SIFT-Realistic Rendering

Dominik Sibbing*, Torsten Sattler*, Bastian Leibe†, Leif Kobbelt*
*Computer Graphics Group †Computer Vision Group
RWTH Aachen University

{sibbing@cs, tsattler@cs, leibe@vision, kobbelt@cs}.rwth-aachen.de

***Abstract*—3D localization approaches establish correspondences between points in a query image and a 3D point cloud reconstruction of the environment. Traditionally, the database models are created from photographs using Structure-from-Motion (SfM) techniques, which requires large collections of densely sampled images. In this paper, we address the question how point cloud data from terrestrial laser scanners can be used instead to significantly reduce the data collection effort and enable more scalable localization. The key change here is that, in contrast to SfM points, laser-scanned 3D points are not automatically associated with local image features that could be matched to query image features. In order to make this data usable for image-based localization, we explore how point cloud rendering techniques can be leveraged to create virtual views from which database features can be extracted that match real image-based features as closely as possible. We propose different rendering techniques for this task, experimentally quantify how they affect feature repeatability, and demonstrate their benefit for image-based localization.**

***Keywords*-Image generation;Image matching**

I. Introduction

In this paper we address the problem of image-based 3D localization, *i.e.*, the estimation of the camera pose given a query image. Current state-of-the-art approaches for this task [8], [12], [22] represent the scene by a 3D point cloud obtained by Structure-from-Motion (SfM) techniques [23] from a set of densely sampled input images [20]. Such approaches have been demonstrated to work well for collections of isolated tourist sites [8], [12], [22]. However, scaling them to an entire city would require the collection of millions of photographs, which entails substantial effort.

On the other hand, there are a number of current activities to create dense and highly accurate point clouds of urban environments using terrestrial or car-mounted laser scanners. Examples for such endeavors include NAVTEQ True [16] and the Google self-driving car project [24]. Besides capturing pure point data, laser scanners often store a color value for each point and, when combined with camera data, these color values could in principal be augmented with an image gradient or even an entire textured patch. Since laser scanners measure distances with a high precision, it is possible and in practical scenarios common to place the laser scanner only at few sample locations, leading to much larger intervals between the scan positions. This is not a problem for creating visually pleasing reconstructions — the resulting point clouds are still sufficiently dense — but such large intervals are a problem for image based localization, since the local feature extractors used for establishing correspondences between query images and the 3D model (*e.g.*, SIFT [13]) are only tolerant to small perspective changes. In order to obtain good performance for image based localization, a densely sampled image database (capturing the variability of descriptor appearance across viewpoints) is required [8]. The question we would like to address in this paper is thus how to make the point cloud information usable for large scale image based localization.

Inspired by the work of Kaneva et al. [9], who showed that descriptors extracted from photo-realistic renderings are similar to those extracted from real photos, we investigate the problem what rendering techniques are best suited for converting laser scanned point clouds captured at sparsely sampled locations into a large database of densely sampled virtual images, providing local features for image-based localization. We explore a range of different rendering techniques targeted at application scenarios where different levels of information are available for the point cloud. For the case where only colored 3D points are available, we explore different splat rendering techniques. Since those still leave holes in the rendered images, we propose to extend point rendering by inpainting techniques. On the next level, we investigate how the addition of a single gradient vector per 3D point can be used to improve the preservation of color gradients, the key ingredient of local feature descriptors. For this, we make use of advanced image completion techniques to reconstruct virtual images from a sparse gradient field defined by the projected 3D points and their gradient vectors. Finally, we consider the case where entire image regions are available to be associated to 3D points as textures. We adapt an existing splat rendering technique [4] in order to blend overlapping textured splats to create the virtual image. We experimentally evaluate each of the proposed rendering approaches on realistic 3D point cloud data and compare them to a baseline approach working on sparsely sampled image data. We show what matching and localization performance can be expected in each of the scenarios and derive clear usage guidelines.

The rest of the paper is structured as follows. After discussing related work, Section III, describes some preprocessing steps applied on the point cloud and explains our different approaches to render synthetic views. Section IV evaluates the presented rendering techniques and discusses the obtained results.

II. Related Work

Rendering point clouds. To create the illusion of a closed surface, Rusinkiewicz *et al.* render *splats*, oriented discs in 3D space, instead of simple points [21]. Using hierarchical data structures, they are able to easily render datasets containing several million points in real time. Since every splat has a constant color, *splat rendering* might yield too sharp edges between splats. Botsch *et al.* suggest a rendering framework which efficiently averages colors of overlapping splats in a multi-pass rendering approach [4]. Additionally blending normals enables screen-space lighting of the visible surface. This approach produces visually more pleasant images as it blurs hard edges. This behavior is counter-productive for our use case, as it suppresses the gradient information required by keypoint descriptors and detectors such as SIFT. We extend this splat rendering approach and blend textures of overlapping splats. Note that this is different to the approach proposed by Yang *et al.* [27], which assigns several source textures to one splat and defines a view-depending weighting function to blend between them. In our setting we deal with very large point clouds and assign only one texture to each splat, exploiting the knowledge from which position a point was measured. This reduces the number of texture look ups during rendering and we can simply blend the image intensities and/or image gradients of overlapping splats using a rendering pipeline more related to [4]. A more exhaustive overview of point based rendering techniques can be found in [11], [1].

