2013 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS)
November 3-7, 2013. Tokyo, Japan

Automatic Creation and Application of Texture Patterns to 3D Polygon Maps

Kim Oliver Rinnewitz[1], Thomas Wiemann[1], Kai Lingemann[2] and Joachim Hertzberg[1,2]

Abstract— **Textured polygon meshes are becoming more and more important for robotic applications. In this paper we present an approach to automatically extract textures from colored 3D point cloud data and apply them to a polygonal reconstruction of the scene. The extracted textures are analyzed for existing patters and reused if several instances appear. Emphasis of this work is on minimizing the number of used pixels while maintaining a realistic impression of the scanned environment.**

I. INTRODUCTION

Three dimensional environment representations play an important role in modern robotic applications. They can be used as maps for localization and obstacle avoidance as well as environment models in robotic simulators or for visualization in HRI contexts, e.g., tele operation. Most sensors like 3D laser scanners and RGB-D cameras deliver point clouds as raw data. For mapping purposes, when building high resolution maps of large environments, the huge amount of data points poses a problem.

A common approach to overcome the drawbacks of raw point cloud data is to compute polygonal surface representations of the scanned environments. Polygonal maps are compact, thus memory efficient, and, being continuous surface descriptions, offer a way to handle the discretization problem of point cloud data. Automatic reconstruction of polygonal surface representations from point cloud data is an active field of research in computer graphics. Modern 3D sensors are equipped with cameras to produce colored point clouds, so integration of the gathered color information into computed polygon maps is of high interest. The standard procedure for this integration is to generate bitmap textures from the input data and project them onto the polygonal surfaces. Such bitmap textures are the standard representation in computer graphics. For robotic applications they offer opportunities to enhance the stored environment information: Besides photo-realistic rendering, visual features from the generated textures can be used to solve standard problems like visual SLAM or localization. Since features in the textures are inherently associated with coordinates on the polygonal surfaces, they can be located easily in 3D space.

The main problem when using textures in surface reconstruction from scanned robotic environments is that the maps can become quite large, and therefore large texture maps have to be generated. This is problematic since even on state of the art graphics hardware, the maximum allowed size of textures is quite limited. This problem can be solved by reusing small textures with repeating patterns to approximate the real environment. A typical example is a wall, textured by a repeated image of a few bricks. Another way of reducing the amount of texture data is to find instances of objects with the same, not necessarily patterned, texture, and to reuse previously generated textures. In this paper we present an approach to extract repeating texture patterns from color or intensity values in 3D point cloud data and project them onto the corresponding polygonal reconstruction. The used software is integrated into the Open Source Las Vegas Surface Reconstruction Toolkit (LVR) [6], [18] and supports several commonly used exchange formats.

The remainder of this paper is organized as follows: Section II presents the state of the art in textured surface reconstruction from point cloud data. Section III presents the technical details of the developed texture assignment techniques which are evaluated in Section IV.

II. RELATED WORK

Automatic construction of meshes from point cloud data has received much interest in computer graphics. The de facto standard method to generate triangle meshes of arbitrary environment is the Marching Cubes algorithm [9], which comes in a number of variants. [12] provides a comprehensive review. For reconstruction of closed objects methods based on Delaunay triangulations exists [1], [2]. Another successful approach for closed geometries is the Poisson Reconstruction [4].

Kinect Fusion [3] implements a GPU based Marching Cubes variant that uses the inherent structure of the RGB-D camera to create meshes of Kinect data in real time. It has been extended recently to handle larger environments [16]. The PCL implementation of Kinect Fusion [15] can extract textures for the reconstructed surfaces, but is not capable of reusing patterns. A method to extract textured polygon meshes from point cloud data, including high resolution laser scans with RGB information, is presented in [17]. The generated meshes were successfully used as environment representation in Gazebo. This paper extends that work to reduce the amount of generated pixel data by pattern analysis and image matching.

