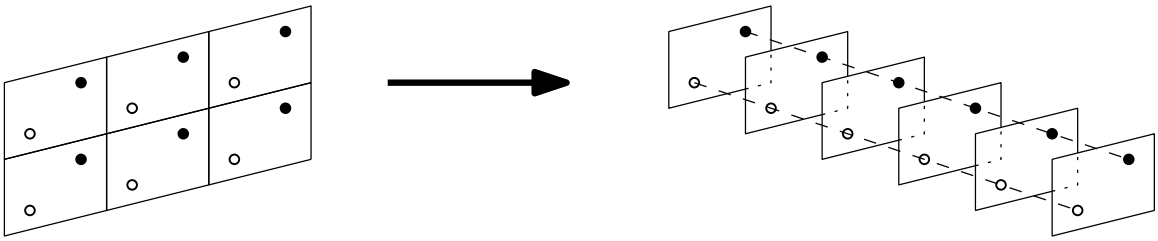
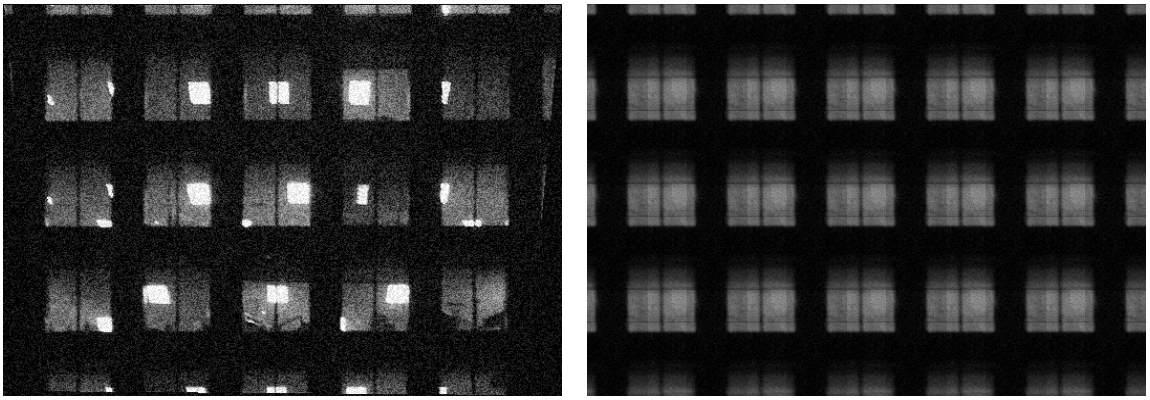


Figure 3.15: Texels arranged in 3D such that pixel groups align, which are indicated by the dashed lines.

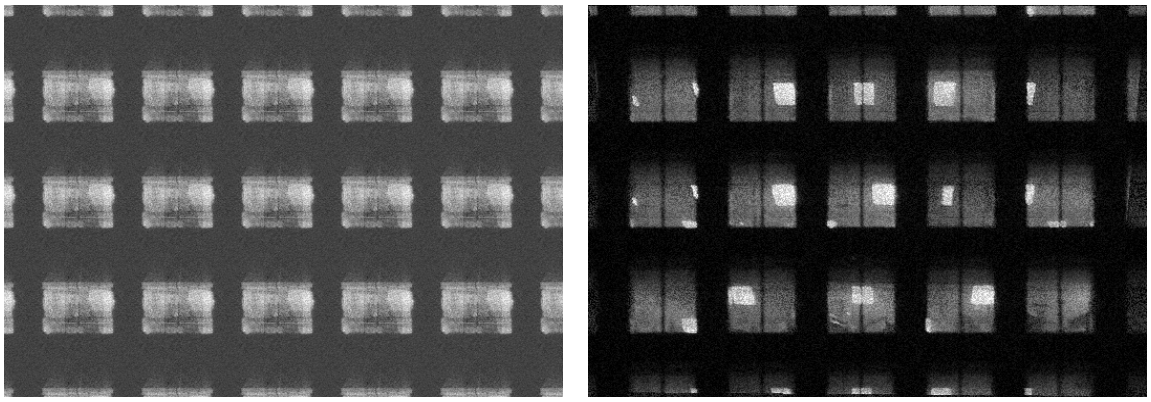


(a) (b)

Figure 3.16: When naively averaging pixels across texels when the input image's texture is not entirely regular (a), local details are lost where texels differ (b).