Alternatively, we could use polygonal surfaces to represent our scenes instead of the original point clouds. However, surface reconstruction algorithms such as Power Crust [2] or Poisson surface reconstruction [10] often produce artifacts in urban environments caused by moving objects, *e.g.*, pedestrians or cars, and translucent objects.

Using synthetic views for image matching. Generating synthetic views of real objects is a common strategy to generate training data. For example, Özuysal *et al.* generate different views of planar patches using affine transformation and use them to learn keypoint detectors [18]. Similarly, affine warping is used to obtain a fully affine invariant version of SIFT [15]. In a related approach, Wu *et al.* exploit information about the 3D structure of the scene to rectify image patches before extracting SIFT descriptors [26]. They show that using such viewpoint normalized descriptors improves the robustness of descriptor matching as it factors out view-dependent changes.

Irschara *et al.* use image retrieval techniques [8] to quickly find a set of database images similar to a query image. Since image retrieval fails if the query image was taken with substantially different viewing conditions than the database images, they generate synthetic images to cover a wider range of viewpoints. To this end, they place virtual cameras in a sparse 3D SfM point cloud and backproject the points into the virtual view. Every visible point is then associated with an existing SIFT descriptor extracted in one of the database images. Combining synthetic views with original photos is shown to improve image retrieval.

Most similar to our approach are the works of Gee & Mayol-Cuevas and Newcombe *et al.* [7], [17]. Newcombe *et al.* generate synthetic views of a densely reconstructed scene and perform robust camera tracking [17]. To estimate the pose of the camera, they thereby find the motion parameters which generate the view that most closely matches the camera image. In a similar approach, Gee & Mayol-Cuevas use regression on synthetic views to re-localize a camera for which tracking failed. Both methods, specifically designed for small scenes, use image intensities to compare the images and require a dense reconstruction of the scene while we try to perform feature matching against synthetic views rendered from a point cloud.

III. Generating Synthetic Views

Generating synthetic views always requires some kind of a 3D model of the object of interest. In our setting we use simple point clouds captured with a Riegl LMS-Z390i laser scanner. We do not convert such point clouds into polygonal representations textured with the photographs, since this involves complicated meshing and parametrization techniques, which likely will fail for point clouds obtained in an urban scene containing moving people, traffic, plants, geometrically complicated facades, reflections etc. We rather only use some simple filter operations and point based rendering techniques to generate the synthetic views needed to compute SIFT features for our localization procedure. Before rendering synthetic views we perform a number of preprocessing steps on the point cloud: First of all we associate an oriented normal to each point and automatically filter outliers based on the distribution of the local neighborhood. Then we estimate a radius for each point, which, together with the normals, enables us to render the point cloud as a closed surface. In order to be able to detect useless views showing a facade from behind, it is necessary to compute oriented normals. Since our point cloud $P = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$ is obtained by merging several individual scans from different positions, we can associate a scan position $\mathbf{s}(i) \in \mathbb{R}^3$ to each point $\mathbf{p}_i \in \mathbb{R}^3$. The local neighborhood $N_r(i) = \{j \mid \|\mathbf{p}_j - \mathbf{p}_i\| < r\}$ contains all points $\mathbf{p}_j \in P$ which are close to $\mathbf{p}_i$ (we select $r = 0.2$ meters). We compute the unoriented normal $\mathbf{n}$ to be the normal of the plane $(\mathbf{n}, \delta)$ minimizing

$$E(\mathbf{n}, \delta) = \sum_{j \in N_r(i)} \mathbf{n}^T \cdot \mathbf{p}_j - \delta$$

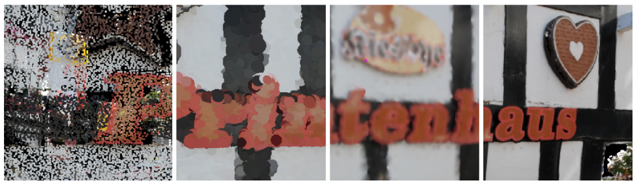


Figure 1: Different rendering techniques. (1) Simple point rendering produces gaps. (2) Naive splat rendering produces strong gradients at wrong image positions. (3) Blending techniques smooth out gradient information. (4) Textured splats are nearly photorealistic renderings.

and orient it towards $\mathbf{s}(i)$, meaning $\mathbf{n}^T \cdot (\mathbf{p}_i - \mathbf{s}(i)) < 0$.

To remove small cluttered point sets, like those typically produced by fast-moving objects, we apply a simple filter discarding points $\mathbf{p}_i$ with $|N_r(i)| < m$, where we set $m = 5$ and $r = 0.3$. Laser scanners radially shoot rays into the environment to create sample points and thus do not produce a uniform sampling. We compensate for this and remove redundant points by applying a grid filter, which assigns at most one representative point per $2cm^3$ grid cell.

