To appear in IEEE Transactions on Visualization and Computer Graphics

Real-Time View Planning for Unstructured Lumigraph Modeling

Okan Erat, Markus Hoell, Karl Haubenwallner, Christian Pirchheim, Dieter Schmalstieg
Graz University of Technology

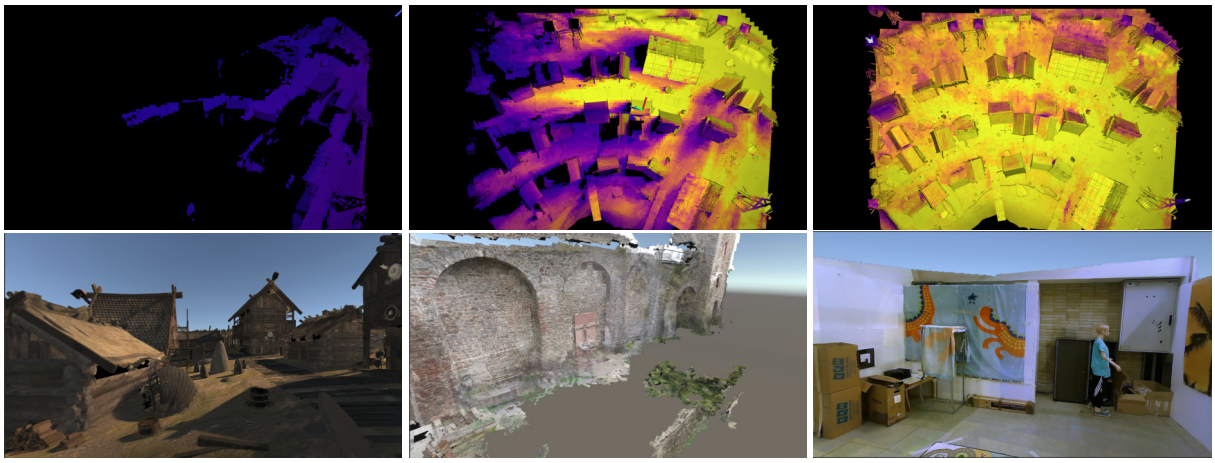

Fig. 1. (Top row) Progression of real-time view planning; unseen parts of the scene are black, and brighter colors (purple to yellow) mean more views covering a portion of the scene. Gray color indicates the surface has been seen but could not be textured. (Bottom row) Images rendered using the unstructured lumigraph.

Abstract— We propose an algorithm for generating an unstructured lumigraph in real-time from an image stream. This problem has important applications in mixed reality, such as telepresence, interior design or as-built documentation. Unlike conventional texture optimization in structure from motion, our method must choose views from the input stream in a strictly incremental manner, since only a small number of views can be stored or transmitted. This requires formulating an online variant of the well-known view-planning problem, which must take into account what parts of the scene have already been seen and how the lumigraph sample distribution could improve in the future. We address this highly unconstrained problem by regularizing the scene structure using a regular grid structure. Upon the grid structure, we define a coverage metric describing how well the lumigraph samples cover the grid in terms of spatial and angular resolution, and we greedily keep incoming views if they improve the coverage. We evaluate the performance of our algorithm quantitatively and qualitatively on a variety of synthetic and real scenes, and demonstrate visually appealing results obtained at real-time frame rates (in the range of 3Hz-100Hz per incoming image, depending on configuration).

Index Terms—Lumigraph,virtual reality,rendering,real-time,view planning,keyframe selection,multi-view.

1 Introduction

Simultaneous localization and mapping (SLAM) can obtain geometric reconstructions in real time. In particular, RGB-D sensors make it easy to obtain dense geometry. The reconstructed scene can be rendered with *free viewpoint* control by the user. Free-viewpoint rendering can instantly be used on incomplete scenes, even while the reconstruction is still ongoing. Apart from feedback for the camera operator, instant reconstruction enables important novel applications of *mixed reality* (MR) [1], such as telepresence [2], physically embedded games, as-built documentation, interior design, or preview for videography.

However, instant reconstruction is still an emerging technology and requires further technical improvements. In particular, while real-time *geometric reconstruction* is very mature, real-time *photometric reconstruction*, i.e., acquiring surface colors and textures, is often only treated as a byproduct of geometric reconstruction. A typical approach is to cache a single, averaged color, either per voxel of a volumetric model or per mesh vertex. Such a low-pass filtered representation is often not enough to preserve high-quality appearance.

• *E-mail: okan.erat, hoell, karl.haubenwallner, pirchheim, schmalstieg@icg.tugraz.at*

Many techniques for photometric reconstruction exist, but most are designed for *offline* use. Approaches that run for minutes or hours [3–5] and require unbounded memory are not compatible with MR requirements. In particular, global methods depend on scene complexity and are generally unable to maintain strict bound on computation times.

As an alternative to global optimization, photometric reconstruction can harvest the rich input image stream directly in the form of image-based modeling and rendering (IBMR). For best results in larger scenes, IBMR is usually combined with a dense geometric model, yielding a variant of a lumigraph [6]. Table 1 summarizes a typical IBMR pipeline, which can roughly be divided into two stages, modeling and rendering.

1. The *modeling* stage is most often run offline and executed jointly with geometric reconstruction. IBMR modeling consists of four sub-stages: *View capturing* obtains the input images. *View planning* decides which *views* (images annotated with camera pose) from an input image stream are kept. If a view is kept, *view refinement* tries to rectify any shortcomings of the view data, such as improving the spatial registration with the scene geometry or correcting of color and brightness artifacts. *View pre-selection* is concerned with associating views with portions of the scene geometry for later rendering. Note that, in this stage, view selection must be done without knowing the actual viewpoint of novel view that will be requested later.

2. The *rendering* stage always runs at the target frame rate. Its purpose is to generate, per frame, a novel view for a user-controlled viewpoint. Rendering consists of two sub-stages, view selection and view blending. *View selection* performed during rendering selects appropriate views to sample guided by the novel viewpoint. It can operate per frame or per rendered fragment. Finally, *view blending* determines the weights given to individual views for a rendered fragment.

Throughout this pipeline, the number of views considered is reduced continuously, to make the computational effort of subsequent stages tractable. However, for an operational IBMR system, only capturing and blending are mandatory, while the intermediary stages (planning, refinement and selection) are optional. Many systems assume that a reasonable set of views has been captured in advance and concentrate only on the rendering stage, i.e., selection and blending. Another family of techniques attempts to synthesize the best possible view-independent texture map from the available views using offline optimization, concentrating on refinement and selection. In this case, the rendering trivially reduces to conventional, view-independent texture-mapping. There are a few end-to-end IBMR systems, which consider the entire pipeline, but they typically assume sufficient storage for all views. Hence, they have no need for view planning.

In real-time applications, the requirements are different. For instance, in a telepresence application, we would like an IBMR model from an exploring camera to be instantaneously usable in a free-viewpoint rendering at a remote site. Hence, the input stream must be processed immediately to obtain a photometric reconstruction. Even if we allow a small delay, such as a few seconds or even tens of seconds, for new photometric data to become available to IBMR, we cannot afford to unconditionally cache all images from an input stream and process them later, since we will never catch up. Instead, we must be able to process them at frame rate.

For this purpose, view planning becomes mandatory. We are not the first to consider IBMR view planning, but early work in this area [7, 8] only intended to cover each scene primitive in a *single* view, thus solving a variant of the *art gallery problem* [9], which – like other set cover problems [10] – are NP-hard. In contrast, we carefully select *multiple* views to build an *unstructured lumigraph* [11]. Therefore, our generalized view plan has three simultaneous requirements:

1. Planning must consider coverage of every primitive in every view.
2. Planning must operate strictly incremental on an input stream.
3. Planning plan must operate within bounded memory.