Texture analysis has been an extensive field of research since the late 1950's. A commonly used method to determine the similarity of the images is cross correlation [5]. Recently pattern analysis has become of interest for large image data

[1]Knowledge Based Systems Group, Osnabrück University, Albrechtstr. 28, 49076 Osnabrück, Germany {krinnewitz, twiemann}@uni-osnabrueck.de

[2]DFKI Robotics Innovation Center, Osnabrück branch, Albrechtstr. 28, 49076 Osnabrück, Germany. first_name.last_name@dfki.de

DFKI's Osnabrück branch is sponsored by the State of Niedersachsen.

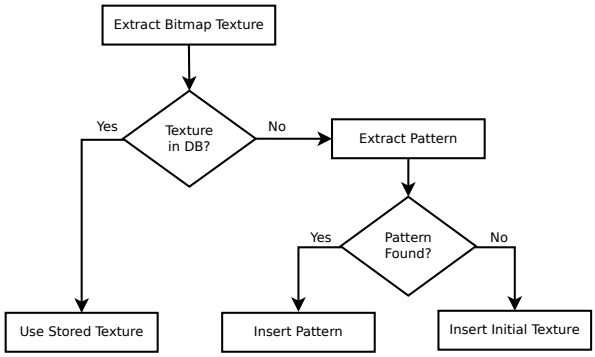


Fig. 1. Texture generation and storage management.

bases. [10] presents an approach to match existing patterns to large image sets using Gabor Wavelets. A survey of different image matching approaches is given in [8]. An approach to extract pattern from images was presented in [14]. Our work extends this approach to make it more robust via testing the extracted pattern textures against the initial bitmaps.

III. TEXTURE GENERATION AND ASSIGNMENT

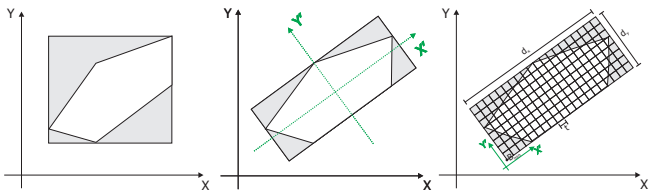
This section describes the model texturization process. Starting with a method to manage textures during the reconstruction process in LVR [6], followed by how the bitmaps are computed, it concludes with a discussion of the problems of pattern recognition and identification of reuseable textures. The textured areas in the reconstructed meshes are planar polygons in the environment as described in [18].

A. Texture Management

The structure of our texturing software is shown in Fig. 1. In the first step an initial bitmap for a polygon is computed. This image is compared with the images stored in a texture database for the current model. The comparison is partially done using feature descriptors as described in Section III-C. To speed up the matching process, we store the image features together with the raw bitmap data in the database. If a matching texture is found, it is applied to the current polygon. Otherwise the bitmap is analyzed to detect patterns (see Section III-C). If a periodic pattern is detected, the corresponding pattern bitmap is extracted and added to the database, otherwise the initial pattern is stored. If different data sets from the same environment are reconstructed, the produced databases can be reused. In the same way, precomputed standard textures can be applied to the reconstruction.

B. Initial Texture Generation

Texture generation is done by putting a rectangular grid of fixed cell size t over each polygon. The voxel size of the grid determines the resolution of the texture image. For each cell in the grid, a color value is calculated by averaging the colors of the k nearest points in the given point cloud. To keep the textures small, it is necessary to find a grid alignment that maximizes the used area (cf. Fig. 2). In the presented example, an axis aligned pixel grid would contain a large amount of unused pixels (gray) while an image in



(a) Axis parallel (b) local system (c) texture pixels

Fig. 2. Finding the right coordinate system for texture generation. In the presented case, an axis aligned pixel grid (a) would result in a lot of unused space (gray). An aligned bounding box would reduce this space (b). The actual pixel map as calculated as shown in (c).

