View-Region Optimized Image-Based Scene Simplification

PUNEET LALL, Google Inc.
SILVIU BORAC, Google Inc.
DAVE RICHARDSON, Google Inc.
MATT PHARR, Google Inc.
MANFRED ERNST, Google Inc.

Fig. 1. Our algorithm is capable of processing complex scenes into a representation that renders in real-time on mobile hardware. The images show the final rendered output. Left: The LANDSCAPE scene from PBRT, an environment with over 3.1 billion instanced trianges that takes hours to render in a CPU ray tracer. Center: The SMITHY from the blacksmith environment, a scene designed for high-end PC GPUs. Right: The COLOSSEUM from Google Earth, captured at desktop-level quality.

We present a new algorithm for image-based simplification of complex scenes for virtual reality (VR). The algorithm transforms geometrically-detailed environments into a layered quad tile representation that is optimized for a specified viewing region and is renderable on low-power mobile-class VR devices. A novel constrained optimization formulation ensures that the scene can be rendered within a predetermined compute budget, with limits on both primitive count and fill rate. Furthermore, we introduce a new method for texturing from point samples of the original scene geometry that generates high-quality silhouettes without the drawbacks of traditional point splatting.

The resulting representation achieves a visual fidelity that was previously impossible on mobile graphics hardware; our algorithm can typically generate a high-quality representation of visually-rich scenes with billions of triangles using just 72k triangles and a single high-resolution texture map (with generally only about 50% more texels than a stereo panorama). The effectiveness of the approach is demonstrated with a set of challenging test cases.

CCS Concepts: • **Computing methodologies** → **Image-based rendering**; **Virtual reality**; **Shape modeling**;

Additional Key Words and Phrases: VR, scene simplification

Authors' addresses: Puneet Lall, Google Inc. Silviu Borac, Google Inc. Dave Richardson, Google Inc. Matt Pharr, Google Inc. Manfred Ernst, Google Inc.

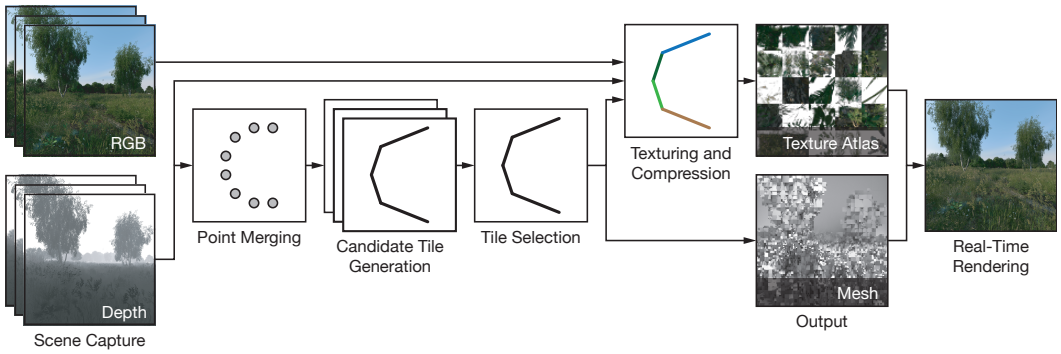

Fig. 2. We capture complex scenes by rendering color and depth images from within a specified viewing region and then approximate the scene with a set of textured quad tiles. A tile optimization algorithm constrains the output tiles to fit within a user-defined triangle and fill rate budget. Silhouettes are generated using a space carving approach capable of resolving even sub-pixel geometry, and content-adaptive atlas compression guarantees bounded texture size. This Landscape scene, with over 3.1 billion instanced triangles, can be accurately represented with only 72 thousand triangles, a 4096^2 texture atlas, and a constrained peak fill rate, together ensuring real-time rendering on low-powered mobile GPUs.

ACM Reference Format:
Puneet Lall, Silviu Borac, Dave Richardson, Matt Pharr, and Manfred Ernst. 2018. View-Region Optimized Image-Based Scene Simplification. *Proc. ACM Comput. Graph. Interact. Tech.* 1, 2, Article 26 (August 2018), 22 pages. https://doi.org/10.1145/3233311

1 INTRODUCTION

Consumer VR hardware is finally a reality, but content creation remains a challenge. When they put on a headset, users expect to be brought into high-quality experiences not unlike the latest console games and VFX-packed movies. However, bringing cinematic realism to virtual reality is no easy task.