A. *Sift-Realistic Rendering*

3D model-based localization approaches require pictures taken from many different point of views. Therefore, it is important to be able to render new synthetic images from arbitrary positions. Generating such virtual images could, *e.g.*, be done by rendering simple points using the OpenGL rendering pipeline, but selecting a good point size is hard. For points being rendered rather small, there will be many gaps between projected points. Large points, on the other hand, can easily produce hard edges between neighboring points. In both cases, gradients computed in those images are highly unstable and would lead to very unreliable feature descriptors. In what follows, we will describe the used rendering techniques which are suitable for visualizing information associated to the point cloud. This information may range from pure point colors over image gradients to entire image regions used to texture small surface patches.

Splat Rendering. The first suggested technique, Splat Rendering, is commonly used in the graphics community and usually needs an associated normal, a color, and a radius r_i for each point $\mathbf{p}_i$ to render the points as small discs. To compute this radius, one could, *e.g.*, define a global constant. Although we applied a grid filter, the point cloud still might contain undersampled regions, so we suggest to locally adapt the splat radius for each point. For this we propose to first divide the disc with normal $\mathbf{n}_i$ and center $\mathbf{p}_i$ into 12 sectors. Then we project each neighbor $\mathbf{p}_j \in N_{r=0.2}(\mathbf{p}_i)$ onto the disc and compute for every sector the closest projected neighbor point w.r.t. $\mathbf{p}_i$. We set the splat radius to be the maximum of all closest projected neighbor points. In order to compute this maximum correctly, we do not consider empty sectors. When setting a global scaling factor for the splat radii (*e.g.*, to 0.8), we ensure that splats partially overlap with each other. This creates splats also large enough in regions where the next neighbor in a certain direction is far away.

Using splat rendering will close gaps between points and thereby reduce the occurrence of problematic gradients between foreground and background pixels. A simple implementation of this technique renders point primitives and adapts the point size in the vertex shader, while the fragment shader computes the correct depth value for a perspectively correct appearance of the splats. However, this technique produces undesired hard edges between neighboring splats. This is why we implemented a three-pass rendering approach similar to [4]. The first pass renders the splats without any color information and generates a depth profile of the scene. *I.e.* we store the distance to the visible surface in every pixel of the virtual image. When shifting the scene slightly into the viewing direction during the first pass, a simple depth test can be used in the second pass to remove splats lying far behind the visible surfaces. For each pixel of the virtual image, the second rendering pass sums up the colors and normals of the remaining splats, which lie in the proximity of the visible surface and which perspectively project onto the respective pixel. The third and last rendering pass normalizes the per-pixel sums of colors and normals by dividing them by the number of splats projecting onto this pixel. Using the normal information, it is possible to adapt the final color values according to a local lighting model. Although this generates visually pleasant color transitions between neighboring splats, the technique represses existing gradients leading to less significant feature descriptors, as we will show in our experimental section.

When additional texture information is available we can enrich the gradients of our rendered image. For this, we propose an adaptation of the method presented in [4]. Instead of using the same color value for each splat, we store an index of the image (with known projection matrix P) the color value was taken from. During the second rendering pass, we then use P to project the fragments of a splat into the respective reference image and thereby texture the splat. As shown in Fig. 1, the resulting images are nearly photorealistic. In our experiments, we refer to the described rendering techniques as *Points*, *Splats*, *Phong Splats*, and *Textured Splats*, respectively (*c.f.* Fig. 1).

Texturing splats propagates as much information as possible from the set of sparse images into the synthetic views, but storing all images together with a large point cloud requires a lot of GPU memory for large urban scenes. We therefore designed two less resource intensive completion techniques which can be applied in a post processing step on images produced by point based rendering. The aim of these techniques is to preserve image gradients, which are the key component of the SIFT descriptor. Instead of using complicated heuristics to adapt the size, shape, and alignment of splats to get an optimal blending result, we compute synthetic images in image space using inpainting [3] and color adaption techniques [19].

For the first proposed rendering technique, we associate an image intensity to each point while for the second method we require to additionally attach an intensity gradient to each point, which is easily obtained, since most 3D scanners are equipped with a consumer level camera used to colorize the point cloud. As a preprocessing step, we first generate a 2.5D image (using an arbitrary simple splat or point rendering technique) to identify fore- and background pixels. This on the one hand identifies the pixels we need to inpaint and on the other hand ensures that the resulting image is only affected by visible points. In what follows, we describe both rendering techniques in more detail.