In this paper, we address these requirements by incrementally filling a *view store* of bounded size for arbitrarily large scenes with thousands of input views. We organize the scene using a regular discretization, similar to the one used by Hedman et al. [3]. However, since our plan must be constructed online and in bounded memory, we cannot rely on any global optimization procedures. Instead, for each new view, we compute a *coverage* score and use it as a cache replacement metric. Our method also performs view-processing, correcting for erroneous viewpoints and exposure differences between views. Moreover, we also describe how view pre-selection can be formulated using the coverage score. Finally, we demonstrate that our view planning can run at interactive rates and delivers IBMR quality comparable to IBMR using an unbounded number of views, both for synthetic scenes and for real-world video sequences. Thus, our work is the first to solve a general view planning problem for IBMR in real time and bounded memory.

2 Related work

Computer graphics models generated by reconstruction from image or depth sensors can be organized according to the amount of geometry and photometric data they contain. Pure geometric models do not contain any photometric information, while pure lightfields [12, 13] do not contain any surface geometry.

Since the most widespread computer graphics model consists of a polygonal surface mesh covered by a texture map, many reconstruction methods aim at creating this type of model. Real-time methods, such as volumetric fusion from RGB-D sensors [14], usually cache averaged colors, either per surface point or, if a mesh is extracted, per vertex. Color averaging often leads to a loss of contrast and a blurry appearance, even if colors are cached densely on the surface [15].

Stage	Time unit	No. of views considered
Modeling		
View capturing	Per input image	Infinite
View planning	Per input image	Fixed upper bound
View refinement	Per novel view	Fixed upper bound
Rendering		
View selection	Per stored view	Only close-by views
View blending	Per fragment	Only views with weight> 0

Table 1. Image-based rendering pipeline

Offline methods which assume that detailed geometry has already been recovered concentrate on extracting an optimal texture map from a set of input images (views) covering the geometry [4, 5, 16, 17]. However, even with perfect registration of views to geometry, baking the image information from multiple views into a single texture map destroys view-dependent aspects of surface materials.

Image-based rendering (IBMR) gives a stronger emphasis to photometric information by subsampling a view-dependent plenoptic function [18]. For small objects or scenes, pure IBMR methods rely exclusively on densely sampled views, while replacing detailed scene geometry with crude proxy geometry (e.g., a single plane or a sphere). For such *outside-in* scenes, the possible viewpoints are typically restricted to an orbit around the object or even a narrow zone inside such an orbit. A drawback of pure IBMR methods comes from their excessive storage requirements for larger scenes, where free-viewpoint navigation is desirable. For such *inside-out* scenes, better storage efficiency can be obtained by combining more detailed geometry with sparser views.

Thus, *geometry-based IBMR* methods, such as the lumigraph [6], combine a proxy geometry with sampled views. With view-dependent texture mapping [19] we can build an unstructured lumigraph [11], which can render new views from sparse and irregular view sets without special preprocessing. However, poorly registered views may lead to blurriness and ghosting artefacts. A lot of research has addressed better registration, for instance, by using floating textures [20] or by resampling input views into surface lightfields [21] indexed per surface point instead of per viewpoint.

Forward-projection IBMR replaces global geometry with local geometry per input view, for example, from superpixel segmentation [22, 23] or local structure from motion [3]. These methods can better suppress artifacts stemming from sparse or incomplete geometry.

Besides registration, a second challenge for geometry-based IBMR methods is that they do not trivially scale to a large number of views. This implies that the acquisition of views, or *view planning*, deserves specific attention in IBMR systems. If a finite number of targets views is known in advance, view planning can aim to optimally cover these views [24, 25]. If the target views can be assumed to lie within a bounded "walking" range, view planning becomes a variant of the *art gallery problem*, aiming to cover every surface point in the scene with at least one view [7, 8].

Unfortunately, texturing with a single view is not sufficient for high quality IBMR. It is more meaningful for modern IBMR to consider a form of reprojection error between sampled views, to decide if an additional view should be acquired. For instance, recent work [3] has proposed offline view planning for larger scenes by subdividing the scene into parts and establishing an explicit mapping between views and scene parts. However, this work does not place an upper bound on the size of the view cache, and does not run in real time. This makes it unsuitable for interactive reconstruction [26] or telepresence [27–29].

View planning for outside-in scenes has been demonstrated at interactive rates to guide a user's acquisition with a handheld camera [26]. Our work has the additional requirement that view planning needs to consider surrounding geometry of inside-out scene and not only the

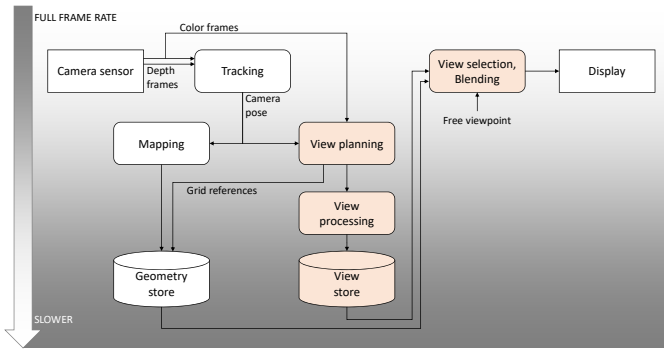

Fig. 2. We extend SLAM with lumigraph modeling and rendering including a novel real-time view planning approach. This diagram shows existing components in white and new components in orange.

orbital space around an outside-in scene.

While relatively little work on view planning for lumigraphs exists, the problem received more attention in multi-view reconstruction [30]. Here, the main objective is scalability: Naive structure from motion in large image collections suffers from quadratic runtime requirements, when every input image is matched with every other input image. A typical approach to reduce the computational complexity is to build a reconstructed model incrementally, by greedily adding views that promise the fastest improvement in model coverage and quality [31, 32]. The metrics employed in selecting new views to be added are partially similar to metrics used in texture map optimization, such as view disparity, occlusion, and distance to the scene [33]. However, the emphasis lies on geometric matching, for example, by tracing silhouettes, not so much on photometric matching. Moreover, these methods are usually employed offline, often in multiple passes [34], assuming full access to all input views simultaneously.

In contrast, view planning in robotics needs real-time performance to support active vision [35]: Given a partial reconstruction, an autonomous robot is instructed to navigate to the position corresponding to a next best view, which promises the best improvement of the scene reconstructed so far [36,37]. Robotic view planning needs to deliver not only a next best view, but also a path plan to navigate there safely [38]. However, real-time constraints are somewhat relaxed, if the robot can simply remain stationary until the plan becomes available. If this is not possible, for example, in case of aerial vehicles [39], the entire plan must be computed in offline phases interleaved with flight phases [40].

Another variant of active vision is to give the user control over the camera, either using a handheld camera [26] or a remote-controlled aerial vehicle [41]. In either case, the user must receive instant feedback on the state of the acquisition to determine if there is sufficient coverage already. Our approach belongs to this category, although we concentrate on the planning algorithm and not on interactive user feedback. The difference over previous work is that we retain real-time performance for optimal view planning of a lumigraph embedded in a 3D scene, and, not just a point cloud or spherical lightfield.

3 Method

We propose a new approach to build an IBMR pipeline around instant modeling of a sparse, unstructured lumigraph. We take advantage of the inherent multi-threading of SLAM [42]. SLAM typically uses a tracking task running at full frame-rate and a slower mapping (i.e., geometric reconstruction) task. We introduce additional tasks for photometric registration and novel view synthesis, which run at their own, independent rates.

Figure 2 shows an overview of our system architecture: White boxes belong to the existing SLAM system, which fills the geometry store. Red boxes describe our novel photometric registration. Every input frame is considered by the view planning component as a potential novel view to be placed in the view store, which is also associating novel views with the scene geometry.

A lumigraph rendering component is responsible for generating novel views. Using the information from the geometry and view stores, this component performs view selection and blending to generate a novel view to display. The framerate of the rendering component is decoupled from the lumigraph modeling, and rendering can even run remotely in a telepresence environment.

We begin by reviewing a basic unstructured lumigraph (section 3.1), as described by Buehler et al. [11]. Then, we explain our core contribution, the real-time view planning algorithm for the lumigraph (section 3.2). We also discuss how the views selected for the lumigraph can be refined in real time (section 3.3), and, finally, how the runtime view selection works (section 3.4).