One possible improvement is to make the assumption that the noise has a low amplitude; in that case, pixels that differ in value due to noise alone will still be relatively similar. Applying an averaging filter to texel pixels can thus be made adaptive by first looking at the pixels' variance, for instance, and using that to determine the strength of averaging; if there is a high variance among pixels in a pixel group, the texel region likely is not regular and averaging should probably not be done in that area. Figure 3.17b illustrates this: the

same input image as in Figure 3.16a has been used, however this time the averaging strength s is adapted to the standard deviation of each pixel group (shown in Figure 3.17a). This is used to linearly interpolate between the pixel value p and the pixel group mean $E[PG_i]$, using interpolation factor $f = c * \sigma$ where c is a normalization constant: filteredValue = $p*(1-f)+E[PG_i]*f$. This gives a much better result (Figure 3.17b). The noise reduction could be improved further by using more advanced adaptive filtering, or by making use of locally neighboring pixels as well in addition to those in the same pixel group.



(a) (b)

Figure 3.17: By adapting the averaging strength to each pixel group's standard deviation (a), details can be preserved (b).

3.7 Deriving the deformation field

Based on the two correspondence fields that we have found, it is of interest if we are able to determine the exact spatial deformation field for each texture in the image. With this field, it would be possible to "unwarp" the textures such that they become spatially perfectly regular. Furthermore, this "warp field" can give us some geometrical information about the textured objects in the image, such as perspective or 3-D shape.

The warp field that we intend to find, specifies for each pixel a vector, which indicates where that pixel should move to in order to unwarp the image. The effect of the unwarp operation on the two correspondence fields we have is similar: all vectors in a correspondence field which belong to the same texture cluster should become the same. This property allows us to derive the warp field from the correspondence fields as follows.

We start by deciding on a "target" vector for each texture cluster to which the cluster's part of the correspondence field should be warped. This can for instance be the median of the high-confidence vectors near the cluster's center. For each correspondence vector, we can now easily calculate how much its tip should move relative to its origin to make it equal

to this target vector, by subtracting the correspondence vector from the target vector. See Figure 3.18.

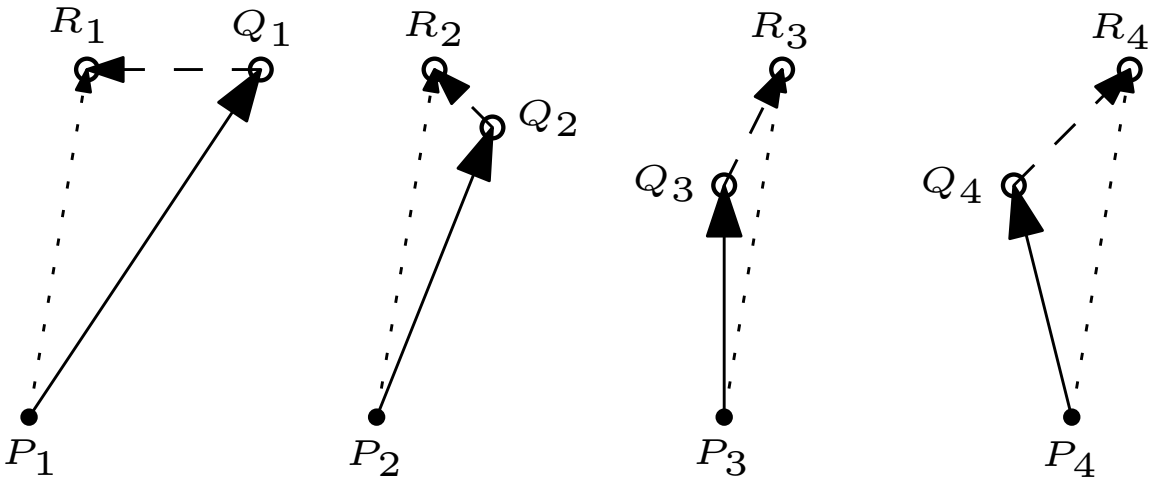


Figure 3.18: The solid vectors (P_iQ_i) represent part of one of the correspondence fields, the dotted vectors (P_iR_i) are the target vector field, and the dashed vectors (Q_iR_i) are the relative warp field. To unwarp this field, the points Q_i should move to R_i, relative to P_i.