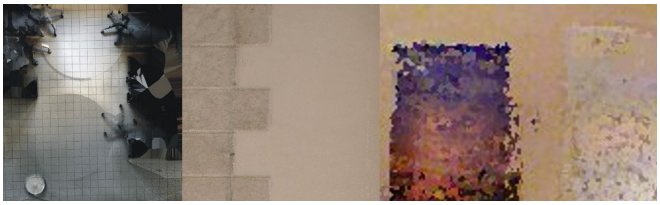


Fig. 3. Examples for generated bitmaps from different data sets: High density laser scanner data (left and middle) and Kinect (right).

the green coordinate system would be significantly smaller (only 70% of the axis aligned version). To determine the best alignment we compute the Principle Component Analysis (PCA) of the polygon vertices. The first two eigenvectors of the result deliver a good estimation for the alignment of the polygon in the global coordinate system, thus we define the pixel matrix by determining the bounding box of the polygon in this local system defined by X' and Y'. The achievable texture quality strongly depends on the quality of the input data: Low point densities will cause blurry textures, while dense point cloud data will allow to choose a high resolution for texture generation.

Some examples for automatically produced textures from different kinds of point cloud data are shown in Fig. 3. The left and middle image show textures generated from high resolution laser scanner data (Leica and Faro respectively). The left image (972 $\times$ 1065 pixels) shows the complete texture for the ground floor of an office with a regular tiling. The picture in the middle (303 $\times$ 473 pixels) shows regular bricks on a wall. The aim of the pattern extraction step is to identify this kind of regularities, generate appropriate pattern textures and thus reduce the size of the used textures. The right image displays a texture from a Kinect reconstruction showing posters on a wall without regularities. These examples exemplify a unsurprising fact: The quality of the initial textures strongly depends on the quality of the input data, i.e., scans with high point density allow to generate textures with higher resolution.

C. Texture Matching

The aim of the texture matching step is to identify already stored textures that approximate the surface of the currently considered polygon well enough so that they can be reused. We tested several approaches from image processing to solve this problem: Color histogram based correlation, cross correlation and feature based matching. Histogram

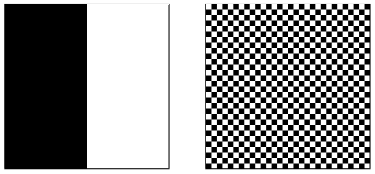


Fig. 4. Pixel distributions and color histograms. The two images on the left will produce the same simple color histogram but different Color Coherent Vector histograms.

based matching is fast, but has a high rate of false positive detections. Cross correlation delivers good results and can be computed fast in frequency space, but it is very sensible to the right separation threshold. Feature based matching delivers the best results, but is comparatively slow. To find a suitable compromise between accuracy and computation speed we choose the following approach: (1) filter out candidates that definitely do not match using Color Coherence Vectors (CCV) [13], exploiting the fact that this method detects not-matching images safely. (2) Instances of already detected patterns within the textures are detected using cross correlation. (3) The remaining images are matched using SURF features and SIFT descriptors. Details of the three steps are presented in the following paragraphs.

1) Color Coherence Matching (CCM): Histogram based matching counts for each color channel the number of occurrences of a given value $h(f)$, $f \in [0,255]$. The matching score d between two images is defined naively as

$$d = \sum_f |h_1^r(f) - h_2^r(f)| + \sum_f |h_1^g(f) - h_2^g(f)| + \sum_f |h_1^b(f) - h_2^b(f)|.$$

One problem when using histogram based approaches to match images is that quite different distributions of pixel values can result in identical histograms (cf. Fig. 4). An approach for solving this problem is to consider the connectivity in images as well, storing for each color histogram entry (r,g,b) in the variable α_f how often a color is present in large blobs with size $\geq s$, and likewise in β_f the number of occurences in blobs of size $< s$, cf. [13]. Blobs are detected via connected component labeling after Gaussian blurring to reduce the number of clusters. The matching score is then refined as:

$$\begin{aligned} d = &\sum_f \left(|\alpha_{f,1}^r/N_1 - \alpha_{f.2}^r/N_2| + |\beta_{f,1}^r/N_1 - \beta_{f,2}^r/N_2|\right) \\ &+ \sum_f \left(|\alpha_{f,1}^g/N_1 - \alpha_{f,2}^g/N_2| + |\beta_{f,1}^g/N_1 - \beta_{f,2}^g/N_2|\right) \\ &+ \sum_f \left(|\alpha_{f,1}^b/N_1 - \alpha_{f,2}^b/N_2| + |\beta_{f,1}^b/N_1 - \beta_{f,2}^b/N_2|\right), \end{aligned}$$