Image Completion. We designed *Intensity Completion* to preserve the intensities of the projected points, while it automatically finds color transitions between projected points to change existing gradients as little as possible. This is done by minimizing the thin plate energy [5]

$$E_I = \sum \|\Delta I(u,v)\|^2 \quad , \qquad (1)$$

where the intensities $I_1, \ldots, I_K$ of the projected points $(u_1, v_1), \ldots, (u_K, v_K)$ are interpolated and define the constraints $I_k = I(u_k, v_k)$ for the linear system. This will preserve the original intensities contained in our data set and since the curvature (or bending energy) of the function $I(u,v)$ is globally minimized, the gradients between samples are affected as little as possible. Using the well-known discrete Laplace operator

$$\Delta I(u,v) = \sum_j w_j \cdot (I(u,v) - I(u + \Delta u_j, v + \Delta v_j))$$

we can set up a linear system of equations to solve for the missing pixel colors. Here $(\Delta u_j, \Delta v_j)$ denote the pixel offset to one of the four (left, right, top or bottom) neighboring pixels. To speed up the computation and to guarantee that the linear system has full rank, we flood fill the foreground pixels starting from the projected point positions and thereby identify the free variables of our linear system.

Gradient Completion is our second approach to preserve gradient information. Instead of interpolating intensity values in the synthetic image, we compute one intermediate image which interpolates image gradients projected from the original images into the synthetic view. Similar to the method presented in [19], the interpolated gradients serve as a guidance field which is integrated to an intensity image using the projected point intensities as additional constraints. One substantial difference to [19] is the selection of the pixels to be inpainted, which again we determine by flood filling the foreground.

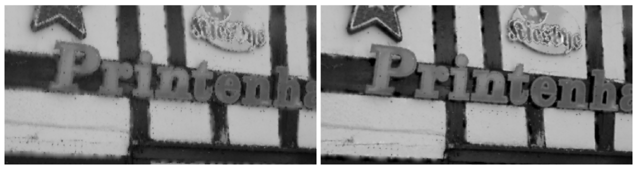


Figure 2: Left: *Intensity Completion* preserves intensities and varies the gradients between projected points as little as possible. Right: *Gradient Completion* additionally preserves intensity gradients taken from the reference image resulting in visually much sharper image features leading to more accurate SIFT descriptors as shown in the results.

To be more precise, we define $G(u,v) = \nabla I(u,v)$ to be the 2D vector field representing the image gradients. For each projection (u_k, v_k) of a point $\mathbf{p}_k$, we add the constraint

$$\nabla I(u_k, v_k) = \nabla J(\mathbf{W}(u_k, v_k)) \cdot \frac{\partial \mathbf{W}(u_k, v_k)}{\partial(u,v)}$$

stating that the intensity gradient at the pixel (u_k, v_k) in the virtual image I should match the gradient at a corresponding pixel in the reference image J. Here $\mathbf{W}(u,v) = (a/c, b/c)^T$ is the homography which maps pixels (u,v) from the synthetic image I to the reference image J. Given the projection matrices $P_I = (Q_I|\mathbf{q}_I) \in \mathbb{R}^{3\times4}$ and $P_J = (Q_J|\mathbf{q}_J) \in \mathbb{R}^{3\times4}$ of the images I and J and the equation $(\mathbf{n}_k, \delta_k)$ of the plane at position $\mathbf{p}_k$, the vector $(a,b,c)^T$ is computed as

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} = [\delta \cdot Q_J - \mathbf{q}_J \mathbf{n}^T] \cdot [\delta \cdot Q_I - \mathbf{q}_I \mathbf{n}^T]^{-1} \cdot \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \quad . \qquad (2)$$

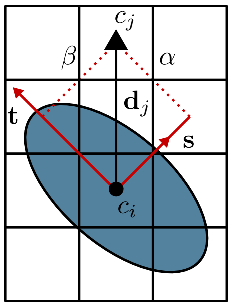
We find the remaining gradients by minimizing the energy

$$E_G = \sum ||\Delta G(u,v)||^2 \qquad (3)$$

similar to *Intensity Completion* which solves for the intensity values. Following the derivation in [19], we compute the final image by fixing intensity values at the projected point positions while simultaneously restoring image gradients according to the previously computed guidance field. We show an example of both methods in Fig. 2. Propagating gradient information into the synthetic view better preserves sharp image features. Both approaches use uniform weights $w_j = 1$ for the Laplace operator in Eq. 1 and 3. We refer to them as *Intensity* (*Int. Comp. (iso)*) and *Gradient Completion* (*Grad. Comp. (iso)*).

Anisotropic value propagation. Using uniform weights leads to an isotropic value propagation, which is not correct for perspective distorted surfaces. We propose a method to adapt the weights w_j of an edge between pixels according to an estimated surface normal at the 3D point $\mathbf{p}$ which projects onto the midpoint of the edge. This normal can be computed by averaging all splat normals intersecting the ray from the camera center through the edge midpoint. Orthographically projecting this normal into image space defines a vector $\mathbf{s} \in$