However, now we only know what the relative warp vectors are for the tips of the correspondence vectors, relative to their individual origins. Most of those origins are themselves the tip of other correspondence vectors. We can follow all such chains of relative dependencies as much as possible, storing for each point what the reference point is and the warp vector relative to it, until a minimal set of unknowns remain. We do have some freedom in selecting where in the image this set of unknowns lies, by choosing to move either the correspondence vector's tip or its origin. For our purposes, we'll choose the unknowns' location to be as close as possible to the line that passes through the texture cluster's center, and is orthogonal to our target vector. This leaves us with a band of unknowns across the texture cluster, having the same thickness as the length of the target vector. Figure 3.19 illustrates all this. In the figure's case, we can resolve the relative warp vectors to depend only on one of the unknowns as follows: if we define $A(x)$ to be the absolute warp vector for point x, then $A(Q_3) = A(P_3) + Q_3R_3 = A(Q_2) + Q_3R_3 = \ldots = A(P_0) + Q_0R_0 + Q_1R_1 + Q_2R_2 + Q_3R_3$. Similarly, the bottom-right points can be made dependent on $A(P_7)$ alone.

If we perform this procedure for both correspondence fields, we end up with two bands of unknowns, intersecting in the texture cluster's center. By using the relative pointers in both relative warp fields, we can reduce the unknown area to be the intersection of the bands of unknowns, which corresponds to the texel in the cluster's center.

It remains to assign absolute warp vectors to these unknown positions, after which all other warp vectors can be derived. If we simply assign zero-length vectors, the resulting warp field does produce a regular texture when applied. However, it will probably not look the way we want it to, as there will typically be strong discontinuities in the warp field. See

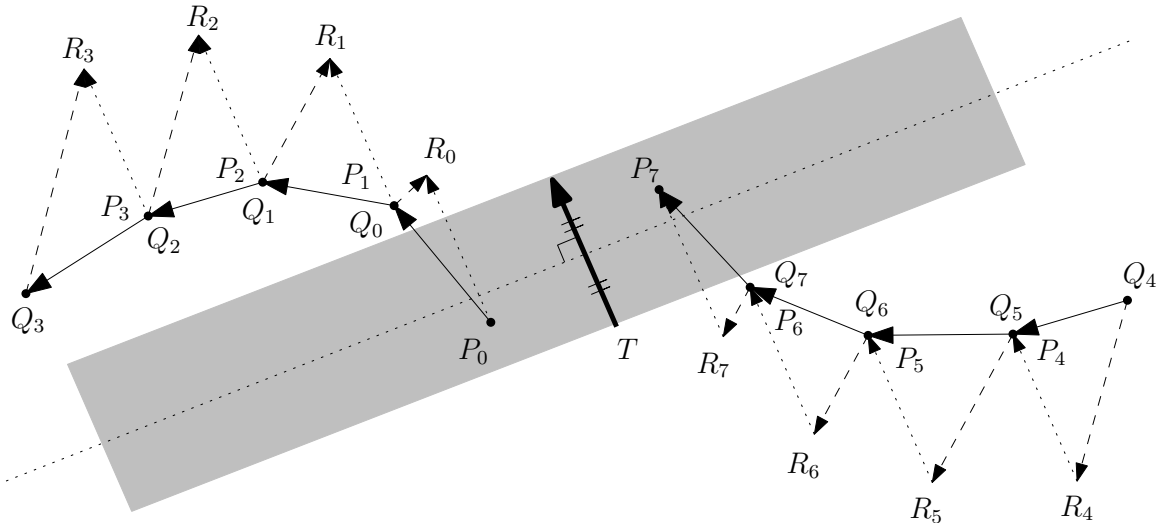
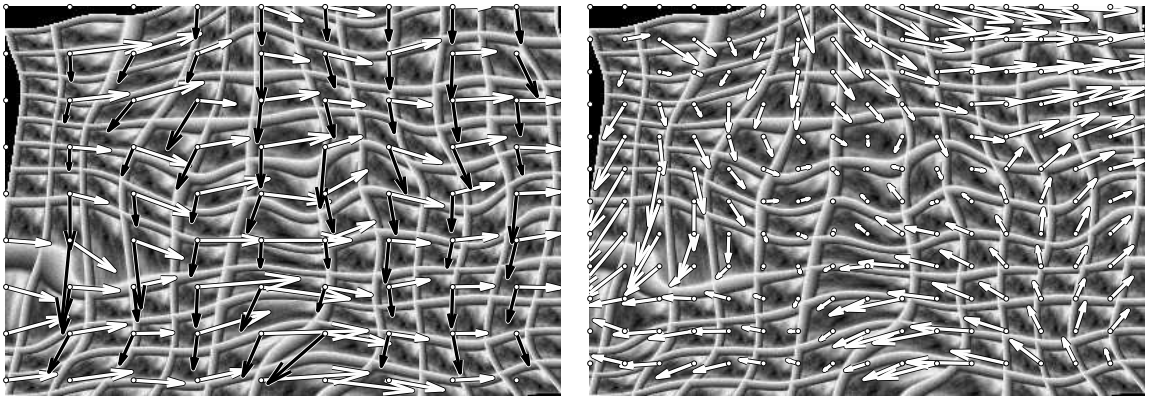