For a VR system to give the illusion that the user is present in another world, rendering performance is key. VR systems must render at high frame rates and do so reliably: dropped frames quickly break immersion. Furthermore, they must render at high resolutions, due to stereo and increased field of view, and with high-quality anti-aliasing due to magnified pixels. As a result, delivering good VR experiences on a high-end PC remains challenging, requiring strict attention to performance and efficiency [Vlachos 2016]. Mobile-class VR devices further constrain the resources available for rendering, relying on GPUs which must operate within a thermal footprint orders of magnitude smaller than their desktop counterparts, limited to generally 10× to 20× less performance than a desktop system [Pharr 2017] [Boos et al. 2016, Figure 11]. These compounding performance constraints pose challenges to content creators.

VR experiences often place users in a largely-static environment with most interaction confined to a limited region around the user. In these scenarios, users typically observe the environment from a relatively-small viewing region. It is this assumption of a limited viewing region that we exploit in our system to simplify the process of content creation.

In this paper, we present a practical method for image-based simplification and rendering of geometrically-detailed, static environments from within a constrained viewing region. Our approach converts complex static environments with billions of triangles into a form that is easily rendered on today's mobile GPUs. Our runtime rendering algorithm can generate a z-buffer that

closely approximates the original scene geometry; dynamic content can be rendered together with environments generated by our system.

The main features of our approach are that it provides:

- A flexible scene-capture and ingest system, only requiring an external renderer that can either trace rays or generate a set of RGBD images of the scene.
- Low runtime rendering cost that can be tightly controlled through user-provided constraints on triangle count, overdraw, and texture memory usage.
- Faithful reconstruction of thin structures and detailed silhouettes with high quality anti-aliasing.
- A compact representation suitable for packaging hundreds of environments within a single application.

Our method is limited in that:

- The output is only valid when rendered from within the selected viewing region.
- Dynamic content cannot be processed and must be added with traditional graphics techniques.
- View-dependent shading effects must be pre-baked or rely on g-buffer based methods.

Despite these constraints, our system is practical and has been successfully deployed in production content authoring pipelines to process hundreds of scenes for mobile VR applications, including Expeditions [Google Inc. 2018] and Blade Runner: Revelations [Alcon Interactive 2018], which combines pre-baked scenes with dynamic and interactive content.

Our pipeline, shown in Figure 2, begins by densely sampling a scene to generate a ray-space surface light field—a large collection of rays through free-space that store the distance where each ray first hits a surface and the outgoing radiance at the surface.

From these samples, our method approximates the sampled surface geometry with a collection of quadrilateral tiles. This representation is a special case of a billboard cloud [Décoret et al. 2003], but our algorithm for solving for these tiles is capable of limiting not only primitive count, but also texture area and peak fill rate required to render the quads; the latter is especially critical for practical rendering of full environments in mobile VR where fillrate can dominate performance.

These tiles are then textured with RGBA atlases that encode filtered surface radiance and silhouettes. To generate these textures from the input surface light field samples, we eschew point-splatting methods in favor of a new approach that interprets the capture as a collection of both point samples and ray samples. Our memory-efficient space carving algorithm generates textures capable of accurately encoding silhouettes of thin structures and subpixel geometry. Texture atlas memory use can be limited with a hard constraint; we apply a content-aware optimization approach that allocates texels where they are most useful to reduce reconstruction error. Typical output complexity with high-quality settings is just 72k triangles and 17 MB of ASTC-compressed texture when matching the display densities of current headsets; this is only approximately 50% more than a display-resolution-matching stereo panorama even though we produce accurate translational parallax in the viewing region; see Section 7 for more detailed results.

2 PREVIOUS WORK

Automatic processing of 3D content for efficient rendering has been a longstanding goal in computer graphics. Here we review work that is applicable to content creation for VR applications and exploits a limited viewing region as we do. We then discuss work that addresses the more general case of unconstrained camera pose, but that uses a geometry-simplification approach, as in our method.

2.1 Omnidirectional stereo panoramas

Some mobile VR systems provide only three degree-of-freedom (3DoF) tracking, which provides viewer orientation but not position. On such systems, omnidirectional stereo panoramas (ODS) [Ishiguro et al. 1990; Peleg et al. 2001; Richardt et al. 2013] have proven to be an effective format for transmitting and rendering compelling VR content [Anderson et al. 2016]. As with our approach, ODS has low, predictable rendering cost and provides a compact representation largely independent of scene complexity. ODS panoramas can also be generated from diverse sources including renderers and camera systems. However, unlike our method, stereo panoramas do not represent translational parallax, and so they cannot take advantage of the 6DoF position and orientation tracking that is now becoming available in integrated VR headsets.