$\mathbb{R}^2$, which has zero length if the camera looks directly onto the splat and which is large if the splat normal is orthogonal to the viewing direction. It basically represents the smaller half-axis of an ellipse formed by the projection of the circular splat. We compute the larger half axis of the ellipse by



defining a vector $\mathbf{t} \in \mathbb{R}^2$ perpendicular to $\mathbf{s}$ and $\|\mathbf{t}\| = 1$. Let $\mathbf{d}_j \in \mathbb{R}^2$ be the edge from pixel i to a neighboring pixel j. Intuitively, we would like to define large weights for this edge if it is parallel to $\mathbf{s}$ and if $\mathbf{s}$ itself is large, since then small distances in image space map to large distances in object space and the influence of the neighboring value should be damped. To be more precise, we compute coefficients α and β such that we can express the direction $\mathbf{d}_j$ as a linear combination of the two half-axes $\mathbf{d}_j = \alpha\mathbf{s} + \beta\mathbf{t}$ This leads to values for α and β in the range $[1, \ldots, \infty]$, which are used to define the edge weights w_i as

$$w_j = \min\left(\epsilon, \frac{1}{\sqrt{\alpha^2 + \beta^2}}\right).$$

We use a small value $\epsilon = 10^{-4}$ to avoid the edge weights to be zero. We refer to these completion techniques using *anisotropic* weights as *Intensity (Int. Comp. (aniso))* and *Gradient Completion (Grad. Comp. (aniso))*.

Textured Splats with Image Completion. Rendering very large textured splats closes small gaps between neighboring splats. But due to small errors in the camera calibration, errors in the computation of the normal and errors in the point measurement, this tend to blur the final image, which reduces the clarity of the available gradients. Rendering small splats on the other hand sharpens the image but may produce gaps between neighboring splats due to the non uniform point sampling, which results unwanted image gradients. So, for the last investigated rendering approach we propose a technique which combines textured splats with an inpainting method to get the best of both worlds. Therefore we slightly adapt the three pass textured splat rendering approach. In the first pass we render large splats resulting in a continuous foreground map F_1 which needs to be colorized. In the second pass we render small splats and discard splats lying far behind the foreground F_1. This second pass also sums up colors and image gradients, which are correctly transformed into the virtual view using the plane induced homography of Eq. 2. Note that this blending in the gradient domain was not considered in [27] and is an alternative way to reduce color transitions between photographs. The third pass normalizes colors and image gradients leading to a gradient image and an intensity image with foreground map F_2. All gradients in F_2 serve as constraints in our *Gradient Completion* to compute a guidance field for all pixels in F_1. Again the final image is computed by fixing intensity values at all pixels of F_2 and restoring the image gradients according to the guidance field. We refer to this technique as *Textured Splats + Completion* (*Tex. Splats + Comp.*).

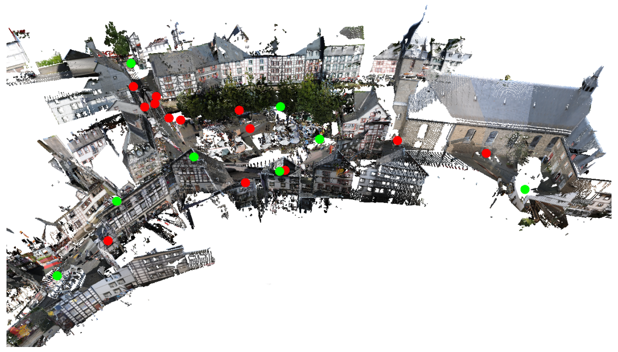


Figure 3: Our dataset, created from eight scan positions (green). Red points denote additional scanning positions not used to create the point cloud.

IV. Experimental Evaluation

We used a Riegl LMS-Z390i terrestrial scanner to capture a large scene in a historic European town from eight distinct locations. The images required to colorize the resulting point cloud, in the remainder referred to as the *included images*, were taken with a Nikon D300 camera mounted on top of the scanner. We then computed the extrinsic calibrations of the camera poses by merging the scans into a single coordinate system. The scanner locations used to create the point cloud are shown as green dots in Fig. 3. The resulting dataset consists of around 9 million 3D points. It also contains tourists standing and sitting around, as well as artifacts caused by walking pedestrians.

Our study of different point cloud rendering techniques is motivated by an image-based localization task in which we want to determine the camera pose for each query photo relative to the point cloud of the scene [8], [12], [22]. We find matches between 2D features and 3D points in the model, which enables us to estimate the camera pose using a 3-point-pose algorithm inside a RANSAC-loop [6]. To obtain these 2D-3D correspondences, we match the query image against a set of database images registered against the point cloud [8], *i.e.*, a set of images for which the 3D points corresponding to their image features are known. In order to localize query photos taken from viewpoints significantly different from the included images, we create a database of rendered views of the point cloud. In the following, we thus show that rendering synthetic images allows us to approximate novel viewpoints in a way that is similar enough to real photos to enable feature matching.

We split our experimental evaluation into two parts. In the first part, we analyze the different rendering techniques presented in Sec. III by comparing the descriptors extracted from the included images to descriptors found in synthetic views rendered from the exact same viewpoints. This allows us to estimate the best-case performance of the different methods. In real-world applications, we obviously do not know the exact position from which a query image was taken, but rather generate synthetic views from sample positions on a regular grid [8]. In the second experiment, we therefore determine the viewpoint invariance of descriptors obtained using the different techniques by investigating the required sampling density. For both experiments, we use a publicly available GPU SIFT implementation [25] to extract SIFT features on real photos with a resolution of 1065×697 pixels and synthetic views rendered with a resolution of 1024×768.