for two images of size N_1 and N_2, and their respective values of α and β for each r,g,b value. This normalized formulation further enables us to compare two images of different size. For the given example a threshold of $s = 1$, for example, leads to $\alpha = 150$ and $\beta = 0$ for the black and white pixels in the left image, and $\alpha = 0, \beta = 150$ for both colors in the right one, thus providing a means for separation. To speed up the matching, we store the histogram vectors for each picture in the database after a new texture was inserted.

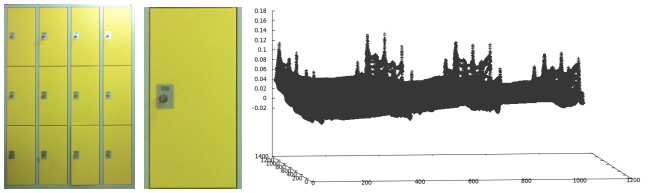


Fig. 5. Polygon texture (left) with corresponding pattern texture (middle) and cross correlation (right)

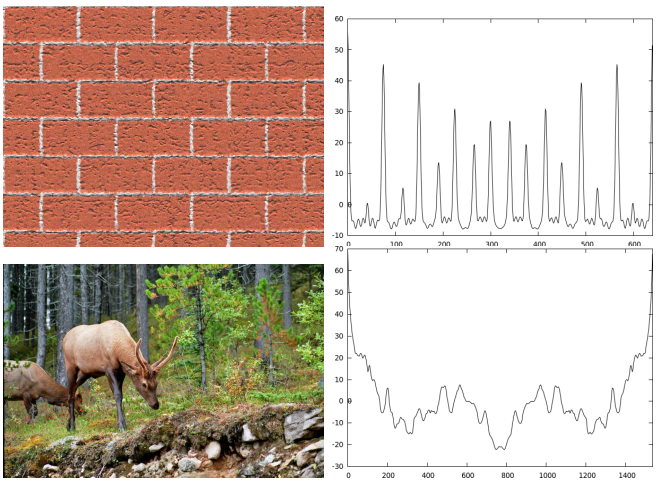


Fig. 6. Summed horizontal correlation values for a periodic (top) and non periodic image (bottom). The vertical curves are not shown, since they are similar in the presented examples.

2) Pattern Texture Assignment via Cross Correlation: After images that are obviously non-matching are rejected based on CCM, we test if an already detected and archived pattern is present in the current texture bitmap. Patterns are detected using cross correlation, defined as

$$c(d) = \frac{\sum_i (I_1(i) - \bar{I}_1) \cdot (I_2(i-d) - \bar{I}_2)}{\sqrt{\sum_i (I_1(i) - \bar{I}_1)^2} \cdot \sqrt{\sum_i (I_2(i-d) - \bar{I}_2)^2}}$$

for two gray scale images I_1 and I_2 with mean gray values of $\bar{I}_1$ and $\bar{I}_2$. The idea is to move the pattern candidate with different offsets d over the current data image. If two images match well at a certain position, the products create high values, which sum up to a high correlation value. The denominator ensures normalization. Thus, if a pattern matches well in the data image, it will create a high correlation value, as shown in Fig. 5. The computation of the cross correlation has to be evaluated for every picture in the database and cannot be precomputed. The naive implementation is expensive. Fortunately, performing the operation in frequency space via Fast Fourier Transform is quick, so there is no critical performance issue with this approach.

3) Feature Based Matching: In addition to the cross correlation test, we integrated feature based matching using the SURF implementation of OpenCV. Descriptor matching is done by an approximate nearest neighbor search using FLANN [11]. Feature based matching works well under constant light conditions and in rich-textured environments with many detectable features. Consequently it will fail in featureless areas. Thus, the feature matching step mainly

TABLE I

EVALUATION OF SUMMED AUTO CORRELATIONS FOR A SET OF IMAGES.