3.1 Basic lumigraph blending

Basic lumigraph blending creates novel views by sampling views from a *view store* $F = \{f_i\}$[1]. Each view $f_i = (c_i, e_i, z_i)$ consists of a camera pose c_i, decomposed into camera position $\mathbf{Pos}(c_i)$ and viewing direction $\mathbf{Dir}(c_i)$, as well as a color image $e_i(\mathbf{x})$ and a linear depth image $z_i(\mathbf{x})$.

Lumigraph blending selects, for each sample position $\mathbf{X}$, the best K views (collected in F_K) according to a weighting function w_i and blends them into a new view $f_o = (c_o, e_o, z_o)$.

Using a projection operator $\mathbf{pr}(\mathbf{X}, f)$ from world space to image space and an inverse projection operator $\mathbf{P}(\mathbf{x})$ from image space to world space, we obtain e_o as follows:

$$e_o(\mathbf{x}) = \sum_{f_i \in F_k} e_i(\mathbf{pr}(\mathbf{P}_o(\mathbf{x}), f_i)) \cdot \frac{w_i(\mathbf{x})}{\sum_{f_r \in F_K} w_r(\mathbf{x})}$$

The weights w_i are obtained by combining terms that describe the geometric (i.e., directional and positional) similarity of a reference view to the novel view. The directional term w_i^{Dir} is described using a clamped cosine of the angle between the reference view and the novel view ($\mathbf{Nrm}()$ denotes vector normalization).

$$w_i^{Dir}(\mathbf{x}) = \max(0, \mathbf{Nrm}(\mathbf{Pos}(c_o) - \mathbf{P}_o(\mathbf{x})) \cdot \mathbf{Nrm}(\mathbf{Pos}(c_i) - \mathbf{P}_o(\mathbf{x})))$$

The position term w_i^{Pos} is described as the ratio of distances to the camera center:

$$w_i^{Pos}(\mathbf{x}) = \max\left(0, 1 - \frac{|\mathbf{Pos}(c_o) - \mathbf{P}_o(\mathbf{x})|}{|\mathbf{Pos}(c_i) - \mathbf{P}_o(\mathbf{x})|}\right)$$

In addition to the geometric similarity, the weight must also ensure that a sample is valid. Buehler et al. [11] only consider that a sample $\mathbf{X}$ must be within the field of view covered by f (assumed to have an opening angle of 2α). This suffices if the geometric model is very simple, but, for complex geometric models, we must additionally take care that a sample $\mathbf{X}$ is not occluded in f and that $\mathbf{X}$ is not closer than Δ_{xy} to a depth discontinuity larger than Δ_z [28], which would make $\mathbf{X}$ unreliable. We combine these constraints in a validity function $valid(\mathbf{X}, f)$ as follows:

$$valid(\mathbf{X}, f) = \begin{cases} 0, \text{ if } |\mathbf{X} - \mathbf{Pos}(c_f)| > z_f(\mathbf{pr}(\mathbf{X}, f)) \\ 0, \text{ if } \mathbf{Nrm}(\mathbf{X} - \mathbf{Pos}(c_f)) \cdot \mathbf{Dir}(c_f) < \cos\alpha \\ 0, \text{ if } \max\limits_{\Delta_{xy} \in SW(2)} |z_f(\mathbf{pr}(\mathbf{X}, f)) - z_f(\mathbf{pr}(\mathbf{X}, f) + \Delta_{xy})| > \Delta_z \\ 1, \text{ otherwise} \end{cases}$$

Here, $SW(U) = \{\Delta_{xy} \in Z \times Z, |\Delta_{xy}| \leq U\}$ is a radial search window in image space. The final weight is obtained by linearly combining the geometric similarity and the validity function using a parameter λ:

$$w_i(\mathbf{x}) = valid(\mathbf{x}, f_i) \cdot (\lambda \cdot w_i^{Dir} + (1 - \lambda) w_i^{Pos})$$

[1] We denote scalars in italics, 2D Euclidean vectors in lowercase boldface and 3D Euclidean vectors in uppercase boldface.

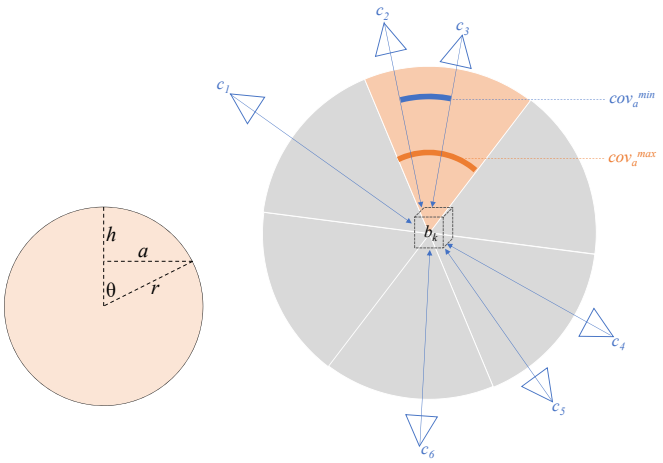

Fig. 3. (left) By dividing the area of a unit sphere, 4π, by the number of views, we obtain a maximum area per view. The area assigned to one view has as an upper bound the area of a sphere cap, $2\pi(1-\cos\theta)$. (right) Example for angular coverage of b_k observed by six views $c_1,\ldots,c_6$. The angular coverage weight is related to the ratio of the minimal observed angle between two cameras, cov_a^{min}, and the maximum possible angle between two cameras, cov_a^{max}.

3.2 View planning

The above description of basic lumigraph blending assumes that F is small enough so all views can be stored and searched at runtime. The key contribution of our paper is the introduction of a real-time solution for *view planning*, which addresses two requirements not handled by basic lumigraph blending: first, choose views from the incoming image stream to store in F, second, obtain a pre-selection so view blending can be done with a constant effort that is independent of the chosen size of F.

Our view planning approach extends the frame store used in lumigraph blending with an additional view-independent data structure: We organize the scene into a regular grid $B=\{b_k\}$, which subdivides the scene geometry G into cells $G(b_k)$. Per cell b_k, we store a set of R^{max} references to F, denoted as $R(b_k)=\{f_{k,r}\}, 1\le r\le R^{max}$.

The cell structure has a number of advantages: It reduces the overall effort compared to processing surface geometry explicitly, it exploits spatial locality, and it decouples the lumigraph from the detailed surface geometry reconstruction. Only during the final rendering are the views associated with individual surface points through indirect texture lookups. The geometric and photometric reconstructions can evolve independently, making our approach robust to variations in computational load and other unforeseen challenges that may occur in a real-time system. For example, new views can be incorporated to refresh the view store after changes to scene geometry or incident illumination. Moreover, cells naturally correspond to blocks of a sparse volumetric data structure, which is now commonly used for large scenes [43].

Coverage metric In order to fill the view store with the best views, we define a *coverage* metric that expresses the benefit of new view in covering the lumigraph. We weight two quality criteria, each expressed by a factor in the range [0,1], which can be seen as a view-independent variant of the directional and positional similarity described above:

- **Directional coverage** cov_d: Views observing a cell should be well distributed in the cell's orientation space, so that every new view covers a portion of the scene from a new angle.

- **Positional coverage** cov_p: Views should have the desired pixel density (neither too dense nor too sparse). Moreover, the view should see as much as possible of the surface inside the cell.

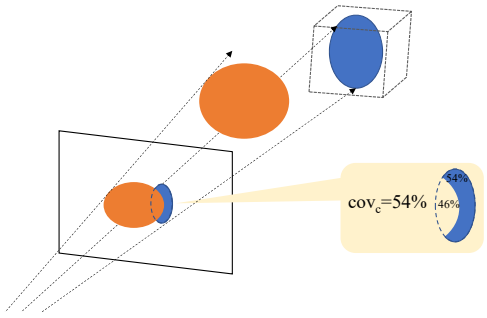

Fig. 4. A view's contribution to a cell is weighted by the relative fraction of the cell's visible surface in the view.