Comparing different rendering techniques. The goal of SIFT-realistic rendering is to generate synthetic images containing feature descriptors that faithfully reproduce the descriptors extracted from real photos. In the following experiment, we thus compare the descriptors from the included images with those found in synthetic views rendered from exactly the same viewpoint. We compare the different rendering techniques, *Point* and *Splat Rendering*, *Intensity* and *Gradient Completion*, and *Textured Splatting (+Completion)*, regarding the number of features found in the images, the repeatability of feature positions, and the similarity of the resulting descriptors. Note that the two texturing methods use image patches from the photos included in the scan and thus represent a best case scenario for SIFT-realistic rendering. We ignore all features for which more than 50% of the pixels used to compute the feature descriptor back-project into empty space.

Fig. 4(a) shows the average number of features found per image. Except for *Intensity Completion* and the texturing approaches, all rendering methods generate a similar number of features. *Intensity Completion* interpolates between pixel intensities and thus implicitly blurs the images, resulting in fewer Difference-of-Gaussian (DOG) extrema and thus fewer features. Similarly, the images obtained using normal and Phong splatting become more blurry by increasing the global scaling factor for the splat size as overlapping splats are blended together. The texturing methods are less affected by the global factor since the blended textures are well-aligned. Therefore, we individually selected the scaling factor for every splatting method that gave the best results over all tests while guaranteeing a large enough overlap between the splats to avoid holes in the renderings.

In order to reproduce the original descriptors from the photos, the features need to be found at similar positions in the rendered views. To evaluate the repeatability of the SIFT detector under the different rendering techniques, we follow the setup by Mikolajczyk *et al.* [14]. A feature from the real photo is considered repeatable if the synthetic view, rendered from the same viewpoint, contains a feature at a similar position such that the intersection-over-union (*i-o-u*) score between the two corresponding regions used to compute the descriptors is at least 0.5 [14]. Fig. 4(b) shows the average ratio of repeatable features per image for the different rendering techniques. *Intensity Completion* and the splatting approaches all yield a much better approximation of the rendered surface than point rendering and thus obtain better repeatability scores. Using *Gradient Completion* results in a significant increase in repeatability since it reconstructs image gradients, and thus DoG extrema, more faithfully. As can be expected, using patches extracted from the original photo yields the best repeatability. We notice that features found on lower scales are less repeatable than those found on higher scales. This drop is caused by slight errors in the registration of the scans or small measurement errors for the points, which in turn results in small artifacts when blending not perfectly aligned textures.

While most rendering techniques result in a low detector repeatability, even 10% of the image features can be enough to enable camera pose estimation if they can be matched successfully between the photos and the synthetic views. We therefore evaluate image matching quality next, following the common setup for image matching using approximate, tree-based nearest neighbor search [13]. For every descriptor $\mathbf{d}$ in the real photo, we find its two nearest neighbors $\mathbf{d}_1$, $\mathbf{d}_2$ in the rendered image. We then apply the SIFT ratio test and establish a match between $\mathbf{d}$ and $\mathbf{d}_1$ if $\|\mathbf{d} - \mathbf{d}_1\| < 0.8 \cdot \|\mathbf{d} - \mathbf{d}_2\|$. A match is considered to be correct if the *i-o-u* score between the corresponding features is at least 0.5. Fig. 4(c) shows the number of images which contain at least k correctly matching descriptors for different values of k. Fig. 4(d) details the ratio between correct and all established matches by showing a cumulative histogram over the inlier ratio for each method. As can be seen, the *Gradient Completion* technique performs significantly better than all methods based purely on colored points. In addition, it achieves higher inlier ratios, indicating that *Gradient Completion* finds fewer wrong matches. Yet, the performance of the two texturing approaches is still significantly better.

Location recognition using synthetic views. The previous experiment demonstrates that it is indeed possible to render synthetic images similar enough to the real photos to enable feature matching. In the second part of our experiments, we show that such synthetic views can be used to recognize *novel images* taken from viewpoints substantially different from the photos included in the scans. Therefore, we measure how close a synthetic view has to be to a novel image to obtain enough matches for pose estimation. We restrict the experiments to the image completing and texturing methods, as they performed significantly better than the point and splat rendering approaches in initial tests.