	Periodic				Non-Periodic			
Img.	$\sigma_{N\rho_x(x)}$	$\sigma_{N\rho_y(y)}$	π_x	π_y	$\sigma_{N\rho_x(x)}$	$\sigma_{N\rho_y(y)}$	π_x	π_y
1	0.68	0.90	62	21	0.13	0.94	4	4
2	1.51	0.79	12	10	0.20	0.96	5	4
3	0.55	0.78	18	13	0.75	1.23	55	22
4	0.75	1.23	55	22	1.74	1.33	8	9
5	1.74	1.33	8	9	0.13	0.76	5	14
6	1.28	1.59	26	15	1.17	1.26	47	12

enhances the matching score for reuse of distinctive textures. A high matching score of cross correlation and feature analysis indicates a high probability of a positive match.

D. Pattern Extraction

To extract patterns from a given bitmap we have to answer two questions: Is there a pattern in this image, and if so, what is the optimal cut to create a texture that reproduces a the pattern correctly? For pattern detection we use auto correlation. The auto correlation $\rho(d)$ of an gray scale image I with $N_x \times N_y = N$ pixels is basically a convolution of the image with itself:

$$\rho(d) = \frac{\sum_{i=0}^{N-1}(I(i)-\bar{I})\cdot(I(i-d)-\bar{I})}{\sum_{i=0}^{N-1}(I(i)-\bar{I})^2}$$

with $d = y \cdot N_x + x$ for a pixel (x,y) in I. In the remainder of this paper we will refer to the auto correlation at a given pixel position as $\rho(x,y)$. To detect patterns in x and y direction we examine the summed correlation values in ρ_x and ρ_y:

$$\rho_x(x) = \sum_{y=0}^{N_y-1}\rho(x,y) \qquad \text{and} \qquad \rho_y(y) = \sum_{x=0}^{N_x-1}\rho(x,y).$$

Two examples of summed correlation values for a periodic and non-periodic image are shown in Fig. 6. As one can see, the functions are symmetric and show peaks at high correlation points. As expected, the periodic image shows more and higher peaks than the non-periodic image. To decide whether an image is periodic, [14] proposes to evaluate the standard deviations $\sigma_{N\rho_x(x)}$ and $\sigma_{N\rho_y(y)}$ of the peak distances weighted by size of the image and number of peaks π_x, π_y in the corresponding direction

$$\sigma_{N\rho_x(x)} = \frac{\sigma_{\rho_x(x)}}{N_x/\pi_x} \qquad \text{and} \qquad \sigma_{N\rho_y(y)} = \frac{\sigma_{\rho_y(y)}}{N_y/\pi_y}$$

and to use a fixed threshold ε to decide if the bitmap is periodic ($\sigma_{N\rho_x(x)}$ and $\sigma_{N\rho_y(y)} > \varepsilon$), semi-periodic ($\sigma_{N\rho_x(x)}$ or $\sigma_{N\rho_y(y)} > \varepsilon$) or non-periodic (both values below ε). The main problem is to find a suitable decision threshold. We have evaluated this approach for a set of images as shown in Table I. The used images can be found on the website [7].

As one can see, there is no obvious decision value based on the analysis of the sample data. There are examples for high and low correlations in both sets. Choosing a fixed value, e.g., 0.8 (the detected values lie in $[0,1.6]$) we would only classify 4 periodic textures correctly while producing 5 false positives for non-periodic cases in the 6 samples.