Figure 3.19: Part of a correspondence field with extreme deformation. The central vector T represents the target vector, the gray area represents the unknowns. On the upper left is a chain of correspondence vectors $P_0Q_0 \ldots P_3Q_3$, for which we want to move the vector tip Q_i towards R_i, relative to the origin P_i. On the bottom right the reverse situation with correspondence vectors $Q_4P_4 \ldots Q_7P_7$, where we want to move the vector origin Q_i towards R_i, relative to the vector tip P_i. This gives for all points Q_i a warp vector Q_iR_i, relative to P_i.

Figure 3.20a and Figure 3.21a. In order to have a more aesthetically pleasing result, the unknowns should be assigned such that there are minimal discontinuities in the warp field.

Obtaining a smooth warp field can be achieved by performing Least Squares optimization. For each unknown absolute warp vector, a complex variable is made, representing a vector. For each of these variables, two constraints are applied: the variable should be as similar as possible to both its upper and left neighbors. For unknowns on the edge of the unknown area, this causes the constraints to be applied on dependent vectors (another unknown and a relative warp vector).

Ultimately, the Least Squares method finds a set of value assignments for the unknowns; from this, the dependent values can also be assigned. The resulting warp field, however, describes how the texture was warped; it has to be reversed before we can perform any un-warping. This is relatively straightforward: we create a new vector field in which the origins and tips of the absolute warp vectors are swapped, that is, we place a vector at the location of each original vector's tip, pointing to the original's origin. This inverted warp field can then be used to remove any spatial deformation the texture has, by putting the pixel value each vector points to in the warped image at the vector's origin in the un-warped image; see Figure 3.21b.

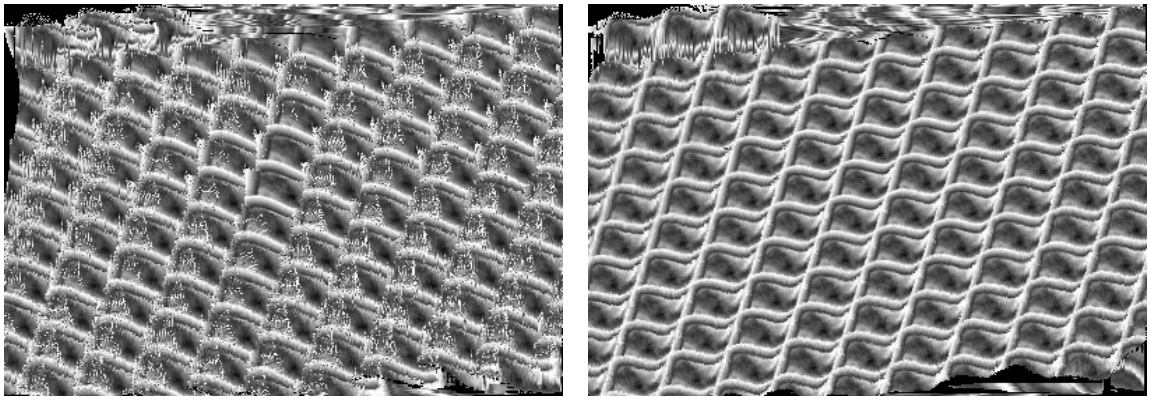


(a) (b)

Figure 3.20: Synthetic image with extreme spatial warping (see Section 4.1), shown with correspondence fields overlaid (a). Also shown with absolute warp field (b), which after inversion is used to obtain Figure 3.21b.

3.8 Extracting texels

For some applications, it can be of interest to obtain the texel of a near-regular texture. It is straightforward to obtain the corners of an arbitrary texel given the correspondence fields. If we pick a point (p_y, p_x) as the first corner of our texel, then we can trivially



(a) (b)

Figure 3.21: Assigning zero-length vectors to the unknowns produces a more or less regular texture (a), but a much more pleasing result is obtained by an assignment that minimizes discontinuities (b).

find the adjacent corners of the texel using the correspondence fields $V1$ and $V2$: they are $(q_y, q_x) = V1(p_y, p_x)$ and $(r_y, r_x) = V2(p_y, p_x)$. The opposite corner can be found either as $(s_y, s_x) = V1(r_y, r_x)$ or as $(s_y, s_x) = V2(q_y, q_x)$; both should give the same result if the correspondence fields are correct, consistent and there is little spatial warping.