We obtain 140 novel images with ground truth positions by registering 13 additional scans, together with their camera

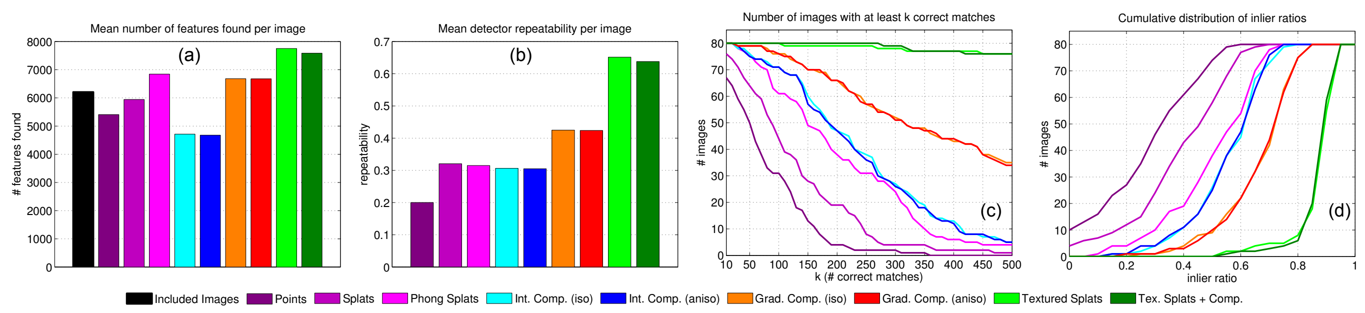


Figure 4: Statistics about (a) the average number of features found in the original images included in the scan (black) and the rendered views and (b) the repeatability of features detected in the original views and the synthetic images. (c) The number of images for which at least k correctly matching features passing the SIFT ratio test can be found between the original photo and the corresponding rendered view. (d) The cumulative distribution of the inlier ratios of the image, *i.e.*, the ratios of the number of correct matches to the number of matches that pass the ratio test.

positions, against the existing point cloud. The point data from the additional scans is only used to register the scans in order to obtain ground truth positions but not for rendering synthetic views. We thus do not consider novel photos showing scene structures not present in the original point cloud, *e.g.*, missing walls. This results in a test set of 126 novel images. Similarly, we allow the texture splatting approaches to utilize only patches from the *included* but not from the *novel* images. In contrast to the previous experiment, the texturing methods are thus no longer biased and can now be treated equally to the other techniques.

We model the ground plane of the scene using a height field on which we generate a regular grid of positions from which synthetic views are rendered, where two adjacent grid positions have a distance of one meter. For each position, we render 12 images by rotating a virtual camera with a field-of-view of $90°$ in steps of $30°$ to ensure that neighboring views have enough overlap. The virtual cameras are placed about 1.7m above the ground plane and are tilted by $15°$ towards the sky [8]. For every feature extracted in a rendered image, we compute its 3D point position by back-projecting the feature into the point cloud. We match each novel photo against all synthetic views rendered from grid positions within 20m of the ground truth location of the photo. The resulting 2D-3D correspondences are then used to estimate the pose of each novel image using a 3-point-pose solver within a RANSAC-loop [6]. We consider a match to be an inlier to an estimated pose if the reprojection error is below $\sqrt{10}$ pixels. We regard a novel image as localized if we can find a pose with at least 12 inliers [12], [22]. As a baseline, we match each novel image against each photo included in the scans.

Fig. 5 shows the results of this experiment. As can be seen, rendering synthetic views using only information about the scene geometry and intensity values for each point (*Intensity Completion*) allows us to localize a similar or larger number of novel images compared to using the

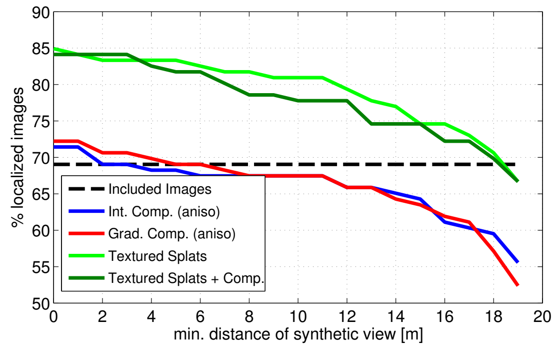


Figure 5: The percentage of novel images that can be localized using synthetic views as depending on the minimal distance between the ground truth position of the photo and the location from which the view was rendered. Compared to matching against the original photos, rendered images allow us to localize novel views for a range of distances.

original image data, as long as the synthetic views are sampled densely enough. Combining the intensity with gradient information (*Gradient Completion*) additionally improves the results. While we observed that *Gradient Completion* performs better than *Intensity Completion* in the first experiment, we cannot demonstrate a similar improvement in localization performance. Combining image data with the scene geometry (texturing techniques) enables us to localize a significantly larger amount of novel photos compared to the baseline approach. Since we only match against synthetic views with a maximal distance of 20m to the ground truth positions of the novel images, only considering rendered images that are at least 19m away naturally decreases the number of localized images. From the results of the baseline, we would expect that we could improve the results by also using views with larger distances. We measure a mean localization error of less than 8cm for each localized image and each method.

