This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TVCG.2015.2430333

Multi-Depth-Map Raytracing for Efficient Large-Scene Reconstruction

Murat Arikan, Reinhold Preiner and Michael Wimmer

Abstract—With the enormous advances of the acquisition technology over the last years, fast processing and high-quality visualization of large point clouds have gained increasing attention. Commonly, a mesh surface is reconstructed from the point cloud and a high-resolution texture is generated over the mesh from the images taken at the site to represent surface materials. However, this global reconstruction and texturing approach becomes impractical with increasing data sizes. Recently, due to its potential for scalability and extensibility, a method for texturing a set of depth maps in a preprocessing and stitching them at runtime has been proposed to represent large scenes. However, the rendering performance of this method is strongly dependent on the number of depth maps and their resolution. Moreover, for the proposed scene representation, every single depth map has to be textured by the images, which in practice heavily increases processing costs. In this paper, we present a novel method to break these dependencies by introducing an efficient raytracing of multiple depth maps. In a preprocessing phase, we first generate high-resolution textured depth maps by rendering the input points from image cameras and then perform a graph-cut based optimization to assign a small subset of these points to the images. At runtime, we use the resulting point-to-image assignments (1) to identify for each view ray which depth map contains the closest ray-surface intersection and (2) to efficiently compute this intersection point. The resulting algorithm accelerates both the texturing and the rendering of the depth maps by an order of magnitude.

Index Terms—Point-based rendering, raytracing depth maps, large-scale models

1 Introduction

The high-quality reconstruction and visualization of large scenes from huge amounts of raw sensor data is an important and particularly challenging task in many application areas, ranging from digitization and preservation of cultural heritage, over virtual reality and games, to planning and visualization for architecture and industry. To virtually recreate such scenes, geometry is reconstructed from scanned 3D point-cloud data and commonly textured from registered high-resolution photographs taken at the original site.

In practice, computing a high-quality texturing from such images is a non-trivial task due to image overlaps, varying lighting conditions, different sampling rates and image misregistrations. One potential workflow represents the geometry as a point cloud again and directly texture-maps the resulting point-based surface [1], [2]. However, this approach can exhibit visible artifacts like illumination seams and texture misalignments, which heavily degenerate the visual quality of the result. A more common approach is to convert the point data into a mesh once [3], [4] and then render the scene as a textured mesh, reducing both memory and bandwidth consumption. In order to obtain the required texturing, an image-to-triangle assignment (also called *labeling*) problem has to be solved, for which state-of-the-art methods [5], [6] use a graph-cut based optimization, which provides a homogeneous and high-quality solution. In large-scale scenarios, this is done once in an expensive preprocessing phase, and the resulting textured mesh is then used for efficient rendering. However, these methods are not very flexible – any change or addition in the geometry or image data requires an expensive relabeling of the mesh – and do not scale well due to the time complexity of the global labeling. Moreover, large-scale scenarios require an out-of-core computation of the mesh [7] and its texturing, imposing an additional maintenance overhead.

State of the art: To break down the problem complexity and accelerate the reconstruction and labeling preprocessing, Arikan et al. [8] introduced a *localized* textured surface reconstruction and visualization approach. They employ a set of *Textured Depth Maps* to represent the scene as a collection of *surface patches*, avoiding the reconstruction and maintenance of the whole surface and significantly reducing the optimization costs by labeling only a set of small depth maps instead of a large out-of-core mesh. These patches are triangulated and stitched at runtime, trading a minor increase in rendering time against a huge decrease in preprocessing time. Moreover, the patch-based representation offers both more flexibility and better scalability, since new patches can be added and textured easily without recomputing the whole surface. However, the rendering performance heavily depends on the number of depth maps and their resolution. This introduces a natural bound on the depth-

• *M. Arikan, R. Preiner and M. Wimmer are with the Institute of Computer Graphics and Algorithms, Vienna University of Technology, Austria.*
E-mail: marikan@cg.tuwien.ac.at

Copyright (c) 2015 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TVCG.2015.2430333

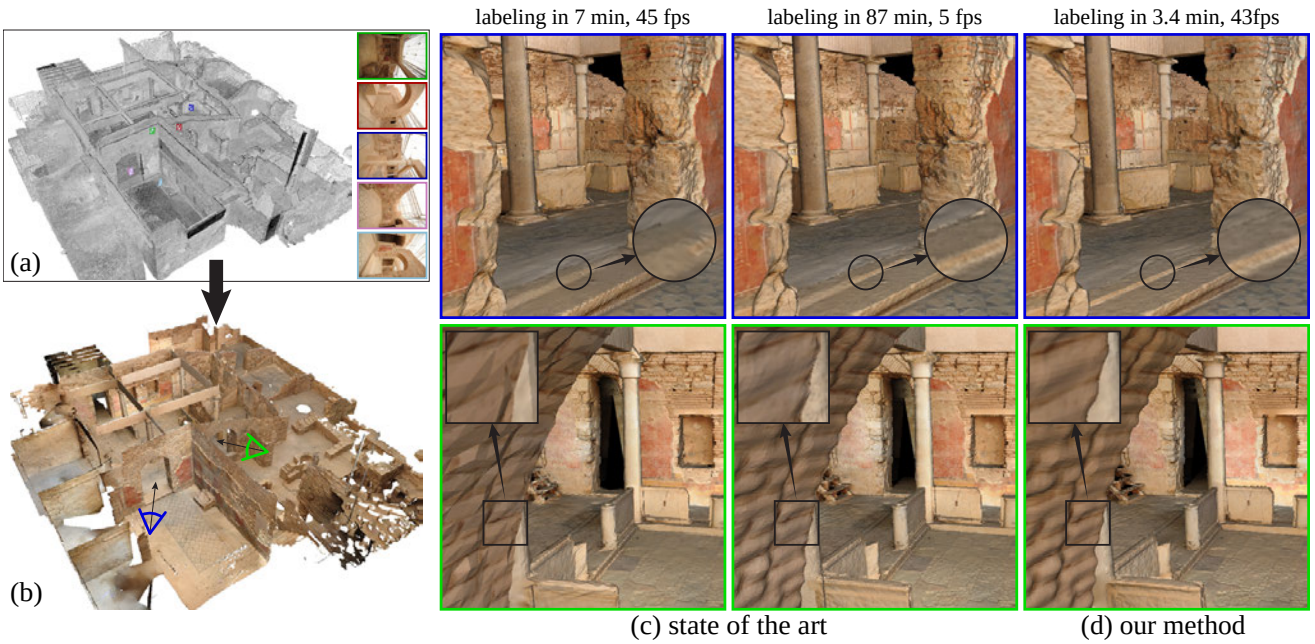

Fig. 1. (a) Point-cloud and image data acquired by a scanner. The data set consists of 682M points and 192 images. (b) Scene overview rendered by our method. (c) and (d) compare the state of the art [8] and our method in terms of performance and quality. The previous approach has to settle with a significantly lower geometric resolution in order to reach the performance of our new method.

map resolution usable to be rendered interactively, thus limiting the achievable geometric quality in the rendered image.

Solution approach: We introduce an *output-sensitive* visualization technique of such a patch-based surface representation. Instead of stitching high-resolution depth maps, which is expensive, we perform a *multi-depth-map raytracing* approach, which efficiently identifies for each view ray the depth map that contains the closest valid ray-surface intersection, and then finds this intersection point. Our method also avoids the labeling of every single depth map in the preprocessing, but instead labels a strongly reduced subset of the original point cloud, which in practice accelerates the labeling process by over an order of magnitude. To obtain high-quality per-pixel labels for texturing, this coarse point set is projected to the screen and its labels are upsampled using a geometry-aware Voronoi decomposition of the depth buffer at runtime.