After finding the corners, the pixel values of the texel need to be extracted. As the texels within the texture may be near-regular, the whole texture should be considered to find pixel values that best represent all the texture's texels. This can be done in a way similar to the noise reduction technique discussed in Section 3.6; in fact, the pixel groups introduced there are practically all we need to derive our texel's pixel values. We use the pixel group mean to assign each pixel value of the texel, and we use the pixel group variance to determine the texel's confidence field.

However, even after completely regularizing a texture, not just any arbitrary texel will suffice for certain applications. This is especially true if the texel is meant for human eyes. To be more specific, it is often the case that a regular texture has some symmetries. These symmetries can be rotations by a certain number of degrees, or they can be horizontal or vertical reflections, sometimes combined with intra-texel translations.

It turns out, that in two dimensions there are exactly 17 such "symmetry classes", and their formal name is Wallpaper groups [Sch78]. Regular patterns are classified into one of these groups based on their symmetries; translation symmetry, which is our main interest in this thesis, is among the symmetries used to classify textures.

We won't go in detail on these Wallpaper groups, instead we'll focus on how to extract a texel which fulfills these symmetries as much as possible. The procedure is not very surprising: we begin with un-warping, de-noising and normalizing the lighting of the image.

We check if a given texel is symmetric when horizontally or vertically mirrored, or when rotated 60, 90, 120 or 180 degrees, combined with a small, intra-texel translation. We choose as texel corner (p_y, p_x) a point which results in a texel with high similarity to its own reflections, rotations and/or translations.

As it may be the case that the texel is only partially regular (such as when it has non-regular occlusions), we use the confidence value for each pixel to perform weighted matching, so only the regular part affects the selection.

Chapter 4

Evaluation

In order to have direct control over how "difficult" the test images are, and as well as to have an automatic ground truth available, we use synthetic images as part of our result measurements. We also use several real-life photos, sourced from the CMU Near-Regular Texture Database [NRTdb].

We compare results of our approach with the current state-of-the-art, the algorithm by Park *et al* [PBCL09].

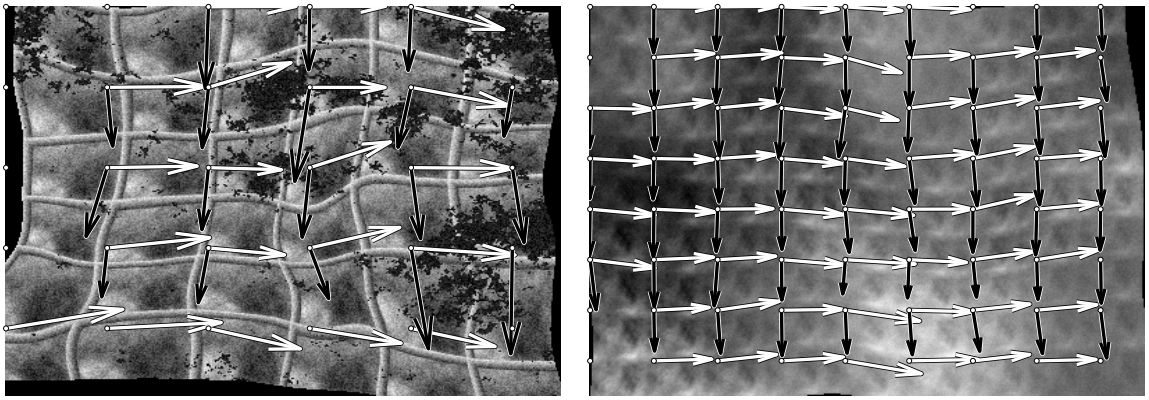
4.1 Synthetic texture generation

As the basis of our texture generator, we use a simple fractal noise function. This function adds several "octaves" of low-pass filtered white noise, where each successive octave has twice the frequency and half the amplitude. In order to have repeatable results, we also specify the seed used to initialize the pseudo-random number generator. We also use a scale parameter, which allows increasing resolution and detail, while still having the same low-frequency information as a lower resolution version. For cases where the noise function needs to be tiled, we perform some blending on the edges, so that opposite sides are similar.