2.2 Light Fields

Image based rendering methods such as light fields can provide a full 6DoF experience. Light fields are capable of representing arbitrary scenes, both synthetic and real-world, given sufficiently dense sampling [Gortler et al. 1996; Levoy and Hanrahan 1996]. Although conceptually elegant, they have high memory requirements and direct light field rendering methods have yet to be demonstrated to be practical for mobile VR. If accurate scene geometry is available, surface light fields are a more compact representation [Wood et al. 2000], however that approach doesn't address the dramatic reduction in complexity that is necessary to bring complex scenes to VR on mobile-class GPUs.

Recently, others have investigated image based rendering methods to support content on 6DoF VR headsets [Boos et al. 2016; Huang et al. 2017; Koniaris et al. 2016, 2017; Luo et al. 2018]. As in our work, these methods also assume a limited viewing region. Some methods [Huang et al. 2017; Luo et al. 2018] build on 360 video and ODS representations to provide a 6DoF effect, but they cannot faithfully represent disoccluded regions as ours does. Others are similar to our method in that they sample synthetic scenes from collections of color and depth images, thus capturing disoccluded content. However, these methods either target high-end desktop GPUs [Koniaris et al. 2017], leaving mobile rendering an open problem, or they have very high storage requirements (as much as 50 GB for a single scene) [Boos et al. 2016]. In contrast to these methods, which use a compression-centric approach to store and interpolate the RGBD images for each view, our pipeline converts all views into a single global geometric representation, using a geometry-simplification approach. As a result, our representation is both compact and efficiently renderable on mobile hardware.

2.3 Impostors

In the general case of geometry simplification for interactive rendering of complex environments, various types of impostor representations have been developed.

Surfels [Pfister et al. 2000] and layered depth images (LDIs) [Shade et al. 1998] are point-based methods. They can be generated using a modified raytracer, similar to our approach. However, these representations cannot adequately reconstruct fine detail in thin geometric structures. Although high-quality rendering has been demonstrated on GPUs [Ren et al. 2002; Zwicker et al. 2001], point rendering remain impractical on power-limited mobile VR hardware due to both a vertex workload beyond what the hardware is capable of and insufficient rasterization performance for millions of pixel-sized points per frame.

Layered representations have been explored in the context of view interpolation [Penner and Zhang 2017; Schaufler 1998]. These approaches represent a scene as a collection of textured fronto-parallel planes, composited with alpha blending. However, many planes are necessary to achieve

compelling results for environments with significant parallax, requiring unconstrained texture memory and fill rate to render.

Textured depth meshes have been proposed as a view-dependent impostor [Jeschke and Wimmer 2002; Wilson and Manocha 2003] that builds on classic mesh simplification approaches [Garland and Heckbert 1997] to generate efficiently-renderable geometry from a volumetric scene sampling. However, they are only capable of representing geometry sampled at the resolution of the voxel grid. In contrast, our approach can faithfully represent subpixel geometry encoded in alpha textures and can do so with higher quality anti-aliasing than when rendering with opaque meshes.

Volumetric representations have been proposed for rendering complex environments interactively [Crassin et al. 2009; Decaudin and Neyret 2009; Gobbetti and Marton 2005]. As with point-based methods, efficient voxel rendering is challenging, requiring either geometry shaders and high fill rate [Decaudin and Neyret 2009], or high computation and bandwidth usage to traverse octrees [Crassin et al. 2009], making them impractical for mobile hardware.

Billboard clouds [Décoret et al. 2003] are the most similar to our layered quad tile representation. They have found applicability in rendering trees and forests [Behrendt et al. 2005; Fuhrmann et al. 2005; Lacewell et al. 2006] and are appealing for their ability to represent detailed silhouettes with low vertex workload. Although various improvements have been proposed, including methods for obtaining better-fit billboards [Andujar et al. 2004], all current methods suffer from unbounded texture memory and/or fill rate requirements. As a result, they are typically reserved for rendering distant, isolated objects to limit the impact of their relatively-high per-pixel cost. In this work, we directly address this critical performance limitation, using a novel optimization method that can impose hard constraints on the total fill rate of our layered quad tile representation. In doing so, we unlock their use in rendering complete environments in performance-constrained mobile VR.

3 SCENE CAPTURE

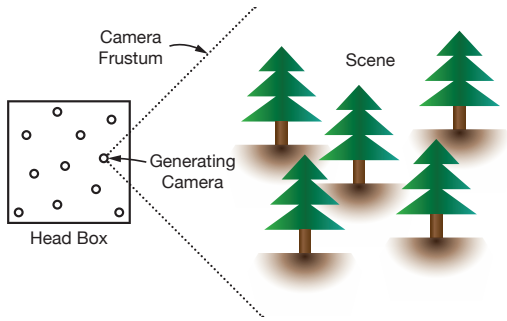

Fig. 3. The scene is captured by rendering RGB and depth cube maps from a set of view points inside the viewing region.