As our main **contribution over the state of the art**, we propose a novel raytracing approach whose performance is independent of the number and resolution of the depth maps, therefore allowing for a high-quality real-time visualization of large scenes at much higher geometric resolution than the previous approach [8] (Fig. 1).

2 Related Work

The problem of textured scene reconstruction and visualization from large point clouds and photographs has been addressed by several authors.

Point-based rendering techniques like *surface splatting* [9], [10], [11], [12] render the input points as elliptical surface primitives *(splats)*, which are blended to obtain a smooth continuous surface. These methods have been coupled with texturing [1], [2] to obtain a textured point-based visualization of a scene. Texture mapping point-based surfaces avoids a costly large-scale mesh reconstruction, but does not produce optimal point-to-texture assignments. This can produce visible artifacts like texture misalignments and illumination seams.

Mesh-based textured reconstruction techniques achieve a continuous high-quality texturing of the scene by performing a global, graph-cut based optimization of the triangle-to-texture assignments on a single huge mesh [5], [6]. These methods produce high-quality visualizations of large scenes, but require a time-expensive preprocessing for the mesh reconstruction and labeling as well as a large maintenance overhead, making it inflexible to changes and extensions in the data set.

Therefore, Arikan et al. [8] recently proposed a **patch-based reconstruction** approach, which breaks down the meshing and labeling complexity by representing the scene by several surface patches, allowing for both a more efficient preprocessing and a more flexible and scalable data management. Their method generates a set of *textured depth maps* in a preprocessing and stitches them at runtime, which strongly couples the rendering performance with the number and resolution of these depth maps.

Copyright (c) 2015 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TVCG.2015.2430333

Our method builds upon this localized approach for the data representation, but alleviates its performance limitations by introducing a solution for an efficient **raytracing of multiple depth maps**. Finding ray intersections with surfaces represented by two-dimensional range maps has various applications, like rendering soft shadows [13], [14] or reflections [15].

Previous methods for raytracing large-scale scenes depend on the use of spatial acceleration data structures. Reshetov et al. [16] employs a spatial kd-tree to detect scene parts that are guaranteed not to intersect with a collection of view rays. Agrawala et al. [13] proposed a hierarchical ray traversal to skip over large sections of a ray that cannot possibly intersect the scene. Xie et al. [14] raytraces a *multi-layer* depth map to reduce shadowing artifacts. To cope with the additional overhead of searching an intersection point in multiple layers, they introduced a hierarchical intersection test against a quadtree, where each node contains the minimum and maximum depth value of the four child nodes in the layer below. In contrast, we use multiple *single-layer* depth maps covering a scene and employ a labeled coarse subset of the original point cloud to *directly* determine the depth map that is first intersected by a view ray. This is done by splatting the label information of these points into the screen, and *upsampling* their labels to obtain per-pixel labels. The resulting label of a pixel then indicates the depth map to be intersected by the pixel's corresponding view ray.

In the following, we give an overview of our preprocessing and rendering pipeline, and then describe each step of our reconstruction and texturing system in detail.

3 Overview

Our method takes as input a high-density 3D point cloud (denoted by P_{HD}), for example from a laser scanner, and a set of high-resolution photographs $\{I_j\}$ with known camera registrations. We propose a two-phase solution for an efficient high-quality visualization of the data.

In the *preprocessing* phase, we generate high-resolution depth maps by rendering the input point cloud P_{HD} from image cameras (Fig. 2a, Section 4.1), and compute an image-to-point assignment (referred to as *labeling*) only for a small subset $P_{LD} \subseteq P_{HD}$ (Fig. 2b, Sections 4.2 and 4.3), which we will call *proxy points*.

At *runtime*, we reconstruct a high-resolution depth buffer, which stores depth values of the scene as viewed from the user's camera. This is done by first splatting proxy points, and then raytracing the precomputed depth maps, starting from coarse splat positions (Fig. 2c, Section 5.1). In a second step, the labels of P_{LD} are used to obtain an upsampled depth-buffer labeling, which is required for texturing the final output image (Section 5.2).

4 Preprocessing

4.1 Generating the Depth Maps

For each image I_i, we generate a depth map D_i by rendering the original point cloud P_{HD} from the same viewpoint and with the same viewing parameters as I_i. For rendering, we use oriented circular splats as rendering primitives and employ an out-of-core octree data structure [17] to store P_{HD} and stream visible points to the GPU. If point normals are not available, we compute them by fitting a least-square plane to a neighborhood of each point. The splat radii are determined from the density of the rendered points [17].

4.2 Generating the Proxy Points

The proxy points P_{LD} are obtained by sub-sampling P_{HD}. To this end, the octree storing P_{HD} is pruned to contain only its k top-most levels, which correspond to the k lowest levels of detail of P_{HD}. As we will show in Section 6.1, the choice of k is a trade-off between performance and rendering quality. We will also demonstrate that using only a small subset of the original point cloud as proxy points strongly accelerates the subsequent labeling stage, but is still sufficient for a high-quality textured reconstruction from the depth maps at render time.

4.3 Labeling

To obtain a point-to-image assignment, first a set of candidate images of each point $\mathbf{p} \in P_{LD}$ is determined. The image I_i is a candidate of $\mathbf{p}$ if $\mathbf{p}$ is not occluded from the camera view of I_i. In the second step, we pick for each point $\mathbf{p}$ its best-suited candidate image I_j for texturing, i.e., $\mathbf{p}$ is *labeled* with the index j.

This assignment has to consider the quality of the image-to-geometry mapping as well as continuity in the texturing (i.e., avoiding visible artifacts between areas labeled by different images). We solve this problem by a graph-cut based optimization, where the quality and continuity criteria are addressed by a data and a smoothness term, respectively. However, instead of operating on triangles as done in previous approaches, we use the $k$nn-graph built upon the points as input graph for the optimization. We use the same data and smoothness term as in Arikan et al. [8]: For the points, the data term favors orthogonal and close image views. In contrast, the smoothness term penalizes label changes with strong color differences along edges between neighboring points.

5 Multi-Depth-Map Raytracing

In this section, we describe how the precomputed data is used at runtime to obtain a high-quality visualization of the scene. We perform two major steps, *surface*

Copyright (c) 2015 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TVCG.2015.2430333

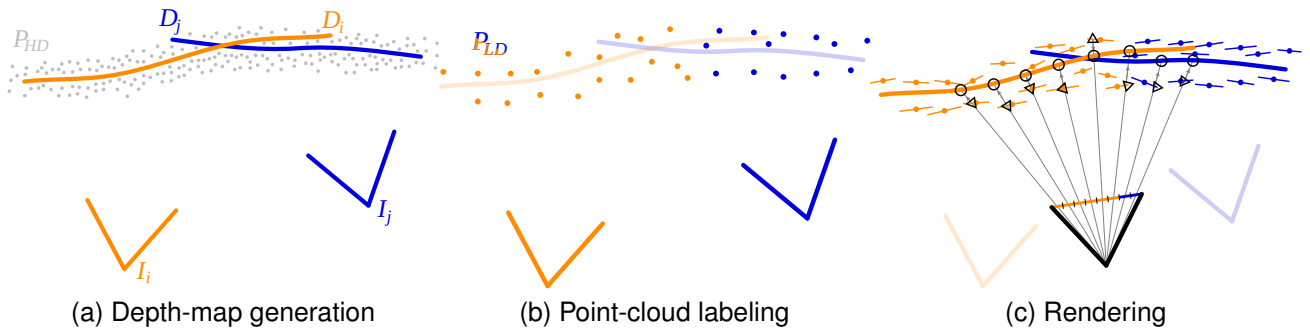

Fig. 2. Overview of our pipeline. (a) High-resolution depth maps are generated by rendering the high-density input point cloud P_{HD} from image cameras. The depth maps D_i and D_j, lifted to 3D, are color-coded by their corresponding images I_i and I_j, respectively. (b) P_{HD} is subsampled, and the resulting low-density point cloud P_{LD} is labeled by the input images, i.e., each point of P_{LD} is assigned to an input image. This concludes the preprocessing phase. (c) Coarse surface positions (marked with $\triangle$) that are equipped with labels are efficiently obtained by splatting points of P_{LD}. Then, starting from these positions, raytracing the respective depth maps yields high-resolution surface positions (marked with $\bigcirc$).