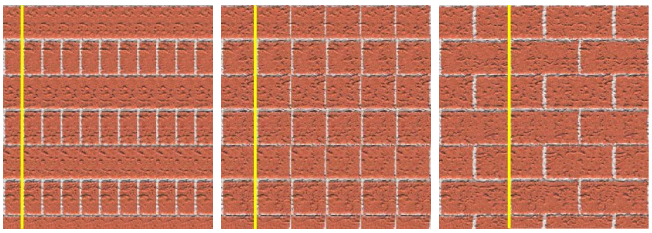


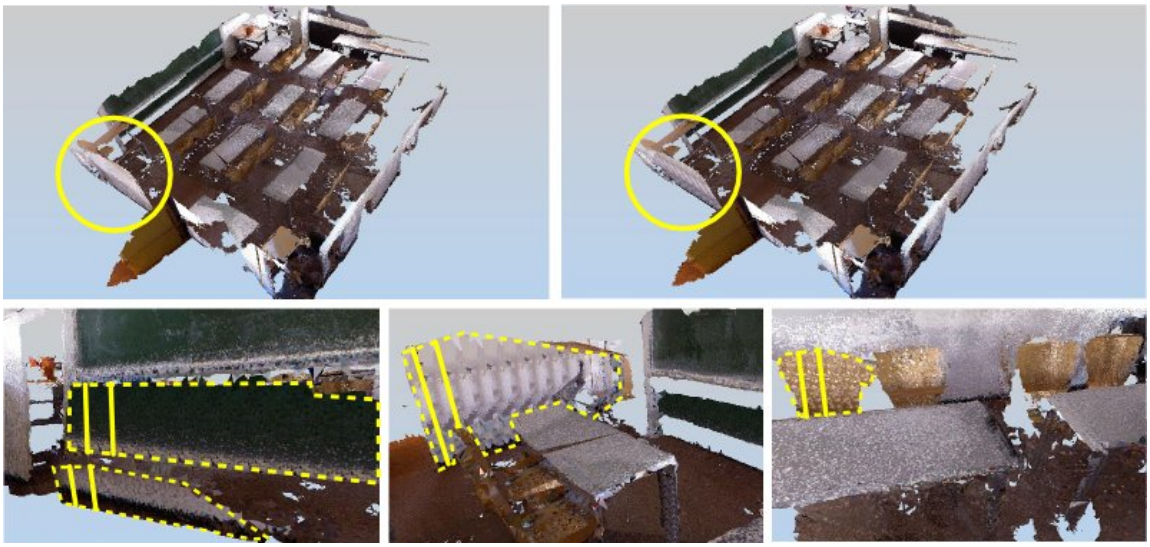
Fig. 7. Pattern extraction using different peak positions to determine the pattern image.

Another possible decision value could be the number of peaks. Although generally the number of peaks is most times significantly higher in periodic images, there are two non-periodic images that show a high number of peaks as well.

To ensure stable detection of patterns, we therefore use another approach; If an image contains more that $\pi_{\min}$ peaks, assume that there is a pattern in the image. For each significant peak, i.e., relative distance and amplitude to the next peak are high, we extract the pattern that is defined by the image coordinates of the detected peaks. An example for possible sub image cuts based on peak positions is given in Fig. 7. The image shows the extracted pattern for different peak offsets (part until the yellow line) and the repeated rendering of the pattern. The example in Fig. 6 (top) has two overlapping series of peaks at $\pi_x = 75, 150, 225, 300$ and $\pi_x = 42, 116, 191, 266$, corresponding to the two brick layers. Choosing the first one from the first series and the first one from the second series at 116 would result in the first image of Fig. 7. This pattern is obviously too narrow. If combine it with 225, the chosen segment is again too small. Combining 75 and 225, ignoring the lower peaks, delivers a suitable result. This example shows that in general, it is hard to find the correct image patch even in simple cases. To identify the best matching pattern, we cross correlate each candidate with the original image and choose the one with the best matching score. The main advantage is that cross correlation favors small patterns, which is desired here.

After finding the patterns we have to determine the transformation of the pattern into the model coordinate system to determine correct texture coordinates for rendering. The needed transformation consists of rotation, translation and scale. Scaling is interesting for reusing textures in cases like stone patterns with different brick size, etc. For the moment we concentrate on finding the correct translation and rotation. The translation is determined by matching the features of the model to their corresponding features from the generated textures, defined by a nearest neighbor search. Next, all statistically "'good"' features (i.e., with a vector distance smaller than two times the minimal distance of all features) are selected. In a RANSAC-like fashion, sets of 3 features are chosen randomly and used to calculate a transformation; the output is the best supported and most stable transformation. The correct scale of a texture is defined by the relative texture coordinates within the bounding box of an planar polygon, i.e., each vertex is mapped to a relative position (u,v) in the image plane with $u,v \in [0,1]$.