In order to easily support a wide variety of sources of content, we have designed our approach to take as input collections of rays from the user-defined viewing region that each record their intersection point and the outgoing radiance from the surface they hit. This is a fully-general sampled representation of the scene, just like traditional light fields. Both production rendering systems and game engines can easily generate this information, without needing to export the full scene geometry or shaders. Our system does expect that there be no scene geometry inside the selected viewing region. (Such geometry is best handled using traditional rendering techniques instead.)

Rasterizers can efficiently generate this representation by rendering views of the scene from a series of points within the viewing region. We render cube-map views of the scene from a well-distributed set of viewpoints within the viewing region and record color and depth at each pixel, using a 3D Hammersley point set [Niederreiter 1992] to position the cameras, ensuring that no two cameras are too close (see Figure 3). Given the camera positions and projection matrices, the resulting images are easily converted to a collection of rays.

While ray tracers can compute color and depth for arbitrary distributions of rays traced into the scene, allowing for a more uniform sampling in ray-space, we have generally just rendered cube-map views with them as well. The workflow advantages of not needing to modify the renderer's source code or to write custom plugins to trace arbitrary rays have been significant, allowing us to process content from a new ray tracer after just an hour or so of integration work.

The number of viewpoints required for a good sampling of the scene depends on the scene's complexity, especially the amount of disocclusion between viewpoints. High-frequency geometry also requires more viewpoints to properly capture areas that are only visible from a small portion of the viewing region, but we have found that 32 to 64 viewpoints work well for most scenes. The resolution of the generated cube maps is chosen to roughly match the resolution (in pixels per degree) of the target head-mounted display. A small number of views may be rendered at a higher resolution for improved anti-aliasing in the texture generation stage.

Another way to sample the scene would be to generate a layered depth image, tracing rays from the center of the viewing region though all surfaces they intersect. However, with this approach, occluded regions of the scene would be included in the output, leading to substantially worse results due to geometry and texture being allocated to parts of the scene that will never be seen.

4 TILING

Hundreds of millions of samples are generated during scene capture—an overwhelming number for even a high-end desktop GPU. Therefore, we seek to generate efficiently-renderable proxy geometry that accurately approximates the scene geometry subject to constraints that model the GPU workload for the proxy geometry. In this section, we will discuss how we compute coarse proxy geometry that represents the scene samples. In Section 5, we will then explain how this geometry is clipped to accurately represent the actual geometry with alpha-mapped textures.

More formally, we start with a collection of points on surfaces of the scene, P, and would like to generate a set of approximating quads, T, optimizing

$$\begin{aligned} &\underset{T}{\text{minimize}} && \sum_{p \in P} \min_{t \in T} E(p,t) \\ &\text{subject to} && \sum_{t \in T} \text{Weight}(t) \leq \text{Capacity}, \end{aligned} \tag{1}$$

where $E(p,t)$ is an energy term that represents the error introduced by approximating p with t, $\text{Weight}(t)$ models the total cost of rendering quad t, and Capacity is a user-specified constraint on the cost of rendering the scene.

We define the origin to be the center of the viewing region. The energy associated with a point p and quad t is then defined as,

$$E(p,t) = \begin{cases} (1 - t_{\text{hit}}(p))^2, & \text{if } t_{\text{hit}}(p)p \text{ lies within t} \\ \infty & \text{otherwise,} \end{cases} \tag{2}$$

where $t_{\text{hit}}(p)$ is the scale factor that when applied to p, projects it onto the plane of t. (See Figure 4.) Note that the error function E models geometric distortion radially, more heavily penalizing error

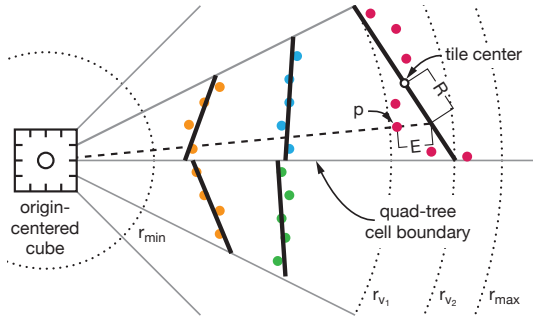

Fig. 4. The energy term $E(p, t)$ (Equation 2) models the error in approximating a scene sample point p with the quad t.

at points closer to the viewing region's center and thus incorporating the fact that we will only render the scene from within a limited viewing region.