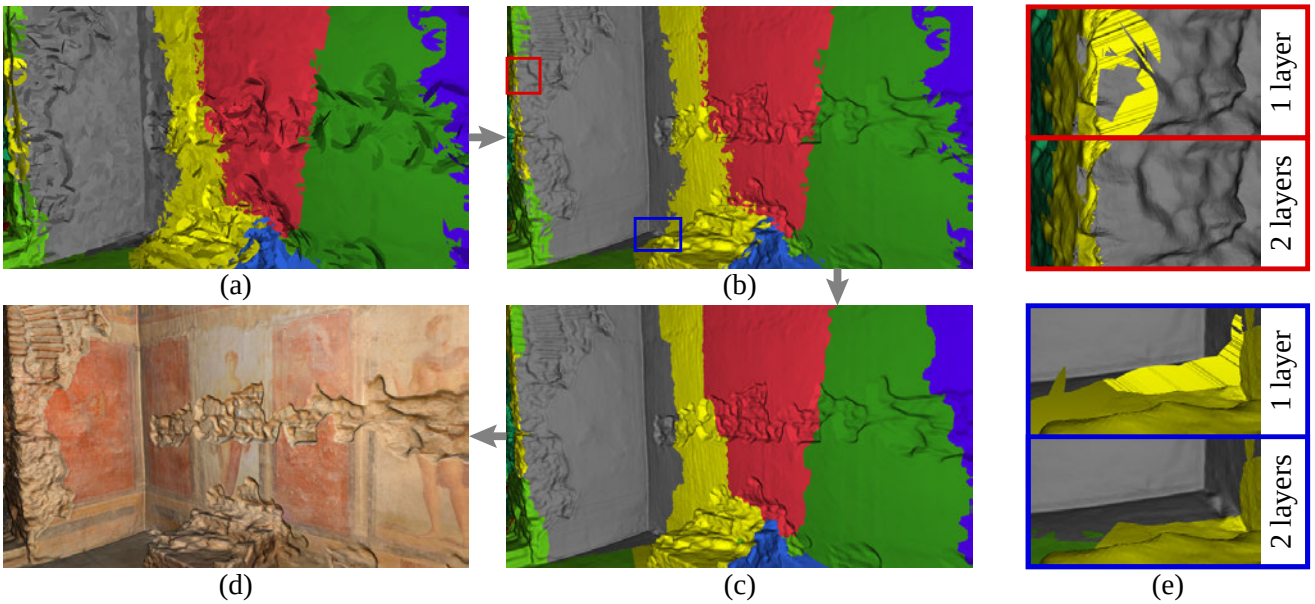

Fig. 3. (a)-(d) Rendering pipeline. (a) Splatting proxy points P_{LD} (color coded according to labels). (b) Raytracing high-resolution depth maps. (c) Per-pixel labeling to be used for texturing. (d) Textured and shaded surface. (e) shows invalid intersections with discontinuity triangles that can occur when raytracing a single depth-map layer along each view ray.

generation and *color mapping*, to render a textured surface.

The surface-generation step first renders P_{LD} as splats to create a depth buffer representing coarse surface positions and a corresponding label buffer (Fig. 3a, Section 5.1.1). For rendering, we employ the same out-of-core data structure [17] that we used to generate the depth maps. Then, starting from these coarse positions, for each pixel the depth map indicated by the label buffer is raytraced in a full-screen rendering pass to produce a high-resolution depth buffer (Fig. 3b, Section 5.1.2).

The following color-mapping step splats P_{LD} again to generate a high-resolution label buffer by upsampling the labels that were output in the first pass (Fig. 3c, Section 5.2.1).

Finally, high-resolution images relevant for texturing are cached to the GPU (Section 5.2.2), and the color of each pixel is retrieved in a full-screen pass by projecting it onto its assigned image based on the depth and label retrieved from the high-resolution depth and label buffers (Fig. 3d, Section 5.2.3).

In the following, we will describe the individual steps of our rendering pipeline in more detail.

Copyright (c) 2015 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TVCG.2015.2430333

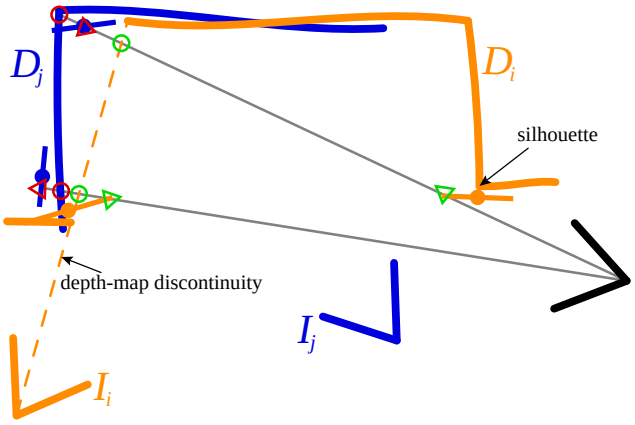

Fig. 4. Multiple label layers for raytracing. Invalid intersections (◯, green) can be caused by initializing raytracing with the front-most splat position (△, green) and its label i. In this case, starting from second-layer positions (△, red) with label j, raytracing D_j produces valid intersection points (◯, red).

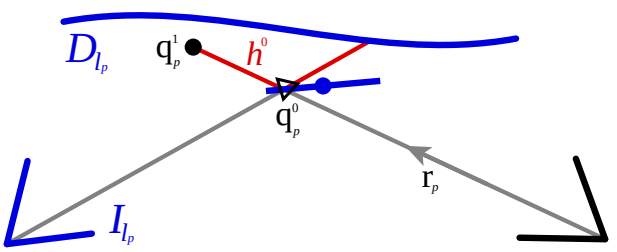

Fig. 5. A single iteration of the iterative search, taking a step of h^0 on $\mathbf{r}_p$. The start position $\mathbf{q}_p^0$ and its label index l_p are retrieved from the closest ray-splat intersection.

5.1 Surface Generation

5.1.1 Visibility Stage

In the first pass, P_{LD} is rendered with z-buffering, writing to a *depth buffer* B_d and a *label buffer* B_l. The generated buffers represent the front-most *label layer*, which will be used in the raytracing pass to compute the intersection points of view rays with depth maps. In particular, a ray cast from the viewpoint through the pixel position $p = (x_p, y_p)$ will intersect the depth map indexed by label $l_p = B_l(x_p, y_p)$, and the intersection search will start at the 3D position $\mathbf{q}_p^0$ corresponding to the depth value $d_p = B_d(x_p, y_p)$.

This fast, *direct* selection technique gives the correct depth map for the vast majority of the view rays in the screen. However, in some cases, the labels in B_l will not correspond to a depth map that contains a valid ray intersection. This mostly happens for proxy points splatted very close to depth-map discontinuities and silhouettes (Figs. 3e and 4). In such a case, we retrieve the depth-map label for the intersection test from the next closer proxy point splat along the ray with a different label. For this, we have to store a second label layer to look up the next depth map for raytracing if no valid intersection point is found in the first depth map (Figs. 3e and 4). To extract this second label layer, P_{LD} is rendered again with z-buffering, and at each pixel p, fragments with label l_p or depth values less than d_p are discarded. The resulting depth and label values are written into two additional buffers. We then extend this approach to multiple layers computed in a depth-peeling fashion [18].