(a) Experimental results for the church data set.



(b) Experimental results for the Kinect data set.

Fig. 8. Comparison of the initial and optimized reconstruction (first rows) for the evaluated real life data sets. Detailed views of extracted patterns are given in the second rows. High resolution images can be found at the website [7].

IV. EVALUATION

To evaluate the quality of the generated texture patterns quantitatively, an appropriate metric is needed; however, we know of no such established method. Thus, for a qualitative analysis, we evaluated our approach in three representative data sets with different characteristics. The first set is an artificially created example to demonstrate the functionality of our framework. The second is a high resolution colored laser scan of a church in a city scene (17.5 million points), and the last presented example is a reconstruction from a scene scanned with a Kinect mounted on a mobile robot and registered with 6D SLAM (10 million points).

Our artificial data set and the created reconstruction is shown in Fig 9. On the left is the colored point cloud, with the generated reconstruction on the right. The scene consists of 6 planes with different textures. As one can see, the blackboard texture is matched correctly but the transformation was determined wrongly, due to a lack of feature machtes that are statistically good enough. Furthermore, our software extracted a pattern for the rips of the heater. The detail of the mounted thermostat is lost when the pattern texture is applied to the reconstruction, but the effect on the general visual impression is negligible. The last used texture was a regular pavement. Again, our software was able the extract a minimal pattern consisting of two cobblestones in width and one in height. In this example, the software extracted all present patterns and detected the blackboard texture correctly, demonstrating the capability of texture optimization for distinct textures.

The quantitatively evaluated data sets were real life applications with point clouds of different quality (Riegl laser

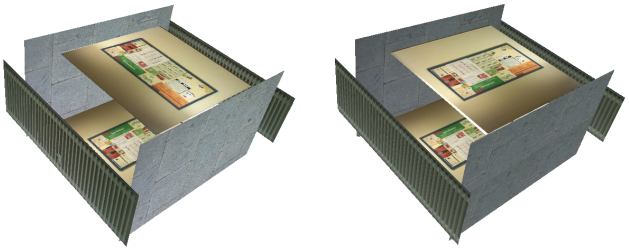


Fig. 9. Texture matching and pattern extraction in a synthetic data set.

scanner vs. Kinect). Screenshots from renderings of the reconstructions are given in Fig 8 for the Riegl data set (a) and the Kinect point cloud (b). The first rows of the two subfigures compare the visual quality of the reconstructions with and without pattern extraction and matching. Areas with substantial visual differences are marked by yellow circles. The second rows show detail shots from the reconstructions where extracted patterns were used. The borders of the patterns are indicated with solid lines, the area where the repeated texture is used is marked dashed. The overall visual appearance is very similar, although unpropitious pattern extractions and wrong matches appear (marked with yellow circles). Generally the pattern extraction works well. Especially the extraction of the pattern for the houses in the city data set and the blackboards in the Kinect data sets reduce the amount of needed pixel data significantly (cf. Table II). The most problematic parameter is the peak threshold for pattern extraction. Our process is sensible to oversegmentation, so the minimum peak value has to be set high enough.

In the large data sets the CCV initially filters out an average of 78% of the textures in the church data set and 69% in the Kinect data set for each new frame, thus reducing the number of considered picture-to-picture matches significantly. The total running time, including surface reconstruction and texture optimization, was 2:28 min. for the church scene and 1:32 min. for the Kinect data set.

Table II presents some more detailed figures about the performed experiments. Given are the number of detected planes, the number of generated and matched patterns and single textures together with the number of positive matches and the number of obviously wrongly correlated textures. The church data set shows a lot of extracted patterns. Due to the high quality threshold for matching that was needed to suppress false positive detections on the roofs, the matching score for reusing patterns is small. From non-pattern textures 11 textures were reused, mainly on the sides of the tower. The Kinect data sets on the other hand features a high number of reused textures, both pattern and non-pattern. This is due to multiple instance of chairs and desks that can easily be reused in the class room example. The number of false positive detection is relatively small in both data sets (about 3% in the Kinect data set and 2.5% in the laser data). The main problem is that false positives significantly reduce the visual quality of the reconstruction, therefore the relevant parameters for matching have to be chosen with care.