We optimize the quads based on a model of three key factors that affect GPU performance: the number of primitives to draw, their total area as seen from the center of the viewing region, and the peak overdraw—the largest number of fragments the rasterizer generates for any pixel when the scene is rendered. We have found that these three give a sufficiently expressive model in practice:

- $L_{\text{tris}} \in \mathbb{Z}$ limits the total number of triangles. Its value controls the load on GPU vertex processing and the rasterizer.
- $L_{\text{area}} \in \mathbb{R}$ limits the total area of the quads projected onto the unit sphere at the center of the viewing region—an approximation of the total rasterizer load they induce. Because our quads are textured with a projective parameterization, as described in Section 5, the value of the L_{area} constraint also implicitly controls maximum texture consumption.
- $L_{\text{peakOD}} \in \mathbb{R}$ limits the peak overdraw required to render the scene from any pose within the viewing region. This additional overdraw limit is necessary to prevent degradation of rendering performance from some views, especially for scenes with uneven distribution of high-frequency geometry, which typically require multiple layers of overlapping quads.

In order to make the optimization tractable, we make an approximation when computing the peak overdraw estimate: we estimate peak overdraw for a set of F discrete camera orientations. To do so, we first compute an upper bound of each quad's projected area as seen from any point in the viewing region and then charge that quad's consumed fill rate to the camera orientations that can see it from the center of the viewing region. While this approach doesn't give a strict bound on fill rate over all possible camera positions and orientations, we have found good results with a 90-degree field of view and use $F = 126$ sampled views. In Section 7, we show that even with these approximations, the peak overdraw constraint is nevertheless either met or only slightly exceeded in practice.

Any of these limits alone is insufficient to yield a practical system; one might, for example, approximate a scene with many large fronto-parallel quads, as in recent view interpolation approaches [Penner and Zhang 2017]. Such a system would have an acceptably low primitive count, but would require a large amount of texture memory and would be limited by the fill rate required to composite all of the layers.

We can now define the capacity vector,

$$\text{Capacity} = \begin{bmatrix} L_{\text{tris}} \\ L_{\text{area}} \\ L_{\text{peakOD}} \\ \vdots \\ L_{\text{peakOD}} \end{bmatrix},$$

as well as the multidimensional weight for each quad t,

$$\text{Weight}(t) = \begin{bmatrix} \text{Tris}(t) \\ \text{ProjArea}(t) \\ \text{PeakOD}_1(t) \\ \vdots \\ \text{PeakOD}_F(t) \end{bmatrix},$$

where $\text{Tris}(t) = 2$, the number of triangles used to represent each quad, $\text{ProjArea}(t)$ is the area of the projection of a quad onto the unit sphere, and $\text{PeakOD}_i(t)$ is the estimated peak overdraw required to render quad t from the ith camera position.

Solving the general optimization problem of Equation 1 while considering a solution space of arbitrary 3D quads seems intractable, so instead we restrict the search to only consider geometry consisting of *tiles* derived from a quadtree structure. We consider quadtrees on the six faces of a cube centered in the middle of the viewing volume and then define a tile to be the intersection of a plane with the square frustum extruded from a quadtree cell. As discussed in Section 4.3, restricting the solution space to tiles formed from this tree-like subdivision enables the use of an efficient combinatorial optimization algorithm.

Our algorithm for solving for these tiles consists of three steps, described in subsequent sections:

Point merging. First, all captured scene samples are merged into a single point cloud to reduce memory and computation requirements.

Candidate tile generation. Next, a diverse, overcomplete set of tiles are fit to this point cloud using an iterative clustering algorithm.

Tile selection. Finally, a subset of these candidate tiles is selected so that the entire scene is covered while geometric distortion is minimized and resource constraints are satisfied.

4.1 Point Merging

When capturing environments that include high-frequency geometry, e.g. foliage, it may be necessary to sample from many camera views to ensure coverage of partially-occluded regions of the scene. However, this often results in needlessly-dense sampling of simple, unoccluded surfaces. Therefore, all scene samples are first adaptively coalesced into a condensed point cloud, with redundant points discarded for efficiency in subsequent processing stages.

We stream samples from disk and bin them into six perspective-space voxel grids arranged over a cube-map at the center of the viewing region. The points are merged into a single point cloud by selecting at most one representative sample for each voxel via reservoir sampling, with all other points quantized to it. The resulting merged point cloud records the number of samples that were merged with each point for use in subsequent processing stages. Environments can thus fit in a few hundred megabytes of RAM during processing regardless of the density of the original capture, which may otherwise consist of gigabytes of point data.