5.1.2 Raytracing Pass

We render a full-screen quad and perform for each screen-space pixel p an iterative search in the high-resolution depth map D_{l_p}, followed by a binary search. The iterative search starts at $\mathbf{q}_p^0$ and uses a stepsize that adapts to the current estimated distance to the intersection. The next point on the ray is computed as follows:

$$\mathbf{q}_p^i = \mathbf{q}_p^{i-1} + h^{i-1} * \mathbf{r}_p, \tag{1}$$

where $\mathbf{r}_p$ is the normalized ray direction. The adaptive stepsize h^{i-1} is calculated as the signed distance of $\mathbf{q}_p^{i-1}$ to D_{l_p} along the line to the center of projection of I_{l_p} (Fig. 5). The distance is signed since the low-resolution depth-buffer value used as initialization can lie in front or behind the high-resolution depth map.

Since $\mathbf{q}_p^0$ provides a sufficiently good initialization, only a few iterations are required (except at oblique angles) to find a pair of points $\mathbf{q}_p^{k-1}$ and $\mathbf{q}_p^k$ enclosing an intersection. In a second step, the interval $[\mathbf{q}_p^{k-1}, \mathbf{q}_p^k]$ is refined by a binary search to find a more accurate approximation $\hat{\mathbf{q}}_p$ of the intersection point.

We then check whether $\hat{\mathbf{q}}_p$ lies on a depth discontinuity of D_{l_p}. For this, we detect the four texels of D_{l_p} (yielding two triangles in 3D) that are nearest to the projection of $\hat{\mathbf{q}}_p$ into D_{l_p}, and assume a discontinuity if the depth disparity between any two triangle vertices is above a user-defined threshold (20cm in our examples). Averaging the two triangle normals also provides us with per-pixel normals, which can be optionally used for lighting effects. In case of a depth discontinuity, raytracing is re-performed to find an intersection with the depth map retrieved from the next label layer (Fig. 4).

The results of the raytracing pass basically refine for each pixel the depth value and – in case of a discontinuity – the label value originally obtained from splatting P_{LD}.

5.2 Color Mapping

5.2.1 Labeling Pass

The aim of this rendering pass is to equip the high-resolution depth data from the previous pass with labels that are suitable for texturing. Unfortunately, we cannot use the label buffer B_l created in the visibility stage as is, since due to the low resolution of P_{LD}, this buffer exhibits non-regular borders between

Copyright (c) 2015 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TVCG.2015.2430333

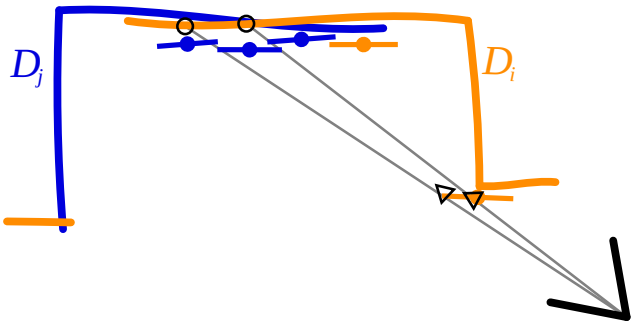

Fig. 6. Illustration of false labels near silhouettes. View rays through the splat at the silhouette have valid intersections with D_i. Therefore, this splat projects its label i to the background, causing corresponding pixels of that region to be assigned the label i instead of j.

differently labeled regions (Fig. 3b) and false labels near silhouettes (Figs. 6 and 8c).

Instead, we compute a Voronoi decomposition of the screen space into equally labeled regions. The seeds of this decomposition are specified by the projection of the points $\mathbf{c}_j \in P_{LD}$ into screen space, and distances between pixels and seed points are measured by the Euclidean distances of the respective points $\hat{\mathbf{q}}_p$ and $\mathbf{c}_j$ in 3D. This way, each pixel will be assigned the label of its closest seed $\mathbf{c}_j$. This results in a high-resolution label buffer with per-pixel labels upsampled from the sparse labeling information in P_{LD}.

In practice, this is implemented by rendering P_{LD} as splats using z-buffering, with the depth value of a splat at pixel p manually set to the 3D Euclidean distance $d(\mathbf{c}_j, \hat{\mathbf{q}}_p)$ between the splat center $\mathbf{c}_j$ and the point $\hat{\mathbf{q}}_p$. This pass stores at each pixel p (corresponding to the surface point $\hat{\mathbf{q}}_p$) the label of $\mathbf{c}_j$ with $j = \text{argmin}_j\, d(\mathbf{c}_j, \hat{\mathbf{q}}_p)$.

5.2.2 Image Management

In this step, we employ an out-of-core streaming technique [8] for continuously caching the currently most relevant images into a GPU texture array, where the relevance of an image is measured by the frequency of occurrence of its label in the updated label buffer.

5.2.3 Texturing Pass

A full-screen quad is rendered to retrieve the color of each pixel p by projecting $\hat{\mathbf{q}}_p$ onto the image indicated by the updated label buffer.

In a last step, we perform an online screen-space leveling method [8] to balance the color intensities between regions textured by different photographs and thus reduce illumination seams in the final output image.

6 Results

We have tested our approach on three different data sets acquired by a laser scanner (Table 1, Fig. 17).

TABLE 1
Scene characteristics.

Model	# Points	# Images
Hanghaus 2 Wohneinheit 6 (Scene 1)	35M	188
Hanghaus 2 Wohneinheit 1 (Scene 2)	682M	192
Centcelles (Scene 3)	1091M	161

Scene 1 and 2 are scans of different building units in terrace house (Hanghaus) 2 in the excavation of ancient Ephesus, while Scene 3 is a scan of the cupola of the Roman villa of Centcelles. In the following, we discuss performance and quality tradeoffs depending on the algorithm's main parameters, and give a detailed analysis of memory consumption, reconstruction error compared to ground truth, and the convergence of the iterative search. Then, we will compare our approach (denoted by DMRT) to the related depth-map triangulation approach (denoted by DMT) in terms of both quality and performance.

All results in this paper were produced on a PC with an Intel i7-4770K 3.50 GHz CPU, 32 GB RAM and NVIDIA GeForce GTX TITAN GPU. A framebuffer resolution of 1280×720 was used in all our experiments and the accompanying video.

6.1 Performance and Quality Tradeoffs

Number of layers. Currently, we extract layers in a depth-peeling fashion [18], which requires a geometry pass for every single layer. Therefore, the choice of the number of layers is a trade-off between rendering performance and quality. Table 2 shows that, even using more than ten layers, DMRT achieves real-time frame rates. For the measurements in this table, we used a proxy point cloud that is sub-sampled from the original point cloud by a factor of 686 (as in Fig. 17). The table also shows a breakdown of the running time of the algorithm by its stages.

Size of proxy point cloud. In our approach, another key criterion for the rendering performance and quality as well as the labeling time is the size of the proxy point cloud. Fig. 7 shows renderings for different parameters. As expected, the number of layers required for a high-quality rendering decreases with growing sizes of proxy point clouds. For the same layer count on the other hand, a DMRT reconstruction with more proxy points results in an increase of the labeling time and a decrease of the rendering performance.

For more performance results, see Section 6.5.

6.2 GPU Memory Consumption

The GPU memory usage of our method is affected by several factors, including the number of input images, the size of the proxy point cloud, the layer count, and the framebuffer resolution.

For all our test scenes, we generated *depth maps* of size 1024×684. Each map consumes 2.8MB of GPU

Copyright (c) 2015 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TVCG.2015.2430333

$\|P_{LD}\|$	4.4M	1M		0.2M	
labeling in	16.4 min	3.4 min		0.8 min	
# layers	3	3	4	4	7
fps	44	54	48	49	36

Fig. 7. Results for different parameters. The red ellipses indicate regions of some artifacts.