TABLE II

RESULTS FOR PATTERN EXTRACTION AND TEXTURE REUSE.

		Non-Patterns		Patterns			Compression
Data set	# Planes	Gen.	Matched	Gen.	Matched	False	(before/after)
Artificial	6	1	1	2	2	0	0.8 / 0.3 MB
Kinect	90	12	55	10	13	3	12.1 / 2.3 MB
Church	349	171	11	167	0	9	28.0 / 2.9 MB

V. CONCLUSION AND FUTURE WORK

This paper presented an approach to automatically detect and extract patterns in reconstructions from 3D point cloud data. The extracted textures can be reused to reduce the amount of pixel data needed for textured rendering. The generated maps are small and compact, and can be used in robotic applications like scene visualization. Future work will concentrate on detecting regularities on surfaces with overlaying objects, like posters on stone walls. The present results show that methods from pattern analysis can be successfully used. False positive matches may be reduced by including more sophisticated approaches from theoretical pattern analysis than the ones presented here.

REFERENCES

[1] N. Amenta, S. Choi, and R. K. Kolluri. The power crust. In *Proc. 6th ACM Symp. on Solid Modeling and Appl.*, pages 249–266, 2001.

[2] O. Devillers. The Delaunay Hierarchy. *Intl. J. Foundations of Computer Science*, 13, 2002.

[3] S. Izadi, R. A. Newcombe, D. Kim, O. Hilliges, and et al. Kinectfusion: real-time dynamic 3d surface reconstruction and interaction. In *ACM SIGGRAPH 2011 Talks*. ACM, 2011.

[4] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *Proc. SGP '06*, pages 61–70. Eurographics Association, 2006.

[5] R. D. Keane and R. J. Adrian. Theory of cross-correlation analysis of piv images. *Applied scientific research*, 49(3):191–215, 1992.

[6] Las Vegas Surface Reconstruction. `http://www.las-vegas.uni-osnabrueck.de`.

[7] Las Vegas Surface Reconstruction. `http://www.las-vegas.uni-osnabrueck.de/patterns.html`.

[8] Y. Liu, D. Zhang, G. Lu, and W. Ma. A survey of content-based image retrieval with high-level semantics. *Pattern Recognition*, 40(1):262–282, 2007.

[9] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *ACM SIGGRAPH*, 1987.

[10] Bangalore S Manjunath and Wei-Ying Ma. Texture features for browsing and retrieval of image data. *IEEE Trans on Pattern Analysis and Machine Intelligence*, 18(8):837–842, 1996.

[11] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *Proc. VISSAPP'09)*, pages 331–340. INSTICC Press, 2009.

[12] T Newman and H Yi. A survey of the marching cubes algorithm. *Computers & Graphics*, 30(5), 2006.

[13] G. Pass, R. Zabih, and J. Miller. Comparing images using color coherence vectors. In *Proc. of the 4th ACM intl. conf. on Multimedia*, pages 65–73. ACM, 1997.

[14] M. Petrou and P. García-Sevilla. *Image processing - dealing with texture*. Wiley, 2006.

[15] R. B. Rusu and S. Cousins. 3D is here: Point Cloud Library (PCL). In *Proc. ICRA 2011*, 2011.

[16] T. Whelan, M. Kaess, M.F. Fallon, H. Johannsson, J.J. Leonard, and J.B. McDonald. Kintinuous: Spatially extended KinectFusion. In *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, Sydney, Australia, Jul 2012.

[17] T. Wiemann, K. Lingemann, and J. Hertzberg. Automatic map creation for environment modelling in robotic simulators. In *Proc. 27th Conf. on Modelling and Simulation (ECMS 2013)*, May 2013 (to appear).

[18] T. Wiemann, K. Lingemann, A. Nüchter, and J. Hertzberg. A toolkit for automatic generation of polygonal maps – Las Vegas Reconstruction. In *Proc. ROBOTIK*, München, 2012.