The most basic application of this noise function is to generate a texel, and repeat it several times. This forms a regular texture. For this, we use a parameter which determines how many repetitions there are. We also support drawing a grid on the regular texture with selectable thickness, to help visualization and/or correspondence field finding. By multiplying such a texture with another noise image, shadows and highlights can be simulated, with the desired strength and frequency. We also use the noise function to obtain a random warp field, in order to deform the image; for this, we use two separate noise fields, one for the horizontal direction and another for vertical. Finally, we define two uniform correspondence fields based on the regular original image, and warp them using the warp field. Furthermore, we can add a selectable amount of white noise. To simulate imperfections,

occlusions can be generated as well, by using a thresholded noise image as mask. All of this gives us easy access to ground truth data for the warp field, correspondence fields and lighting variations.

To uniquely identify images generated with all these parameters, we define their filename as follows, an image with the parameters:
(seed:42, resolution:100, period:12, warp:0.1, light:0.1, grid:0.1, noise:0.1, occlusions:0.3)
would be named "gen-S42-R100-P12-W.1-L.1-G.1-N.1-O.3".



(a) gen-S42-R400-P6-W.1-L.1-G.1-N.3-O.4 (b) gen-S33-R400-P9-W05-L9-G0-N1-O0

Figure 4.1: A few examples of generated images, with overlaid correspondence fields

4.2 Evaluation procedure

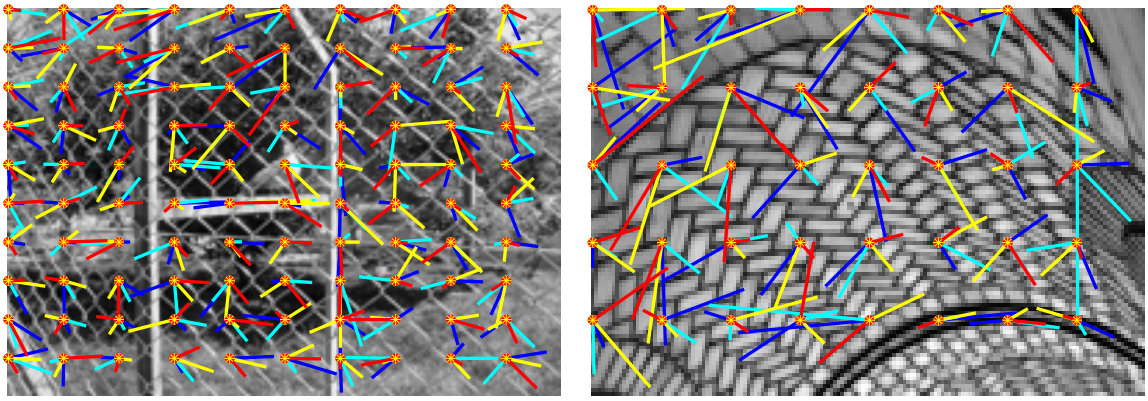
We first test each type of deformation (spatial, light, noise) independently using generated textures, and find the deformation level at which the candidate algorithm succeeds to find the correspondence field in less than half of the attempts. This gives a rough indication of how sensitive the algorithm is to these deformations. We also try several deformation combinations, as the candidate algorithm may be robust against one deformation at a time, but not multiple simultaneously.

As several possible correspondence fields are similar, we simply test if the period of the vectors and the texel area is the same to decide if it matches the ground truth.

Besides testing using the generated textures, we also use several real-world images, sourced from the CMU Near-Regular Texture Database [NRTdb].

4.3 Results

A few detected vector fields can be seen in Figures 4.2a and 4.2b. More processing is needed to derive true correspondence fields from this. Figure 4.2b does show some potential for competitive results with more processing, as the state-of-the-art method tends to find a too small texel with this image. However, not much can be said without actually doing this processing.



(a) (b)

Figure 4.2: Raw vector fields found for a few images, best seen in color

4.4 Performance

For an honest performance comparison, implementations of both methods should be run on the same machine. Unfortunately, the implementation of Park *et al* requires a 64-bit MATLAB version, which was not available. Instead, running times published in their paper are used directly, so this performance comparison serves as a rough indication only. For their Mean-Shift Belief Propagation (MSBP) results, an Intel dual core T7500 @ 2.2 GHz with 3,070 MB memory is specified; it is assumed this same machine was used to obtain the overall running times. Our implementation will be running on an Intel Core2 duo T7200 @ 2.0 GHz with 2,048 MB memory; hopefully these machines are similar enough that running time comparisons are relevant when scaling clock speeds.