TABLE 2
Average performance of DMRT rendering (in ms) for different numbers of layers, measured during a walkthrough of Scene 2.

# layers	1	3	5	7	9	11
visibility stage	2	6	9.8	13.7	17.5	21.2
raytracing pass	1.2	2	2.3	2.7	3	3.3
labeling pass	2	2	2	2	2	2
img. man.	2.5	2.8	2.9	3.1	3.3	3.3
texturing pass	6.3	6.4	6.3	6.3	6.3	6.3
total	14	19.2	23.3	27.8	32.1	36.1
fps	71	52	43	36	31	28

memory (one float per pixel). As described in Section 5.2.2, the *high-resolution input images* are cached in a GPU texture array on demand. We reserved 1GB of GPU memory for them. We resort to *low-resolution images* (of size 256×171) if input images are not available in the texture array. All of these are stored on the GPU, and each requires 0.13MB. As an example, rendering Scene 2 requires 563MB for the 192 depth maps and low-resolution images.

Furthermore, each point of the *proxy point cloud* is represented by six floats for the position and the normal vector, and an integer for the label. A screen-space pixel in a *layer* requires two floats, one for the depth, and the other for the label. Therefore, an optimal DMRT rendering of Scene 2 with $|P_{LD}| = 1$M at a resolution of 1280×720 and five layers (see the accompanying video) occupies an additional 65MB of GPU memory (37MB for the proxy points and 28MB for the layers).

6.3 Ground-Truth Comparison

In order to analyze the reconstruction error of DMRT, we rendered the scene from the viewpoint of one of the image cameras, and compared the color output and depth buffer at different stages of our rendering pipeline to the original image and its corresponding high-resolution depth map, respectively (Fig. 8). This comparison can give a first impression of the

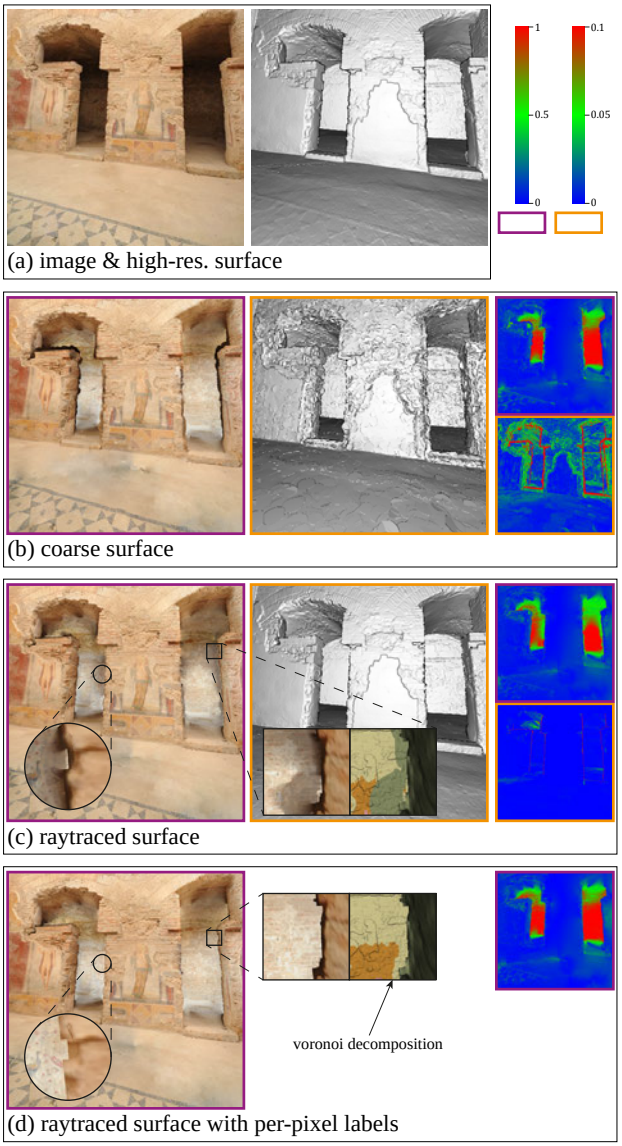

Fig. 8. Analysis of the reconstruction error. The scene is rendered as seen by the image shown in (a). The error is measured as the deviation of the color output and depth buffer at different rendering stages (b)-(d) from the reference image and its corresponding high-resolution depth map, respectively. The color and depth differences are visualized as heat maps shown in the right column.

Copyright (c) 2015 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TVCG.2015.2430333

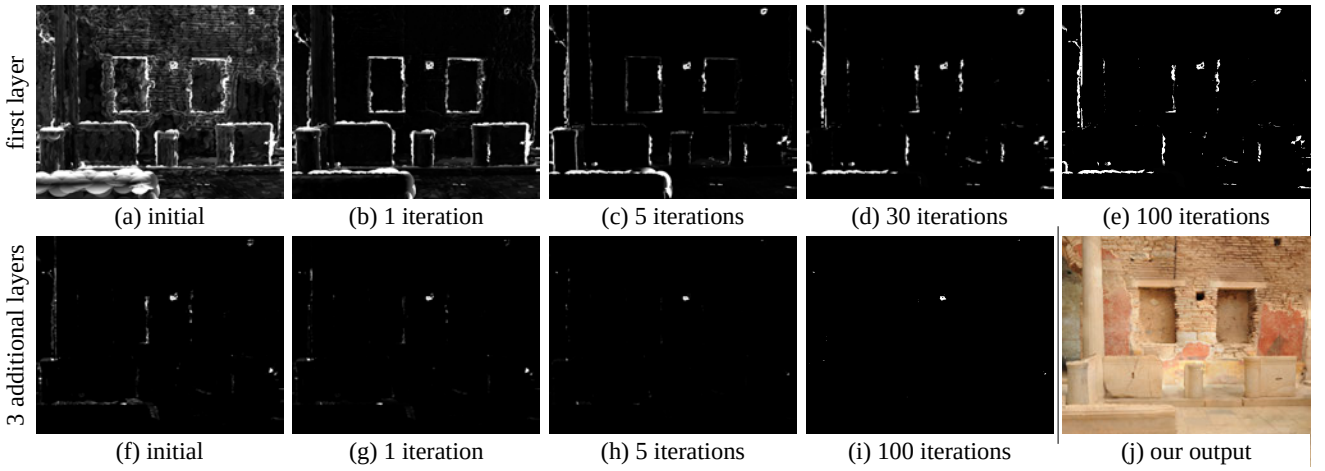

Fig. 9. (a)-(e) show absolute stepsizes of the iterative search in the first layer. For each of the white pixels in (e), where raytracing of the first layer fails, a maximum of three more layers are consecutively searched until a valid intersection is found. (f)-(i) show stepsizes in the layer where raytracing succeeds. Note that for the visualization, all stepsizes are multiplied by 10 and then clamped to the range $[0, 1]$.

reconstruction error, however note that due to different lighting conditions when acquiring the different images, a full match to the "ground truth" is not possible. The rightmost column in Fig. 8 shows color and depth differences as heat maps. For this analysis, we used a sub-sampling factor of 686 ($|P_{LD}| = 1$M) to generate the proxy point cloud, and rendered the scene using four layers. Figs. 8c and 8d show the DMRT reconstruction after a maximum of 100 iterative search and 20 binary search iterations.

A comparison of the heat maps (of depth differences) in Figs. 8b and 8c shows that raytracing reduces the overall depth error. As expected, remaining differences are maximal at oblique angles and silhouettes. However, note that the differences at silhouettes are not generated by our raytracing method. Instead, these occur naturally since the depth map of the image and the raytraced depth maps have different sampling rates of the observed surface, and thus exhibit slight geometric variations at silhouettes.

Interestingly, the overall color error is minimal, except inside the two small rooms. This is because the labeling assigns the points there to images that have better geometric resolution, but were acquired under different lighting conditions than the reference image in Fig. 8a.