From the results of this last experiment, we derive the following usage guidelines: Texture splatting makes it possible to obtain better localization performance than is possible with the original images captured from the scanning positions. In addition, they can be rendered in real time even for medium-scaled city models. They therefore provide a major benefit for recognition from laser scanned data. *Intensity Completion* achieves a similar performance as using the original photos, enabling image-based localization even if only the original (colored) point cloud is available. In addition, storing only (colored) 3D point data yields a more compact data representation compared to also storing the additional photos. If gradient information is available in addition to the intensities, *Gradient Completion* can be used to improve the localization performance at the cost of storing only a single gradient vector per 3D point.

V. Conclusion

In this paper, we have shown that it is possible to render synthetic views in a way that allows us to match SIFT descriptors extracted in them against descriptors found in real photos. We have investigated different usage scenarios, based on the amount of information available for the point cloud. Since classical point and splat rendering techniques are not suitable for our task of image-based 3D localization, we have proposed individual rendering approaches for the different scenarios. *Intensity Completion* uses only the intensity information from the 3D points to create SIFT-realistic renderings, while *Gradient Completion* requires additional gradient information to reconstruct image content more faithfully. Both techniques achieve good location recognition performance in the case that only colored point cloud data can be used. If photos are available, *texture splatting* techniques lead to even better recognition performance than using the original images. Our results show that rendering synthetic images allows us to localize additional views substantially different from the included photos.

In future work, we plan to integrate the synthetic views into a full image-based localization pipeline and to evaluate different types of feature detectors and descriptors.

Acknowledgments. This project has partially been funded by the DFG Cluster of Excellence UMIC (DFG EXC 89).

References

[1] M. Alexa and M. Gross. Point-based computer graphics. In *SIGGRAPH*, 2004.

[2] N. Amenta, S. Choi, and R. K. Kolluri. The Power Crust. In *SMA*, 2001.

[3] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester. Image inpainting. In *SIGGRAPH*, 2000.

[4] M. Botsch, A. Hornung, M. Zwicker, and L. Kobbelt. High-quality surface splatting on today's GPUs. In *SPBG*, 2005.

[5] J. Duchon. Splines minimizing rotation-invariant semi-norms in Sobolev spaces. In *Constructive Theory of Functions of Several Variables*. 1977.

[6] M. Fischler and R. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *CACM*, 24(6):381–395, 1981.

[7] A. P. Gee and W. Mayol-Cuevas. 6D Relocalisation for RGBD Cameras Using Synthetic View Regression. In *BMVC*, 2012.

[8] A. Irschara, C. Zach, J.-M. Frahm, and H. Bischof. From Structure-from-Motion Point Clouds to Fast Location Recognition. In *CVPR*, 2009.

[9] B. Kaneva, A. Torralba, and W. T. Freeman. Evaluating Image Feaures Using a Photorealistic Virtual World. In *ICCV*, 2011.

[10] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *SGP*, 2006.

[11] L. Kobbelt and M. Botsch. A survey of point-based techniques in computer graphics. *Comp. Graph.*, 28(6):801–814, 2004.

[12] Y. Li, N. Snavely, and D. P. Huttenlocher. Location Recognition Using Prioritized Feature Matching. In *ECCV*, 2010.

[13] D. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91–110, 2004.

[14] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. Van Gool. A comparison of affine region detectors. *IJCV*, 65(1-2):43–72, 2005.

[15] J.-M. Morel and G. Yu. ASIFT: A New Framework for Fully Affine Invariant Image Comparison. *SIAM J. Imaging Sciences*, 2(2):438 –469, 2009.

[16] NAVTEQ. Navteq true - data collection, 2013.

[17] R. A. Newcombe, S. Lovegrove, and A. J. Davison. DTAM: Dense Tracking and Mapping in Real-Time. In *ICCV*, 2011.

[18] M. Özuysal, M. Calonder, V. Lepetit, and P. Fua. Fast Keypoint Recognition Using Random Ferns. *PAMI*, 32(3):448–461, 2010.

[19] P. Pérez, M. Gangnet, and A. Blake. Poisson image editing. In *SIGGRAPH*, 2003.

[20] M. Pollefeys *et al.*(19 authors). Detailed Real-Time Urban 3D Reconstruction From Video. *IJCV*, 78(2-3), 2008.

[21] S. Rusinkiewicz and M. Levoy. QSplat: a multiresolution point rendering system for large meshes. In *SIGGRAPH*, 2000.

[22] T. Sattler, B. Leibe, and L. Kobbelt. Fast Image-Based Localization using Direct 2D-to-3D Matching. In *ICCV*, 2011.

[23] N. Snavely, S. M. Seitz, and R. Szeliski. Photo tourism: Exploring photo collections in 3d. In *SIGGRAPH*, 2006.

[24] S. Thrun. Rethinking the automobile, 2011.

[25] C. Wu. SiftGPU: A GPU implementation of scale invariant feature transform (SIFT), 2007.

[26] C. Wu, B. Clipp, X. Li, J.-M. Frahm, and M. Pollefeys. 3D model matching with viewpoint-invariant patches (VIP). In *CVPR*, 2008.

[27] R. Yang, D. Guinnip, and L. Wang. View-dependent textured splatting. *The Visual Computer*, 22(7):456–467, 2006.