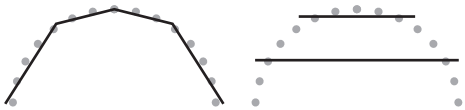

Fig. 5. Parts of a scene can be represented by different configurations of tiles with different performance characteristics. A sphere may be approximated by 4 quads (left), or by 2 layered quads (right) with approximately double the fill rate and texture requirement. Our algorithm generates many such configurations and then selects from among them, optimizing resource usage over the entire scene.

4.2 Candidate Tile Generation

From the merged point cloud, we next seek to generate a diverse, overcomplete set of candidate tiles that approximates the geometry.

As shown in Figure 5, the same geometry can be well-approximated with varying numbers of layers or with varying subdivision, resulting in different trade-offs between triangle count, texture memory usage, and fill rate. By fitting a variety of different candidate tiles in this step, our algorithm is able to subsequently use combinatorial methods, described in Section 4.3, to efficiently select a subset of tiles that fits within the user-specified resource budget.

We first bin all coalesced points into quadtrees imposed over the faces of a cube centered around the viewing volume. For all quadtree cells, including interior nodes, we fit tiles to the points inside their extents using a clustering algorithm inspired by the method of Cohen-Steiner et al. [2004], and use these tiles as candidates.

Our algorithm partitions the points within each cell into clusters, $S_1, \ldots, S_k$, with associated tiles, $t_1, \ldots, t_k$, to solve

$$\underset{S_1,\ldots,S_k,t_1,\ldots,t_k}{\text{minimize}} \sum_{i=1}^{k} E_{\text{cluster}}(S_i, t_i).$$

We use a k-means style algorithm, alternating between two steps to reduce the energy function:

Tile Assignment. Points are greedily assigned to the tile with the lowest energy.
Tile Refinement. Tile parameters are optimized to reduce total energy given the current partitioning of points.

We iteratively apply these steps within a divisive clustering method to fit $1, 2, \ldots, K$ tiles to each cell. That is, we begin by fitting only a single tile to all of the points inside the cell and periodically split the highest-energy tile into two tiles, with a new tile initialized with fronto-parallel orientation and position derived from the highest-energy point. As this process progresses, each of the K clusterings is emitted as a candidate tiling for that cell, to be considered during tile selection.

Although we wish to minimize the energy $E(p, t)$ (Equation 2), optimizing it presents two challenges. First, we have found that direct optimization of $E_{\text{cluster}}(S_i, t) = \sum_{p \in S_i} E(p, t)$ with tiles parameterized by plane equation coefficients is prone to finding bad local minima. Additionally, not all planes correspond to valid tiles—for example, those at grazing angles may not intersect their associated quadtree frustum along the four relevant edges.

Therefore, we parameterize each tile t_i as a disk with a center, $c_i \in \mathbb{R}^3$, and normal, $n_i \in \mathbb{R}^3$. We then optimize with the following when generating candidate tiles:[1]

$$E_{\text{cluster}}(S_i, t_i) = \sum_{p \in S_i} \left(E(p, t_i) + \gamma R(p, t_i)\right) + B(t_i)$$

where $R(p, t_i)$ is a term designed to improve convexity by penalizing distance to the tile center:

$$R(p, t_i) = \frac{\|c_i - t_{\text{hit}}(p)p\|^2}{\|c_i\|^2}.$$

$B(t)$ imposes a penalty on grazing-angle tiles:

$$B(t) = \sum_{v \in V_t} \max\left(r_{min} - t_{\text{hit}}(v), 0\right)^2 + \max\left(t_{\text{hit}}(v) - r_{max}, 0\right)^2,$$

where V_t is the set of four vertices at the corners of the quadtree cell containing t projected onto the unit sphere and r_{min} and r_{max} define the bounds beyond which to impose a penalty for grazing-angle tiles. In practice, we set r_{min} to 1% of the viewing volume radius and r_{max} to the radius of a bounding sphere of the input geometry. We also decay from $\gamma = 0.1$ to $\gamma = 0$ during optimization to avoid getting stuck in bad local minima at the start, and we solve for up to $K = 8$ tiles per cell.

In our implementation, tile assignment is performed with a linear scan over all tiles to assign each point. To refine tile parameters, we use Levenberg–Marquardt via Ceres Solver [Agarwal et al. 2018], which we find to converge to useful optima despite using a non-convex energy function. For efficiency, intermediate tile refinement steps optimize with only a randomly-selected subset of points, which enables trading off quality for processing time.

Before being processed by the selection algorithm, all tiles are dialated by a small amount to ensure that the coarse quad geometry doesn't have seams or cracks.