Fig. 8c shows that while raytracing resolves the geometry at silhouettes adequately, it generates false labels among these regions by mapping the labels of proxy splats to the background (see also Fig. 6). As we have shown in the accompanying video, these false labels generate ghosting artifacts during animations, and are resolved by our per-pixel labeling step (Fig. 8d).

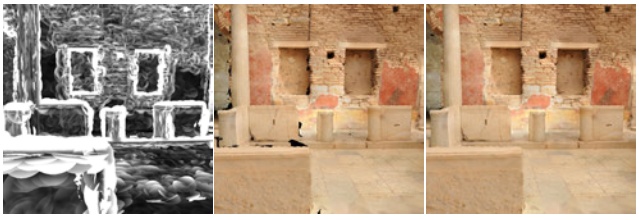

Fig. 10. Worst-case scenario. (a) shows a poor initialization of the stepsizes of the iterative search, therefore requiring many layers for a high-quality visualization. Our output with four (b) and eleven (c) layers.

6.4 Convergence

In this section, we analyze the convergence of the iterative search with adaptive stepsize, which is responsible for finding a "tight" pair of points enclosing an intersection point to seed the binary search. We also discuss the limits of our rendering method for a synthetically generated scene configuration.

We rendered the scene using the same parameters as in Section 6.3. Figs. 9a-9e show absolute stepsizes of the iterative search in the first label layer. For some pixels, our raytracing failed to find intersections in this layer. These pixels are marked white in Fig. 9e, and for each of them, an intersection point is searched in three additional layers. Figs. 9f-9i show absolute stepsizes in the layer where an intersection point is found.

We perform a total of $c_{total} = \sum_{i=1}^{k} c_i$ iterative search iterations for each pixel, where $1 \leq c_i \leq c_{max}$ is the number of iterations performed in the ith layer. The maximum iteration count in each layer is bound by c_{max} (100 in this example), and k refers to the index of the layer where the intersection is found (or the

Copyright (c) 2015 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TVCG.2015.2430333

TABLE 3
Comparison of the labeling times and rendering performance on Scene 2.

DMT	depth-map res.	256×171	512×342	1024×684
	aver. num. of labels per depth map	46		
	labeling times	7 min	23 min	87 min
	min/avg/max fps	32/45/84	11/17/46	3/5/16
DMRT	depth-map res.	1024×684		
	$\|P_{LD}\|$	1M		
	# labels	192		
	# layers	5		
	labeling times	3.4 min		
	min/avg/max fps	30/43/74		

user-defined maximum layer count).

In practice, iterative search converges in a few iterations to an intersection point, if any. Otherwise, it terminates early if an intersection with a discontinuity triangle is found. In our experiment, the iteration count c_{total} was on average 4.6 over all pixels, and it took the raytracing pass 2.4 ms to complete (including the binary search procedure).

The convergence of the iterative search is only guaranteed if each texel along the projection of the view ray onto the depth map is visited, which is slow if the depth-map resolution is high. The iterative search with adaptive stepsize, on the other hand, proved very efficient in practice to find in a few iterations a pair of points enclosing an intersection point.

In order to see the performance of our raytracing for a poor initialization of the stepsizes (Fig. 10a), we multiplied the splat radii by 2.5, and rendered the scene again. In this scenario, the iterative search required on average $c_{total} = 9.5$ iterations per pixel, and the raytracing pass completed in 4.2 ms. Even though our raytracing was still efficient, four layers was not sufficient to obtain a high-quality result (Fig. 10b). To obtain a comparable result (Fig. 10c) as in Fig. 9j, eleven layers were required, and the raytracing pass performed in 8 ms with $c_{total} = 11$ on average. We see that the most performance-critical part of our rendering pipeline is still the extraction of the layers, while searching for intersections in these layers is quite efficient (see also Table 2).

6.5 Comparison to DMT

Finally, we compare our method to the related depth-map triangulation approach on Scene 2. For this comparison, we used a proxy point cloud of size 1M and five layers for the DMRT approach. Our experiments suggest that this configuration is more than sufficient for a not completely artifact-free, but high-quality DMRT rendering. On the other hand, depending on the chosen stitching threshold, DMT can produce severe artifacts (Fig. 11).

Table 3 compares the labeling times and rendering performance of DMT and DMRT for differently sized

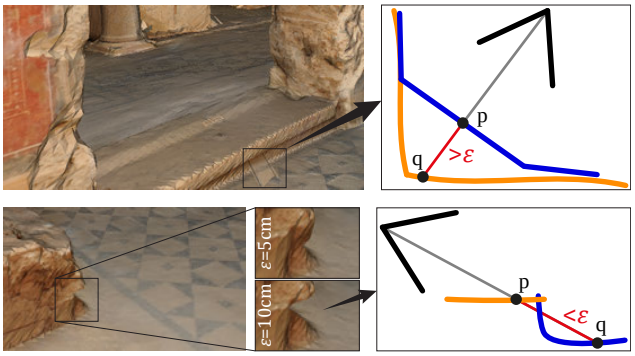

Fig. 11. DMT's stitching artifacts. Top: Due to a small stitching threshold ε, the points $\mathbf{p}$ and $\mathbf{q}$ are considered as non-overlapping by the DMT, leading to the point $\mathbf{p}$ on the low-resolution depth map to be chosen for texturing. Bottom: In DMT, visibility is not resolved for features smaller than the ε threshold. Thus, the invisible point $\mathbf{q}$ can shine through the front surface.

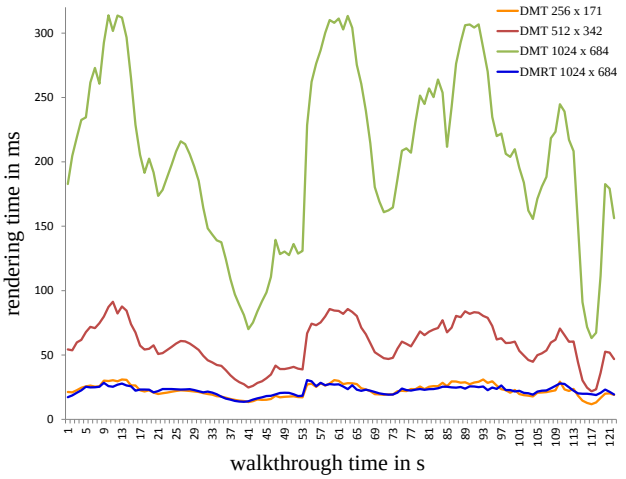

Fig. 12. Comparison of the rendering performance of DMT and our DMRT approach for a walkthrough of Scene 2. Using high-resolution depth maps, our method runs at 43 fps, being on average about an order of magnitude faster than the previous work, which has to settle with a quarter of the resolution to reach this performance.

depth maps. Since the resolution of the depth maps does not have a direct effect on the performance of DMRT, we used the highest resolution for our approach. The table shows that DMT strongly couples the labeling time and rendering performance to the resolution of the depth maps used to represent the scene. If we aim for an *equal-quality* comparison (Figs. 1c right and 1d), DMT needs to label 192 depth maps of size 1024×684, which takes about 26 times longer (87 min) than labeling the 1M proxy points used by DMRT (3.4 min). While DMT cannot render depth maps of this size in real time anymore (5 fps on average), our new raytracing method is about 9 times

Copyright (c) 2015 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TVCG.2015.2430333

(a) one layer (b) two layers (c) three layers

Fig. 13. (a)-(c) show results of raytracing different number of layers along each view ray. Raytracing a single layer (a) produces severe artifacts (black background pixels), especially near silhouettes. In this example, artifacts produced by raytracing two layers (b) are barely visible. Adding a third layer almost completely removes artifacts.