We compute the overall coverage cov as a weighted sum of directional and positional coverage:

$$cov(f,b_k)=\lambda\cdot cov_d(f,b_k)+(1-\lambda)\cdot cov_p(f,b_k)$$

Directional coverage. We determine the minimum angular deviation cov_d^{min} to all views already selected for a particular cell (represented by its centroid, $\mathbf{Pos}(b_k)$):

$$\gamma=\max_{f_i\in R(b_k)}\mathbf{Nrm}(\mathbf{Pos}(c_f)-\mathbf{Pos}(b_k))\cdot\mathbf{Nrm}(\mathbf{Pos}(c_i)-\mathbf{Pos}(b_k))$$

$$cov_d^{min}(f,b_k)=\delta(|R(b_k)|,0)+(1-\delta(|R(b_k)|,0))\cdot\gamma$$

Here, δ denotes the Kronecker delta function. If more views are stored in $R(b_k)$, the angle between them must become smaller. For R^{max} views (all references are filled), the Tammes number denotes the maximum angle $\theta(R^{max})$ between views [44]. We can estimate an upper bound (Figure 3, left) to the cosine of the Tammes number as $cov_d^{max}=\cos(\theta(R^{max}))=2\cdot(1-2/R^{max})^2-1$. Therefore, we obtain an approximate angular coverage cov_d from the minimal observed angle between views and the largest possible angle between views (Figure 3):

$$cov_d(f,b_k)=\min\left(1,\frac{1-cov_d^{min}(f,b_k)}{1-cov_d^{max}}\right)$$

Positional coverage. Positional coverage combines a term judging the resolution of f with respect to b_k and a term describing what fraction of b_k is visible. The distance of a cell to the view is given by d:

$$d(f,b_k)=|\mathbf{Pos}(c_f)-\mathbf{Pos}(b_k)|$$

We express how well the distance of a view matches an ideal distance d_{max} using a Gaussian g with variance σ^2, centered around the ideal distance d_{max}.

$$g(f,b_k)=\exp\left(-\frac{(d(f,b_k)-d_{max})^2}{2\sigma^2}\right)$$

The second term, the visible fraction of a cell (in pixel area units A_{px}), is determined as the ratio of the visible pixel count, pix, to the total pixel count. The visible pixel count is obtained by rendering a position buffer $\mathbf{P}(\mathbf{x})$ and obtaining a cell id cid. The cell id is generated by quantizing $\mathbf{P}$ with a factor q and computing a spatial hash, such that $k=cid_i(\mathbf{x})=hash(\lfloor\mathbf{P}_i(\mathbf{x})\cdot q\rfloor)$ if $\mathbf{x}\in b_k$. Using cid, we can easily determine a visible pixel count pix_{vis} per cell, i.e., the visible pixels inside b_k:

$$pix_{vis}(f,b_k)=\sum_{\mathbf{x}\in f}\delta(cid_f(\mathbf{x}),k)$$

The total pixel count is obtained by projecting the total surface area $A(b_k)$ of the scene geometry contained in cell b_k from world space into f (Figure 4). To avoid an exaggerated influence of very densely or very sparsely populated cells, we constrain the value to lie in the interval

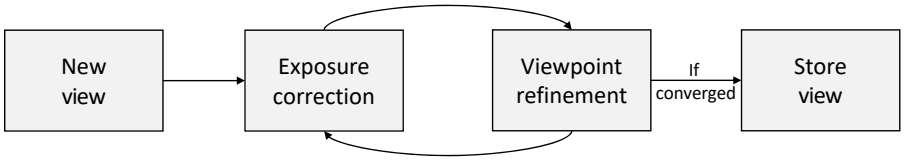

Fig. 5. Before a new view is accepted into the view store, exposure and viewpoint are refined in an alternating optimization.

Fig. 6. Quality comparison of "city wall" scene, from left to right: ground truth, lumigraph rendering with view processing (pose and exposure refinement), lumigraph rendering without view processing. In fthe latter case, blurriness and exposure differences reduce image quality.

from one pixel, A_{px} to the projection of a cell face area A_{cell} into the view. Then, we convert from world-space area into pixel area units by normalizing with A_{px} to obtain a total pixel count pix_{total}.

$$pix_{total}(f,b_k) = \frac{1}{A_{px}} \cdot \max\left(\frac{\min(A(b_k),A_{cell})}{d^2(f,b_k)}, A_{px}\right)$$

Now we weight the pixel count by distance quality and cell coverage to obtain the positional coverage

$$cov_p(f,b_k) = g(f,b_k) \cdot \frac{pix_{vis}(f,b_k)}{pix_{total}(f,b_k)}$$

Candidate view evaluation. We use the coverage metric $cov()$ to decide if we add a candidate view f to F or not. To this end, we seek to increase the summed coverage $scov$ over all views and cells. Per cell, the coverage is clamped by a constant $fcov^{max}$ to avoid a bias towards cells covered by many views.

$$fcov(F,b_k) = \sum_{f_i \in F} cov(f_i,b_k)$$
$$scov(F) = \sum_k \min(fcov(F,b_k), fcov^{max})$$

We keep f, if it improves the coverage by at least Δ_{scov}, i.e., $scov(F \cup f) > scov(F) + \Delta_{scov}$. When the view store is full, f must replace a victim v. By replacing every existing view f_i with f and finding the optimal coverage, we determine the victim $v = \arg\max_{f_i \in F} scov(F \cup f \setminus f_i)$. For efficiency, we keep the views in a list sorted by coverage, and only consider victims that are among the lowest-ranking views. If no victim v can be found such that $scov$ is increased by at least Δ_{scov}, f is not kept.

3.3 View processing

Before we store a candidate frame f, we must match its exposure to the existing views and ensure that its viewpoint is as accurate as possible, so that ghosting artifacts resulting from reprojection errors are minimal (compare images in Figure 6).

Using a subset of surface points $\mathbf{X}_j$ with $valid(\mathbf{X}_j, f) = 1$ as sample points [45], we determine an exposure correction factor E_f. The exposure correction scales a new view f such that it best agrees with the median value of the other views $f_i \in F$ (each scaled with an exposure correction factor E_i) according to a robust metric m (such as the Tukey estimator). We use the median for robustness against outliers that come from observing specular reflections. The energy function $J(f)$ describes the agreement among measurements:

$$J(f) = \sum_j m\left(E_f \cdot e_f(\mathbf{pr}(\mathbf{X}_j, f)) - \underset{i}{\text{median}}(E_i \cdot e_i(\mathbf{pr}(\mathbf{X}_j, f_i))\right)$$

By minimizing $J(f)$, we obtained the desired exposure factor $E_f = \arg\min_{E_f} J(f)$. After obtaining an initial estimate for the exposure correction of f, we rectify small errors in the viewpoint associated with f by making small changes to the external camera parameters, $\mathbf{Pos}(c_f)$ and $\mathbf{Dir}(c_f)$ and recomputing $J(f)$. A search for a local minimum of $J(f)$ using the method of Farnebäck [46] determines the optimal camera pose $c_f = \arg\min_{c_f} J(f)$. We optimize exposure compensation and pose correction until convergence [47] (Figure 5). If the geometric reconstruction used as input contains views with pose outliers, the optimization can get stuck in a local minimum that can be detected by thresholding the residual error; such outlier views are discarded.

3.4 View selection

View selection is composed of two parts, a pre-selection part computed every time a new view is accepted into the view store, and a final selection part executed in the fragment shader during view blending.

Pre-selection fills the references $R(b_k)$ when a new view f arrives. We add f to $R(b_k)$ if the cumulative coverage per cell is at least increased by Δ_{fcov}, i.e., we make sure that $fcov(R(b_k) \cup f) > fcov(R(b_k)) + \Delta_{fcov}$. If $R(b_k)$ is full, we determine a victim to be replaced with f in $R(b_k)$ as $v_k = \arg\max_{f_i \in R(b_k)} fcov(R(b_k) \cup f \setminus f_i)$. If $R(b_k)$ is not yet full, $\Delta_{fcov} = 0$.

During view blending, the fragment shader iterates over the $R(b_k)$ and uses the validity function to determine if a particular view should contribute to the lumigraph at the given location or not. Out of the remaining views, the $K = 3$ best ones are used to determine the color of a pixel as described in section 3.1.