4.3 Tile Selection

After all of the candidate tiles have been generated, we next select a subset of them that covers the captured environment, minimizes total distortion, and satisfies all resource constraints.

We consider the space of valid selections of tiles that cover the entire scene without unnecessary overlap using the tree-like structure of the generated candidates induced by the quadtree. That is, to select a valid set of tiles, our algorithm must select either, (a) one of the candidate tilings generated for each root quadtree cell, or (b) the union of a valid set of tiles selected for all child quadtree cells. We denote $[s_1, \ldots, s_M]^T = s \in \{0, 1\}^M$ to be a Boolean vector encoding the selection of the M candidate tiles, $t_1, \ldots, t_M$, emitted from the previous stage. We then use ValidTiling(s) to indicate valid selections of tiles.

If there were no resource constraints, then the optimal selection could be found by simply recursing over all quadtrees: for each node, we could compare the costs of its candidate tilings with the best tiling that can be constructed by selecting candidates from its child subtrees. In doing so, we would effectively solve:

$$\begin{aligned} \underset{s}{\text{minimize}} \quad & d^T s \\ \text{subject to} \quad & \text{ValidTiling}(s) \end{aligned}$$

[1]The optimization is actually performed over the coalesced points. In turn, the first two terms of the expression being summed are scaled by the number of points that p represents, giving nearly the same result as if all of the scene sample points were used.

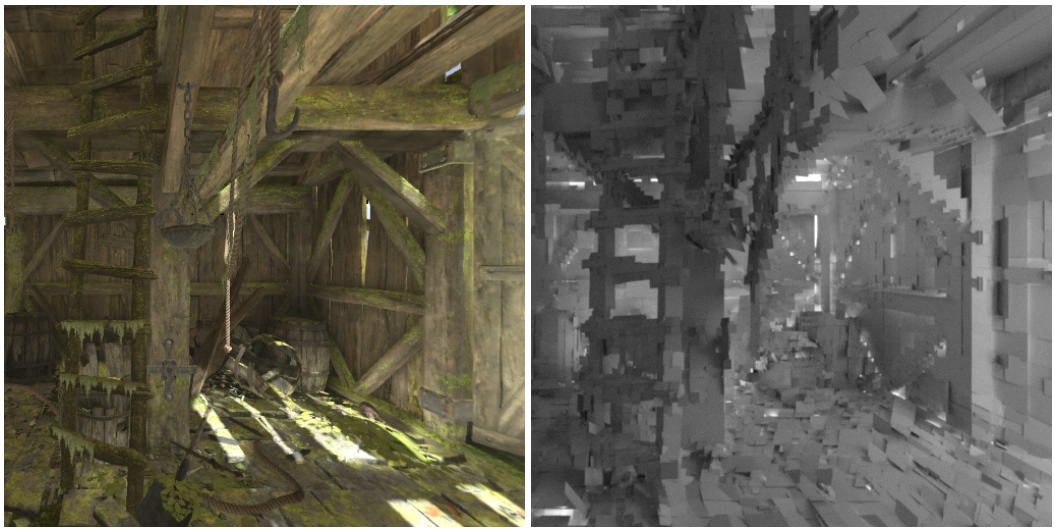

Fig. 6. Left: Smithy scene rendered from a point off-center in the viewing region. Right: Visualization of the quads generated by the tile selection algorithm using a path tracer. Note how the tiles are oriented to fit the scene geometry.

where $[d_1, \ldots, d_M]^T = d$ for the ith candidate tile is computed as a sum over the points, S_i, associated with the tile,

$$d_i = \sum_{p \in S_i} E(p, t_i).$$

However, given our resource constraints, the complete selection problem is more difficult:

$$\begin{aligned} \underset{s}{\text{minimize}} \quad & d^T s \\ \text{subject to} \quad & \begin{cases} \text{ValidTiling}(s) \\ w \cdot s \leq \text{Capacity} \end{cases} \end{aligned}$$

where w is an $(F+2) \times M$ matrix of weights for all candidate tiles.

To solve this problem, we apply a Lagrangian relaxation and optimize with subgradient descent, following the method described by Fisher [2004]. We introduce a slack variable, $\lambda \in \mathbb{R}^{(F+2)}$, and define the Lagrangian problem:

$$\begin{aligned} \text{Selection}_d(\lambda) = \min_s \Big(d^T s + \lambda^T (w \cdot s - \text{Capacity}) \Big) \\ \text{subject to} \begin{cases} \text{ValidTiling}(s) \\ \lambda \geqq 0 \end{cases} \end{aligned}$$

To evaluate $\text{Selection}_d(\lambda)$, we recurse over the quadtrees as previously described. We evaluate the selection of tiles within subtrees in parallel, resulting in efficient execution on multicore CPUs.