In the Park *et al* paper, average running times in minutes are specified for three texture sets D1, D2 and D3:
D1: $4.48 + -2.47$
D2: $4.91 + -2.98$
D3: $5.89 + -4.30$
with an overall average of $D : 5.09 + -3.57$.

Roughly speaking, we can say that their running time averages 5 minutes. Unfortunately, the exact test set used in their paper seems to be no longer available, however most of the textures can be found in the Near-regular Texture database, under "Near-regular Texture Test Set for Lattice Detection". As the resolution of most larger images there is 800x600, we will use images of that size.

Performance has been tested with an image of size 800x600; this took 554 seconds running at 1330 MHz (single-threaded), which would be 335 seconds at 2.2 GHz. As our algorithm's speed depends on resolution alone (not on image content), this single measurement suggests that the used MATLAB implementation appears to perform at a speed approaching that of the state-of-the-art method. It should be noted though that their 5 minute average times are from an early implementation; it is reasonable to assume that their currently available 64-bit version has been optimized and would perform faster.

It is interesting to see what kind of speed-up a native implementation of our method would achieve.

Chapter 5

Conclusion

We have introduced an algorithm for texture regularity detection, by extending and improving upon RSRE. Improvements include a different set of parameters, which the quality of the result depends much less on. Furthermore our new method is more resilient to spatial deformations, while still being able to deal with brightness and contrast variations. We have discussed ways to remove these deformations as well, once the vector field is found.

A MATLAB implementation achieves running times comparable with the state-of-the-art method; an efficient native implementation could have the potential of greatly improving on these times. Preliminary results suggest that there is potential for competitive quality levels, but it requires more processing than we have described.

5.1 Future work

A native implementation of our method should be made; it is likely that an interesting performance gain could be obtained over the MATLAB implementation used for performance measurements.

The current implementation of our method uses a lot of candidate vectors, which costs performance. It may be worthwhile to investigate whether less candidates can be used, while still giving a similar level of quality. Perhaps a hierarchical approach could be investigated as well, as has recently been done for 3-Dimensional Recursive Search (3DRS) [HBvdV+11].

The un-warping operation described in Section 3.7 makes use of Least Squares to assign a suitable set of warp vectors to a block of unknowns; this block's size depends on the texel size, which means for large texel sizes, the running time might be longer than is acceptable. A faster method could be investigated for this application.

Texture regularity detection can also be applied to movies, which should be easier from the second frame onwards as the correspondence fields found for the first frame can be used as

initialization. An interesting application could be to remove temporally changing spatial distortions when filming some regular pattern. For instance, a movie of a swimming pool floor with regular floor tiles, seen through turbulent waves. The regular pattern of the floor's tiles could be used to derive the deformation field, which could enable removal of the wave's distortion and give a clear view of the bottom. The same approach could be applied to eliminate heat haze against a regular background, for instance.

Appendix A

Applications

In this Appendix several interesting applications of texture regularity detection are shortly discussed, with details on how our method can be used to implement them.

A.1 Compression

After we have found the correspondence, warp and light fields, and have extracted the texels, it becomes possible to subtract the texel from each texture in the unwarped/normalized image. This makes the textured area much easier to compress, as it will become mostly uniform. However, there is a trade-off: in order to restore the texture, some extra information will have to be transmitted over the compression channel. The resulting image will have to be re-warped, and brightness/contrast needs to be re-applied as well. Fortunately, both the warp field and the brightness field are typically relatively low-frequency, so this extra information should take relatively few bits as it is low in entropy.

A.2 Foreground removal

Sometimes, the foreground in a photo contains some near-regular texture which we do not want. An example of this is a photo made through a fence, where we want to remove the fence. Based on the pixel confidence values we have, it should be relatively easy to decide which pixels are candidates for removal: those pixels which have a high confidence, are likely part of a repeating texture element. Removed pixels can be replaced by pixel values based on the surrounding pixels.

A.3 Shape reconstruction

The warp field that we have found, contains some cues about the underlying object's geometry. One of the most common examples of this is a flat plane which has undergone perspective distortion. It is possible to derive the orientation of the plane relative to the camera from this, as is done in Section A.3.2.

A.3.1 Depth estimation

The correspondence fields indicate how texel size changes across a texture cluster. If the textured object's original texture is perfectly regular, this change can be interpreted as changing distance from the camera when both dimensions change, or as changing angle if only one dimension changes. While this does not tell us the absolute camera distance, it can be used to determine relative distances, and with further processing the object's absolute distance could possibly be estimated for each pixel, up to a constant scale factor.