4 Evaluation

We integrated a prototype view planner into the Unity3D game engine on a desktop computer (CPU: Intel i7-5820K 3.30GHZ, GPU: Nvidia GTX 1080Ti). For best performance, the entire pipeline is execute directly on the GPU, using HLSL shaders, while interface programming was done in C# for easy testing. The implementation expects a triangle mesh and an image sequence, annotated with camera poses, that can come from any (real-time or non-real-time) reconstruction algorithm. This allows us to conveniently test our system with a variety of reconstruction engines. We acquired the following four test scenes:

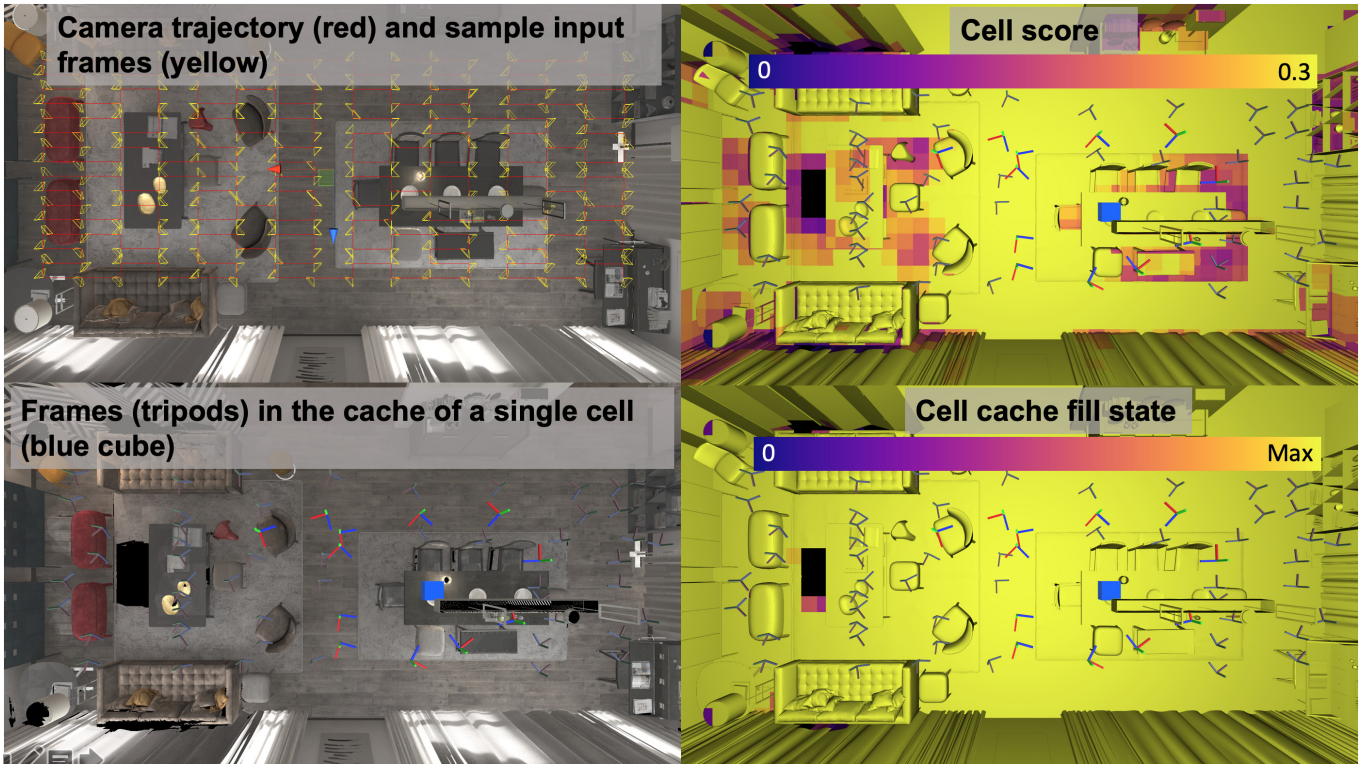

Fig. 7. By using a trajectory following a Hilbert curve for generating the input image sequence in the apartment scene, we achieve a progressive, but homogeneous coverage (Top left). The coverage per cell is visualized (Top right). The homogeneous color indicates that all cells receive a relatively equal coverage in the view store. Note that to increase visual contrast at low scoring cells, scores has been clamped at 0.3. Coordinate glyph marks the camera pose of the views referenced by the blue cell in the middle (Bottom left). The number of references stored per cell is visualized (Bottom right). Most cells fill all their references (yellow), while only a few inaccessible areas do not get covered properly (dark purple).

1. Viking village: synthetic scene with mostly diffuse materials
2. Apartment: synthetic scene with specular materials
3. City wall: real scene reconstructed using the multi-view stereo algorithm of Fuhrmann et al. [48]
4. Lab: real scene scanned with an RGB-D camera and geometrically reconstructed using InfiniTAM [49] [43]

For the synthetic scenes, we generated 5000 input images by densely sampling a trajectory created by 2D Hilbert curve as shown in Figure 7. For the real scenes, we used the original image sequences as input to our view planning algorithm.

We used this setup to analyse how various system parameters influence the results. We evaluate visual quality by comparing rendered views to reference images, reporting image quality as mean SSIM [50], or MSSIM, for 200 test views taken with random camera poses. Finally, we compare image quality of our method to a state-of-the-art real-time method and a commercial offline method.

4.1 Coverage computation

Our first evaluation of the view planning method focuses on the coverage computation. Ideally, view references of a cell should already be full, before we run out of space in the view store.

To understand how much geometry is often (or never) seen in any view, we implemented an in-engine visualization tool which color-codes various aspects of the cells or the contained geometry (Figure 7), such as the number of views observing the cell, the number of registered views, the average coverage per registered views and the directional coverage. For example, it distinguishes geometry that is not seen by any input view from geometry that is seen in an input view, but not selected for the view store. Additionally, in order to assess spatial behavior of our per-cell view planning, we visualize the selected and rejected views as small coordinate axis icons around a particular cell (Figure 7). These visualizations are generated in real-time and could be used during actual scanning, for example, with a handheld RGB-D sensor.

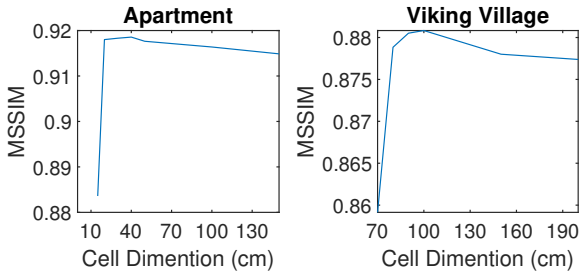

Fig. 8. Effect of cell size on MSSIM. We observe the optimal cell distance depends on the scene diameter, in our examples, 30-100cm.

4.2 Trade-off between positional and directional coverage

We studied how blending of directional and positional coverage terms effects our rendering quality over two different types of trajectories. Using viking village, we compared a Hilbert curve fly-over trajectory (simulating a drone) to a walking-like trajectory with images taken at human eye-level. Each trajectory consisted of 5000 images overall.

Unlike the fly-over, occlusion varies significantly during the walking trajectory. One image may look down an entire street, while another one is complete occluded by a building. The walking trajectory benefits more from an increased weight given to positional coverage (Figure 12).

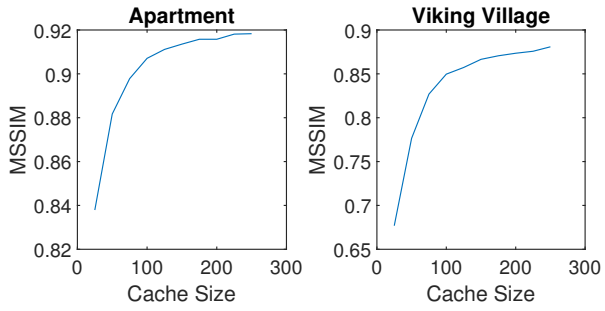

Fig. 9. Diminishing returns in terms of image quality can be observed at reasonable caches sizes of 100-200 frames. MSSIM above 80% typically yields subjectively high image quality.