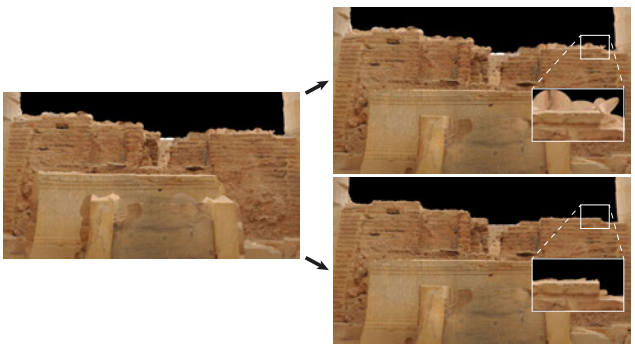

Fig. 14. The left image shows the coarse surface (without raytracing). By using textured coarse positions where raytracing fails, the splats along silhouettes spuriously occlude the background (right top). Therefore, we always discard pixels if no valid intersection with the surface could be found (right bottom).

faster, thus providing a real-time high-quality visualization of the scene (Figs. 1d and 12, see also Table 2). Reducing the depth-map resolution to 256×171 allows DMT to almost match these performance values for labeling and real-time rendering, but noticeably reduces the geometric resolution of the output (Fig. 1c left).

7 Limitations and Future Work

Number of layers We found that extracting a few layers in the visibility stage (Section 5.1.1) is sufficient for high-quality visualizations (Fig. 13). However, in scenes of higher geometric complexity, more layers might be required (e.g., see Fig. 10). At the moment, we use a naive implementation that performs k geometry passes for k layers, which can become inefficient as k increases. In such cases, more elaborate A-Buffer techniques could be incorporated to achieve a multi-layer setup in a single pass [19]. Also, for a few pixels where raytracing fails to find a valid intersection with any of the layers, we show the background color instead of textured coarse surface points (Fig. 14). We

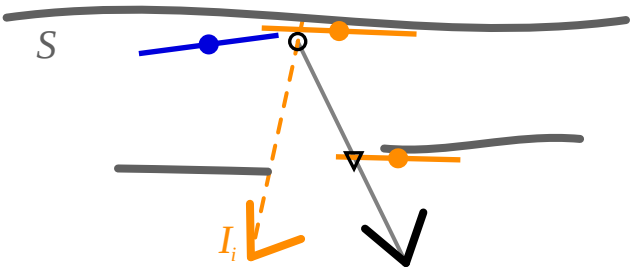

Fig. 15. Sub-sampling issue. The shown view ray intersects a discontinuity edge of I_i. Due to the poor sampling of the surface S by proxy points, there isn't any second layer to search for a valid intersection in this case.

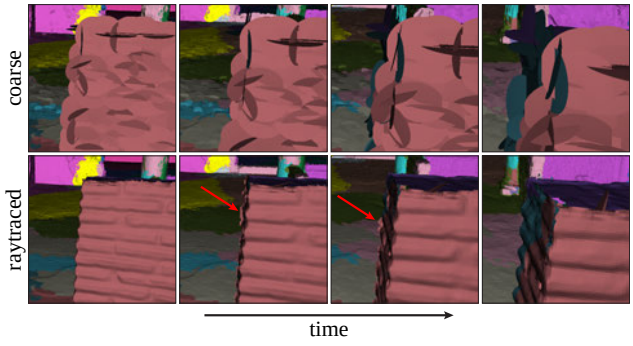

Fig. 16. Label changes under camera motion can lead to view-dependent geometry of silhouettes.

opted for this solution since splats along silhouettes can also occlude the background.

Size of proxy point cloud. As discussed in Section 6.1, the sub-sampling factor is a trade-off between performance and quality. In order to achieve high performance, this factor has to be large enough, but should be small enough to maintain fine surface details. Currently, we discard the highest levels of detail of the input point cloud to obtain proxy points. However, a feature-aware sub-sampling strategy could produce an even better rendering quality, since the generation of the proxy points currently does not take local surface characteristics into account. Fig. 15 illustrates the absence of layers for raytracing, even for a reasonable coverage of the surface by proxy splats.

Motion artifacts. Depth maps can have slightly varying representations of silhouettes based on the viewing angle and distance relative to the observed surface. Thus, label changes under camera motion can lead to raytracing of depth maps with possibly different representations of silhouettes (Fig. 16).

Inherited artifacts. Other rendering artifacts that are inherited from the previous approach [8] are the flickering during animations, and false textures at some silhouettes due to image misregistrations and the noise inherent in point clouds.

Extension. Note that the runtime steps required to

Copyright (c) 2015 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TVCG.2015.2430333

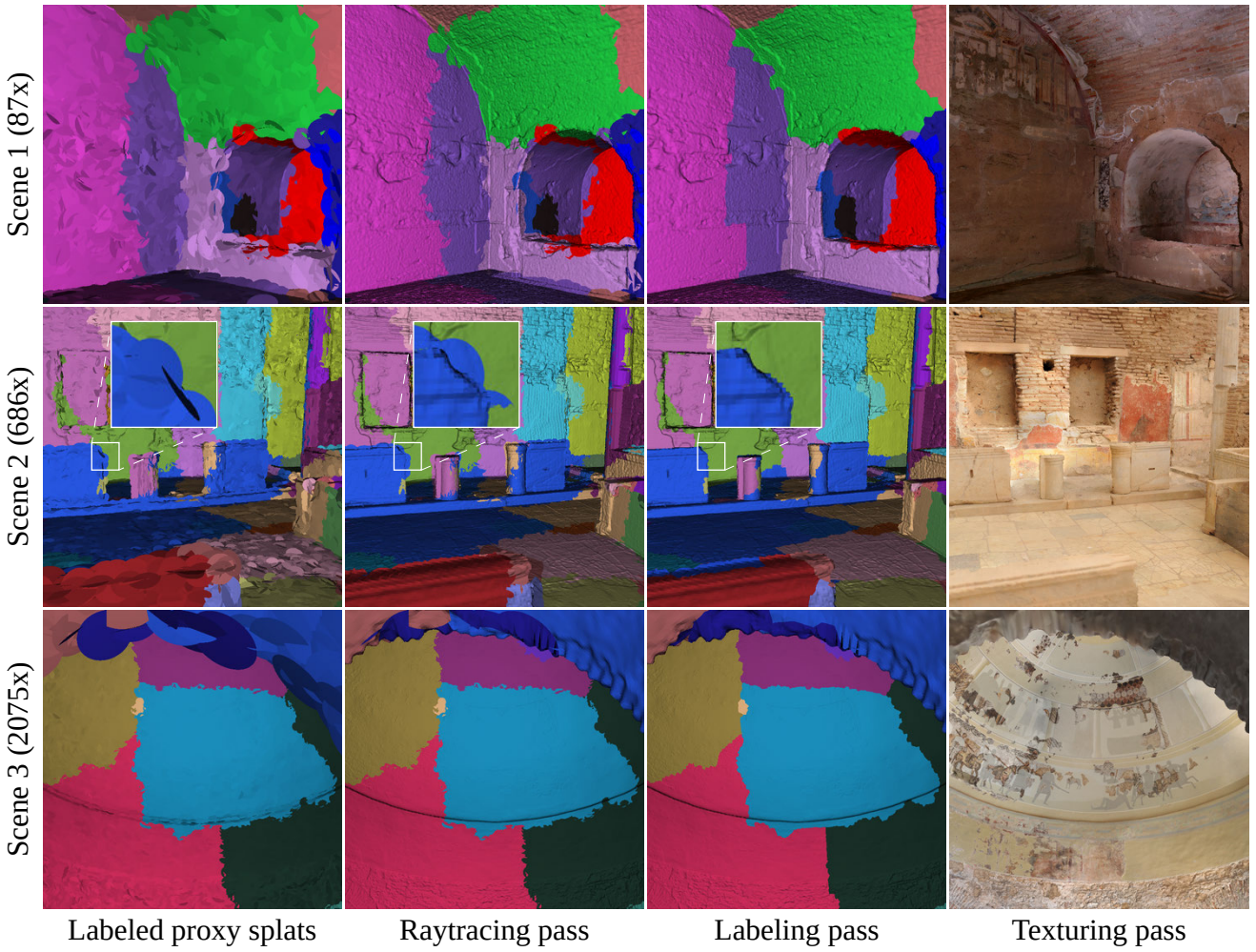

Fig. 17. Results from three data sets. From left to right: Splatted proxy points with (from top to bottom) increasing sub-sampling factors of the original point cloud ranging from $87\times$ up to $2075\times$; raytraced surface without and with per-pixel labels; and textured surface. The insets demonstrate how the labels of the column are mapped to the back wall if the labeling pass is not applied.