A.3.2 Geolocation

One particular instance of shape reconstruction is automatic geo-tagging, as discussed by Schindler *et al* [SKL+08]. They describe a method that exploits the regularity of buildings in typical city photographs. The texture lattice of a building is found and used to determine the relative camera position. Ambiguities are resolved by performing this step for several buildings in the image. A database containing the positions of several known building façades is used to try and find a good match to the observed building orientations. When found, the database provides the longitude and latitude of the buildings, from which a predicted camera position can be derived, which is then used to geo-tag the image.

Bibliography

[Cro84] Franklin C. Crow, *Summed-area tables for texture mapping*, SIGGRAPH Comput. Graph. **18** (1984), no. 3, 207–212.

[DH08] Chris Damkat and Paul M. Hofman, *Efficient local texture regularity estimation*, SIGGRAPH '08: ACM SIGGRAPH 2008 posters (New York, NY, USA), ACM, 2008, pp. 1–1.

[dHBHO93] G. de Haan, P. W. A. C. Biezen, H. Huijgen, and O. A. Ojo, *True-motion estimation with 3-d recursive search block matching*, Circuits and Systems for Video Technology, IEEE Transactions on **3** (1993), no. 5, 368–379, 388.

[Gib50] J.J. Gibson, *The perception of the visual world*, Houghton Mifflin, 1950.

[HBvdV+11] A. Heinrich, C. Bartels, R.J. van der Vleuten, C.N. Cordes, and G. de Haan, *Optimization of Hierarchical 3DRS Motion Estimators for Picture Rate Conversion*, Selected Topics in Signal Processing, IEEE Journal of **5** (2011), no. 2, 262–274.

[HLEL06] James H. Hays, Marius Leordeanu, Alexei A. Efros, and Yanxi Liu, *Discovering texture regularity as a higher-order correspondence problem*, 9th European Conference on Computer Vision, May 2006.

[Jul62] B. Julesz, *Visual pattern discrimination*, Information Theory, IRE Transactions on **8** (1962), no. 2, 84–92.

[LH05] Marius Leordeanu and Martial Hebert, *A spectral technique for correspondence problems using pairwise constraints*, International Conference of Computer Vision (ICCV), vol. 2, October 2005, pp. 1482 – 1489.

[LM96] Thomas Leung and Jitendra Malik, *Detecting, localizing and grouping repeated scene elements from an image*, In Proc. ECCV, LNCS 1064, Springer-Verlag, 1996, pp. 546–555.

[MBSL99] Jitendra Malik, Serge Belongie, Jianbo Shi, and Thomas Leung, *Textons, contours and regions: Cue integration in image segmentation*, Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2

(Washington, DC, USA), ICCV '99, IEEE Computer Society, 1999, pp. 918–925.

[NRTdb] *CMU Near-Regular Texture Database*, `http://vivid.cse.psu.edu/texturedb/gallery/`, Accessed: 2013-07-07.

[PBCL09] Minwoo Park, Kyle Brocklehurst, Robert T. Collins, and Yanxi Liu, *Deformed lattice detection in real-world images using mean-shift belief propagation*, IEEE Transactions on Pattern Analysis and Machine Intelligence **31** (2009), no. 10, 1804–1816.

[PLC08] Minwoo Park, Yanxi Liu, and Robert T. Collins, *Efficient mean shift belief propagation for vision tracking*, 2012 IEEE Conference on Computer Vision and Pattern Recognition **0** (2008), 1–8.

[Sch78] Doris Schattschneider, *The plane symmetry groups: Their recognition and notation*, The American Mathematical Monthly **85** (1978), no. 6, 439–450.

[SKL+08] G. Schindler, P. Krishnamurthy, R. Lublinerman, Y.X. Liu, and F. Dellaert, *Detecting and matching repeated patterns for automatic geo-tagging in urban environments*, CVPR08, 2008, pp. 1–7.

[ST94] J. Shi and C. Tomasi, *Good features to track*, Computer Vision and Pattern Recognition, 1994. Proceedings CVPR '94., 1994 IEEE Computer Society Conference on, 1994, pp. 593–600.

[YFW03] Jonathan S. Yedidia, William T. Freeman, and Yair Weiss, *Understanding belief propagation and its generalizations*, Exploring artificial intelligence in the new millennium (Gerhard Lakemeyer and Bernhard Nebel, eds.), Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003, pp. 239–269.