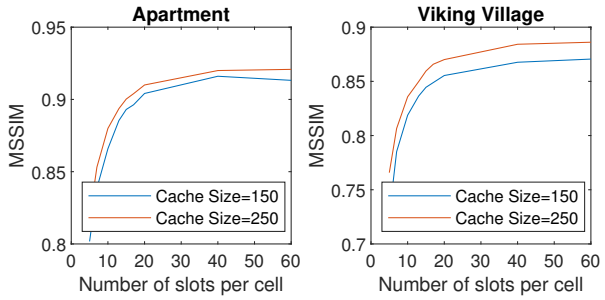

Fig. 10. Effect of number of view references per cell on MSSIM. Diminishing returns can be observed after 15-20 view references.

4.3 Cell dimension

We systematically varied the cell dimension and investigated how it influences the quality. Quality was measured as SSIM between images rendered using our system and reference views, averaged over 200 randomly chosen camera poses per test scene. Consequently, we report MSSIM over all views. We varied cell dimension depending on the overall scene size and plotted the results in Figure 8.

We observed an optimal cell dimension that depends on the size of the scene. Too small cell dimensions can cause a lower SSIM, since the views only have finite resolution, and varying views across very small cells encourages mosaicing artifacts. If the cell dimension gets too large (larger than 30-100cm for the tested scenes), many views will not cover the entire area inside the cell and, in unfortunate situations, parts of the geometry are not properly covered. A heuristic based on (expected) scene diameter is therefore a good solution.

4.4 Distance of views and cells

The parameters d_{max} and σ define the ideal distance of a view from a cell, such that the view covers the cell at the desired resolution. As can be seen in Figure 11, the choice of these parameters is not very sensitive in the apartment scene, which has a small overall diameter, so that even the furthest parts of the scene are covered with good resolution in every view. In contrast, the much larger viking village scene benefits from an appropriate parameter choice. Here, a choice of $\sigma^2 = 20m^2$ yields the best distribution of views with respect to the obtained image quality.

4.5 View store size

We wanted to find out how the number of stored images (a few hundred 2MPix images fit in GPU memory) influences quality. We processed the images using various cache sizes and plotted results for both data sets in Figure 9. Obviously, a larger cache contains more information and can yield a better image quality if used properly. Nonetheless, we observed that cache sizes above 100-200 images (again, dependent on scene characteristics) yield diminishing returns, implying that a finite number of views is sufficient, and a view store of reasonable size can be filled and maintained incrementally.

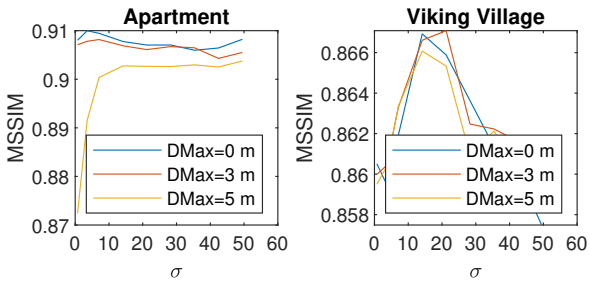

Fig. 11. Effect of view-cell distance parameters σ and d_{max} on MSSIM.

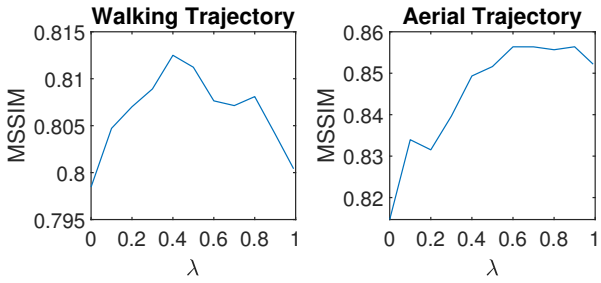

Fig. 12. Effect of changing the weight of directional and positional coverage on MSSIM. The walking trajectory has more occlusions and requires giving a plausible to positional coverage (e.g., $\lambda = 0.4$).

4.6 Influence of per-cell view references on quality

We studied how image quality changes with varying number of references per cell as shown in Figure 10. Note that all lumigraph renderings were produced by blending only three views per pixel, but chosen from the all references stored in a cell. More references per cell increase the chance that views with good coverage of a sample within that cell are found. However, an increasing number of references exhibits diminishing returns after 15-20 references, implying that cells can be properly covered with a small number of views, if chosen carefully. This insight is of practical importance, since the rendering speed depends on the reference number that must be searched in the fragment shader (Table 2). A reference number of 16 appears to be the best trade-off.

4.7 Influence of refinement on quality

We investigated the influence of view refinement on quality by computing the difference between lumigraph rendering results with refinement, without refinement, and the reference images. We present side by side images of before and after image pose refinement in Figure 6. As expected, image quality is significantly increased by the refinement. This result is consistent with observations made by other authors [47, 51]. However, our work demonstrates that a refinement carried out in real time (i.e., without extensive global search and optimization) is feasible.

4.8 Speed

We measured the performance of our algorithm by means of view selection speed and rendering speed. Specifically, we investigated performance impact of number of references as shown in Table 2. The performance reported in the table is obtained while rendering at 1080p resolution along a trajectory through the scene. Performance depends on both the geometric complexity of the scene (which is out of scope of our work) and the number of references. At the recommended number of 16 references, we observe between 3-54ms per input image for view planning (considering a new frame for the view store) and 18-37ms for rendering. There is ample room for optimizations, but we already achieve interactive performance that would be suitable, for example, for viewing a remote scene in a telepresence application.

Data set				Model	(ms)				Render	(ms)	
Name	Tris	Res.	Refine	8 ref	16 ref	32 ref	60 ref	8 ref	16 ref	32 ref	60 ref
Viking village	6.1M	2.1MP	-	4.2	4.5	6.3	22.7	16.6	29.6	61	101.8
Apartment	1.6M	2.1MP	-	3.1	3.1	8.7	16.9	16.5	19	43	89.3
City wall	10.3M	3.0MP	350	37.5	53.5	70.4	104.2	23.6	37.3	69	109
Lab	1.6M	0.3MP	60	2.1	15.2	44.2	63.3	16.6	17.7	29.8	45.9

Table 2. Performance overview. The left block of columns lists model name, primitive count (in million triangles) and input image resolution (in mega-pixels). The middle block gives timings of the modeling stage, and the right block, of the rendering stage, both in milliseconds. Column "Refine" is the average time for refining an input view. Columns labeled "ref" list timings for 8, 16, 32 or 60 references per cell, for either modeling or rendering.

4.9 Quality comparison to reference methods

We visually compare our method to two reference methods on the same input data: The first method is KinectFusion, which accumulates depth images in a volume tabulating a truncated signed distance function and extracts a mesh with per-vertex averaged colors. It represents the class of real-time methods. The second method used for comparison is 3DF Zephyr [2], a leading commercial photogrammetry software. It represents the class of offline method, which can perform arbitrary global optimizations. Zephyr delivers noticeable higher precision geometry than KinectFusion, but required 40 min of compute time on the Lab dataset. Texture quality is optimized as well; however, Zephyr combines texture extracted from multiple into one final surface texture, which is not view-dependent like ours. For a fair comparison, our method used the geometry delivered by KinectFusion under real-time constraints as input, and not the more precise geometry of Zephyr.

Side-by-side image comparisons can be seen in Figure 15. Informally, our method performs on par with 3DF Zephyr, while our method clearly outperforms the result delivered by KinectFusion in the same instant timeframe. KinectFusion's per-vertex coloring exhibits blurry rendering due to lack of high frequencies in its textures. While Zephyr utilizes textures much better, it has no intensity correction or view-dependent representation and shows significant mosaicing artifacts. Visual inspection of magnified details reveals that Zephyr's advantage of more precise geometry is lost if mosaicing cannot be suppressed.