create a high-resolution depth buffer (splatting a small number of proxy points (visibility stage) and performing an efficient raytracing in a full-screen pass) are so fast that they could be run twice per frame. This could be used, for example, to create a shadow map for a moving light source, allowing dynamic shadows at interactive frame rates.

8 Conclusion

In this paper, we introduced a novel multi-depth-map raytracing approach for high-quality reconstruction and visualization of large-scale scenes. In a preprocessing, we generate multiple high-resolution depth maps and perform a graph-cut based optimization of the point-to-image assignments (point labels) on a strongly reduced subset of the original point cloud. At runtime, we first reconstruct a high-resolution depth buffer by raytracing these depth maps, where the labels indicate which depth maps to intersect. In a second step, we compute high-quality per-pixel labels from the sparse label information and use these for texturing the depth buffer.

We have shown that our method allows for a real-time visualization of large-scale scenes at much higher geometric resolution than the related state of the art, which is based on rendering and stitching of many depth maps. Our results also indicate a huge performance gain in the labeling step as compared to the previous method.

References

[1] D. T. Guinnip, S. Lai, and R. Yang, "View-dependent textured splatting for rendering live scenes," in *ACM SIGGRAPH 2004 Posters*, ser. SIGGRAPH '04. New York, NY, USA: ACM, 2004, pp. 51–. [Online]. Available: http://doi.acm.org/10.1145/1186415.1186474

[2] D. Sibbing, T. Sattler, B. Leibe, and L. Kobbelt, "Sift-realistic rendering," in *Proc. the 2013 International Conf. 3D Vision (3DV 13)*, 2013, pp. 56–63.

[3] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," in *Proc. the 4th Eurographics Symp. Geometry Processing (SGP 06)*, 2006, pp. 61–70.

Copyright (c) 2015 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TVCG.2015.2430333

[4] M. Kazhdan and H. Hoppe, "Screened poisson surface reconstruction," *ACM Trans. Graph.*, vol. 32, no. 3, pp. 29:1–29:13, June 2013.
[5] V. Lempitsky and D. Ivanov, "Seamless mosaicing of image-based texture maps," in *Computer Vision and Pattern Recognition (CVPR 07), IEEE*, June 2007, pp. 1–6.
[6] R. Gal, Y. Wexler, E. Ofek, H. Hoppe, and D. Cohen-Or, "Seamless montage for texturing models," *Computer Graphics Forum*, vol. 29, no. 2, pp. 479–486, 2010.
[7] M. Bolitho, M. Kazhdan, R. Burns, and H. Hoppe, "Multilevel streaming for out-of-core surface reconstruction," in *Proc. the 5th Eurographics Symp. Geometry Processing (SGP 07)*, 2007, pp. 69–78.
[8] M. Arikan, R. Preiner, C. Scheiblauer, S. Jeschke, and M. Wimmer, "Large-scale point-cloud visualization through localized textured surface reconstruction," *IEEE Transactions on Visualization and Computer Graphics*, vol. 99, no. PrePrints, p. 1, 2014.
[9] M. Botsch and L. Kobbelt, "High-quality point-based rendering on modern gpus," in *Proc. the 11th Pacific Conf. Computer Graphics and Applications (PG 03)*, 2003, pp. 335–343.
[10] M. Botsch, M. Spernat, and L. Kobbelt, "Phong splatting," in *Proceedings of the First Eurographics Conference on Point-Based Graphics*, ser. SPBG'04. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2004, pp. 25–32. [Online]. Available: http://dx.doi.org/10.2312/SPBG/SPBG04/025-032
[11] M. Zwicker, J. Räsänen, M. Botsch, C. Dachsbacher, and M. Pauly, "Perspective accurate splatting," in *Proceedings of Graphics Interface 2004*, ser. GI '04. School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada: Canadian Human-Computer Communications Society, 2004, pp. 247–254. [Online]. Available: http://dl.acm.org/citation.cfm?id=1006058.1006088
[12] M. Botsch, A. Hornung, M. Zwicker, and L. Kobbelt, "High-quality surface splatting on today's gpus," in *Proc. the 2nd Eurographics / IEEE VGTC Symp. Point-Based Graphics (SPBG 05)*, 2005, pp. 17–24.
[13] M. Agrawala, R. Ramamoorthi, A. Heirich, and L. Moll, "Efficient image-based methods for rendering soft shadows," in *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '00. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000, pp. 375–384. [Online]. Available: http://dx.doi.org/10.1145/344779.344954
[14] F. Xie, E. Tabellion, and A. Pearce, "Soft shadows by ray tracing multilayer transparent shadow maps," in *Proceedings of the 18th Eurographics Conference on Rendering Techniques*, ser. EGSR'07. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2007, pp. 265–276. [Online]. Available: http://dx.doi.org/10.2312/EGWR/EGSR07/265-276
[15] C. Zhang, H.-H. Hsieh, and H.-W. Shen, "Real-time reflections on curved objects using layered depth texture," in *IADIS International Conference Proceedings on Computer Graphics and Visualization*, 2008.
[16] A. Reshetov, A. Soupikov, and J. Hurley, "Multi-level ray tracing algorithm," in *ACM SIGGRAPH 2005 Papers*, ser. SIGGRAPH '05. New York, NY, USA: ACM, 2005, pp. 1176–1185. [Online]. Available: http://doi.acm.org/10.1145/1186822.1073329
[17] C. Scheiblauer and M. Wimmer, "Out-of-core selection and editing of huge point clouds," *Computers and Graphics*, vol. 35, no. 2, pp. 342–351, Apr. 2011.
[18] C. Everitt, "Interactive order-independent transparency," NVIDIA, Tech. Rep., 2001.
[19] H. Gruen and N. Thibieroz, "Oit and indirect illumination using dx11 linked lists," in *GDC*, 2010.

Murat Arikan is a Ph.D. student at the Institute of Computer Graphics and Algorithms of the Vienna University of Technology. He received his M.Sc. degree in Mathematics from Vienna University of Technology in 2008. His current research interests are real-time rendering, point-based rendering, and interactive modeling.

Reinhold Preiner received his B.Sc. degree in Computer Science from Graz University in 2008 and his M.Sc. degree in Computer Science from Vienna University of Technology in 2010. His research interests include reconstruction, geometry processing, and interactive global illumination. He is now an assistant professor and doctoral researcher at the Institute of Computer Graphics and Algorithms of the Vienna University of Technology.

Michael Wimmer is an associate professor at the Institute of Computer Graphics and Algorithms of the Vienna University of Technology, where he received an M.Sc. in 1997 and a Ph.D. in 2001. His current research interests are real-time rendering, computer games, real-time visualization of urban environments, point-based rendering and procedural modeling. He has coauthored many papers in these fields, and was papers co-chair of EGSR 2008 and Pacific Graphics 2012, and is associate editor of Computers & Graphics.

Copyright (c) 2015 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.