To demonstrate the influence that the view planning method alone has on visual quality, we tested the image quality that our incremental view planning delivers compared to a globally optimal version of the same planning. We chose the 250 best out of 5000 keyframes by global optimization over all 5000 keyframes simultaneously, and compare the resulting image quality to the online method (which considers the views in strict sequence, one at a time, and not simultaneously). The globally optimal version of our algorithm results in an MSSIM increase of 1.3% compared to online version. Visual differences are only visible when looking carefully, as can be seen in Figure 13.

5 Conclusions and Future Work

We have described a new method for online view planning in reconstruction applications that builds an unstructured lumigraph. The views for the lumigraph are selected such that they cover the entire scene well without having to revisit older views, which are not longer available in the view store. Such an approach has previously only been explored in next-best view planning in robotics, but with the difference that our algorithm has no control over where the human operator will move the camera to. As far as we know, we are the first to explore this situation, although it is highly relevant for mobile MR applications.

In future work, we will integrate our view planning system with real-time scanning, rather than working from recorded and played-back sequences. While this is conceptually straight forward, it implies two important additional problems: First, testing lumigraph quality is much more difficult if every test run gives a different trajectory. Second, incremental geometric reconstruction via SLAM does make all geometry available immediately. Therefore, a short delay (e.g., a few seconds) of storing incoming images will be required to be able to match geometry to view candidates. It will also require that new portions of the geometry are cross-referenced with existing views periodically to make the most out of the view store's content. In addition to including real-time reconstruction, we will also investigate further algorithmic improvements, such as replacing the simple greedy optimization strategy with a more sophisticated optimization strategy that identifies areas of the scene with good coverage and excludes them from disruptive updates.

[2] http://www.3dflow.net

Fig. 13. Rendering results with our incremental view planning algorithm (top) and the globally optimal version of our algorithm (bottom) are shown. Subtle differences can be realized only in a careful examination.

References

[1] D. Schmalstieg and T. Höllerer, *Augmented Reality - Principles and Practice*. Addison-Wesley Professional, June 2016.

[2] O. Erat, W. A. Isop, D. Kalkofen, and D. Schmalstieg, "Drone-augmented human vision: Exocentric control for drones exploring hidden areas," *IEEE Trans. Vis. Comp. Graph.*, vol. 24, no. 4, pp. 1437–1446, 2018.

[3] P. Hedman, T. Ritschel, G. Drettakis, and G. Brostow, "Scalable inside-out image-based rendering," *ACM Trans. Graph.*, vol. 35, no. 6, pp. 1–11, 2016.

[4] S. Bi, N. K. Kalantari, and R. Ramamoorthi, "Patch-based optimization for image-based texture mapping," *ACM Transactions on Graphics*, vol. 36, pp. 1–11, jul 2017.

[5] M. Waechter, N. Moehrle, and M. Goesele, "Let there be color! Large-scale texturing of 3D reconstructions," in *ECCV*, pp. 836–850, 2014.

[6] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen, "The lumigraph," in *Proc. SIGGRAPH*, pp. 43–54, 1996.

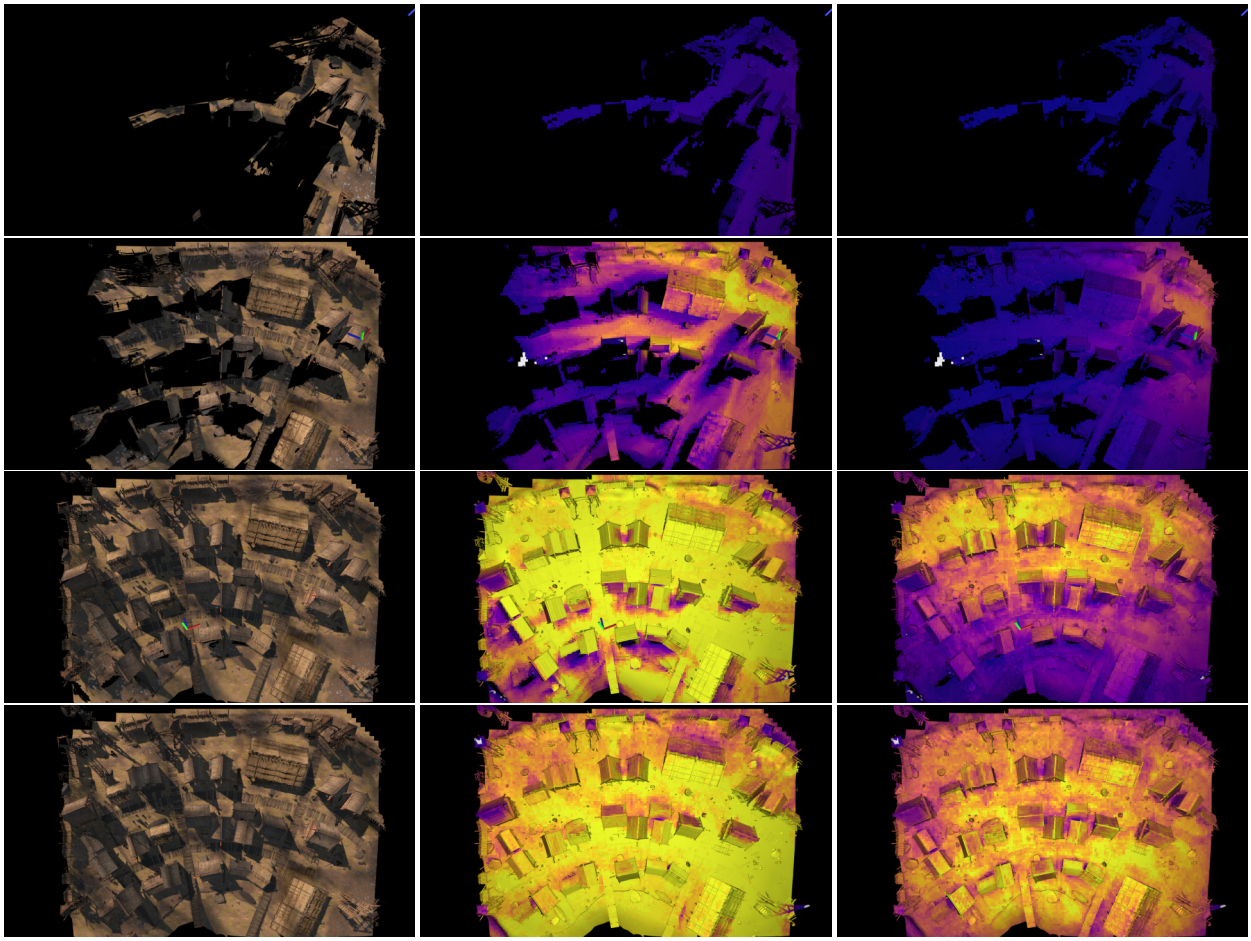

Fig. 14. View planning visualization over time on the *Viking village* dataset. From left to right: (a) Lumigraph rendering (b) Cell reference count $[0, max]$, (c) Normalized cell coverage.

Fig. 15. Comparation of our method to per-vertex colored KinectFusion model and to an offline-reconstruction rendering using a leading commercial photogrammetry (3DF Zephyr). From left to right: (a) KinectFusion's rendering (b) our rendering (c) 3DF Zephyr's rendering results.

[7] S. Fleishman, D. Cohen-Or, and D. Lischinski, "Automatic camera placement for image-based modeling," in *Proc. Pacific Conference on Computer Graphics and Applications*, pp. 12–20, 1999.

[8] P.-P. Vázquez, M. Feixas, M. Sbert, and W. Heidrich, "Automatic view selection using viewpoint entropy and its application to image-based modelling," *Computer Graphics Forum*, vol. 22, pp. 689–700, Nov. 2004.

[9] J. O'Rourke, *Art Gallery Theorems and Algorithms*. New York, NY, USA: Oxford University Press, Inc., 1987.

[10] U. Feige, "A threshold of ln n for approximating set cover," *J. ACM*, vol. 45, pp. 634–652, July 1998.

[11] C. Buehler, M. Bosse, L. McMillan, S. Gortler, and M. Cohen, "Unstructured lumigraph rendering," in *Proc. SIGGRAPH*, pp. 425–432.

[12] M. Levoy and P. Hanrahan, "Light field rendering," in *Proc. SIGGRAPH*, pp. 31–42, 1996.

[13] M. Bolas, A. Kuruvilla, S. Chintalapudi, F. Rabelo, V. Lympouridis, C. Barron, E. Suma, C. Matamoros, C. Brous, A. Jasina, *et al.*, "Creating near-field vr using stop motion characters and a touch of light-field rendering," in *ACM SIGGRAPH 2015 Posters*, p. 19, ACM, 2015.

[14] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," pp. 127–136, 2011.

[15] S. Dong and T. Hollerer, "Real-Time Re-Textured Geometry Modeling Using Microsoft HoloLens," in *IEEE Virtual Reality*, pp. 231–237, 2018.

[16] Q. Zhou and V. Koltun, "Color map optimization for 3d reconstruction with consumer depth cameras," *ACM Trans. Graph.*, vol. 33, no. 4, 2014.

[17] R. Gal, Y. Wexler, E. Ofek, H. Hoppe, and D. Cohen-Or, "Seamless Montage for Texturing Models," *Eurographics 2010*, vol. 29, 5 2010.

[18] J.-X. Chai, S.-C. Chan, H.-Y. Shum, and X. Tong, "Plenoptic sampling," in *Proc. SIGGRAPH*, pp. 307–318, ACM Press, 2000.

[19] P. Debevec, C. Taylor, and J. Malik, "Modeling and rendering architecture from photographs: a hybrid geometry- and image-based approach," in *Proc. SIGGRAPH*, pp. 11–20, 1996.

[20] M. Eisemann, B. De Decker, M. Magnor, P. Bekaert, E. de Aguiar, N. Ahmed, C. Theobalt, and A. Sellent, "Floating Textures," *Computer Graphics Forum*, vol. 27, pp. 409–418, apr 2008.

[21] W.-C. Chen, J.-Y. Bouguet, M. H. Chu, and R. Grzeszczuk, "Light field mapping: Efficient representation and hardware rendering of surface light fields," in *Proc. SIGGRAPH*, pp. 447–456, 2002.

[22] G. Chaurasia, S. Duchêne, O. Sorkine-Hornung, and G. Drettakis, "Depth synthesis and local warps for plausible image-based navigation," *ACM Transactions on Graphics*, vol. 32, 2013.

[23] R. O. Cayon, A. Djelouah, and G. Drettakis, "A bayesian approach for selective image-based rendering using superpixels," in *3DV*, pp. 469–477, 2015.

[24] V. Hlaváč, A. Leonardis, and T. Werner, "Automatic selection of reference views for image-based scene representations," in *ECCV*, pp. 526–535, Springer, Berlin, Heidelberg, 1996.

[25] C. Zhu, L. Yu, and Z. Xiong, "A Noncoverage Field Model for Improving the Rendering Quality of Virtual Views," *IEEE Trans. Multimed*, vol. 20, no. 3, pp. 738–753, 2018.

[26] A. Davis, M. Levoy, and F. Durand, "Unstructured light fields," *Comput. Graph. Forum*, vol. 31, pp. 305–314, May 2012.

[27] M. Dou and H. Fuchs, "Temporally enhanced 3D capture of room-sized dynamic scenes with commodity depth cameras," in *2014 IEEE Virtual Reality (VR)*, pp. 39–44, IEEE, mar 2014.

[28] S. Orts-Escolano, M. Dou, V. Tankovich, C. Loop, Q. Cai, P. Chou, S. Mennicken, J. Valentin, V. Pradeep, S. Wang, S. Kang, C. Rhemann, P. Kohli, Y. Lutchyn, C. Keskin, S. Izadi, S. Fanello, W. Chang, A. Kowdle, Y. Degtyarev, D. Kim, P. Davidson, and S. Khamis, "Holoportation: Virtual 3D Teleportation in Real-time," in *ACM UIST*, pp. 741–754, 2016.

[29] S. Gauglitz, B. Nuernberger, M. Turk, and T. Höllerer, "World-stabilized annotations and virtual scene navigation for remote collaboration," in *ACM UIST*, pp. 449–459, 2014.

[30] W. R. Scott, G. Roth, and J.-F. Rivest, "View planning for automated three-dimensional object reconstruction and inspection," *ACM Comput. Surv.*, vol. 35, pp. 64–96, Mar. 2003.

[31] M. Goesele, N. Snavely, B. Curless, H. Hoppe, and S. M. Seitz, "Multi-view stereo for community photo collections," in *ICCV*, pp. 1–8, 2007.

[32] M. Mauro, H. Riemenschneider, A. Signoroni, R. Leonardi, and L. J. V. Gool, "An integer linear programming model for view selection on overlapping camera clusters," in *3DV*, pp. 464–471, 2014.

[33] A. Hornung, B. Zeng, and L. Kobbelt, "Image selection for improved multi-view stereo," in *CVPR*, 2008.

[34] C. Mostegel, F. Fraundorfer, and H. Bischof, "Prioritized multi-view stereo depth map generation using confidence prediction," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 143, pp. 167 – 180, 2018.

[35] S. Chen, *Active sensor planning for multiview vision tasks*. Springer, 2008.

[36] E. Dunn, J. P. van den Berg, and J. Frahm, "Developing visual sensing strategies through next best view planning," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4001–4008, 2009.

[37] C. Freundlich, P. Mordohai, and M. M. Zavlanos, "A hybrid control approach to the next-best-view problem using stereo vision," in *ICRA*, pp. 4493–4498, 2013.

[38] O. M. Maldonado, S. Hadfield, N. Pugeault, and R. Bowden, "Taking the scenic route to 3d: Optimising reconstruction from moving cameras," in *ICCV*, pp. 4687–4695, 2017.

[39] R. Huang, D. Zou, R. Vaughan, and P. Tan, "Active image-based modeling with a toy drone," in *ICRA*.

[40] B. Hepp, M. Nießner, and O. Hilliges, "Plan3d: Viewpoint and trajectory optimization for aerial multi-view stereo reconstruction," *ACM Trans. Graph.*, vol. 38, no. 1, 2019.

[41] C. Hoppe, M. Klopschitz, M. Rumpler, A. Wendel, S. Kluckner, H. Bischof, and G. Reitmayr, "Online feedback for structure-from-motion image acquisition," in *BMVC*, 2012.

[42] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," in *ISMAR*, 2007.

[43] O. Kahler, V. A. Prisacariu, C. Y. Ren, X. Sun, P. H. S. Torr, and D. W. Murray, "Very High Frame Rate Volumetric Integration of Depth Images on Mobile Device," *IEEE TVCG*, vol. 22, no. 11, 2015.

[44] R. Kusner, W. Kusner, J. C. Lagarias, and S. Shlosman, "Configuration spaces of equal spheres touching a given sphere: The twelve spheres problem," *arXiv preprint arXiv:1611.10297*, 2016.

[45] L. Gruber, T. Richter-Trummer, and D. Schmalstieg, "Real-time Photometric Registration from Arbitrary Geometry," in *ISMAR*, 2012.

[46] G. Farnebäck, "Two-frame motion estimation based on polynomial expansion," in *Proceedings of the 13th Scandinavian Conference on Image Analysis*, SCIA'03, (Berlin, Heidelberg), pp. 363–370, Springer-Verlag, 2003.

[47] T. Richter-Trummer, J. Park, D. Kalkofen, and D. Schmalstieg, "Instant mixed reality lighting from casual scanning," in *ISMAR*, 2016.

[48] S. Fuhrmann, F. Langguth, and M. Goesele, "Mve-a multi-view reconstruction environment.," in *GCH*, pp. 11–18, 2014.

[49] R. J. Weilharter, F. Schenk, and F. Fraundorfer, "Globally consistent dense real-time 3d reconstruction from rgbd data," in *OAGM Workshop*, 2018.

[50] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, pp. 600–612, apr 2004.

[51] M. Meilland, C. Barat, and A. Comport, "3D High Dynamic Range dense visual SLAM and its application to real-time object re-lighting," in *ISMAR*, pp. 143–152, 2013.