We then optimize the dual with subgradient descent:

$$\lambda^* = \arg\max_{\lambda} \text{Selection}_d(\lambda)$$

In some cases, the resulting λ^* may not correspond to a selection that satisfies the resource constraints. If this occurs, we scale λ^* with exponential probing followed by bisection in search of

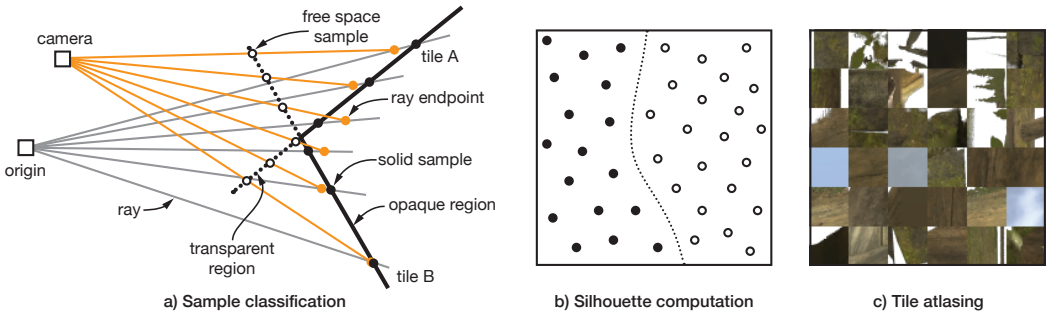

Fig. 7. The texturing algorithm traces rays through the coarse tile geometry to obtain points encoding solid and free space constraints. (a) In the case of a curved surface approximated by intersecting tiles (A and B), these constraints can be determined by draw order, resulting in opaque (solid) and transparent (dotted) regions able to represent the original geometry even if rendering without proper order-independent-transparency. (b) These constraints are resolved into silhouettes by evaluating the implicit surface via nearest neighbor search. (c) The final RGBA textures are then atlased side-by-side.

the best feasible solution along this line. In the limit, increasing λ^* corresponds to selecting a single tile at the root of all quadtrees (in other words, generating a cube with perturbed faces), which should require sufficiently low primitive count, projected area, and fill rate. Figure 6 shows the results of this optimization process with an example scene.

5 TEXTURING

After the coarse tiles have been selected, a texture is generated for each one, encoding an alpha channel—used both for anti-aliasing and to carve out regions of tiles that should be transparent in order to give accurate silhouettes—as well as information for shading, discussed below in Section 5.2.

We use projective texture mapping, parameterizing each tile with homogeneous texture coordinates [Segal et al. 1992]. Doing so gives an even texel density when the scene is observed from the center of the viewing region, and still a fairly even density at other points within it. We have found the projective parameterization to be critical for avoiding artifacts for tiles at grazing angles to the viewer, such as ground planes near the horizon. We parameterize tiles such that each corner of a tile corresponds exactly to a texel center. Tiles can thus be atlased without in-between gutter texels.

Each tile is allocated a square texture, the resolution of which is determined based on the pixel density of the target device such that when viewed from the center of the viewing region, there is at least one texel per pixel. MIP-maps are not used for rendering; unless there is scene geometry close to the boundary of the viewing region (which is undesirable for reasons discussed previously in Section 3), a single texel sampling density generally works well for views throughout the viewing region.

The textures for all tiles are then atlased by first sorting from largest to smallest and then greedily packing them into levels, using an approach similar to that of Lodi et al. [2004]. If the user would like to place a strict limit on texture memory used, we adaptively downsample the texture depending on the content of each tile, as described in Section 5.3.

5.1 Silhouette Determination

The problem of silhouette determination is central to generating high-quality textures; it is necessary to determine which regions of a tile represent solid geometry and which correspond to free space.

We encode this information in alpha textures used to 'clip' silhouettes out of the rendered tiles, in a manner inspired by the tree rendering method of Kharlamov et al. [2007].

Recall that the input to our algorithm is a collection of samples of the scene surfaces. In this context, it may be tempting to consider applying point splatting methods [Zwicker et al. 2001] to re-render the scene into each quad, as was done for texturing billboard clouds [Décoret et al. 2003]. However, such methods can suffer from edge-thickening artifacts, systematically overestimating solid regions, which we have found to limit visual quality for environments with thin structures.

Fundamentally, there is ambiguity inherent in determining silhouettes from a pure point-based method. That is, the absence of a point is not easily distinguishable from a break in the sampled surface. Resolving it is complicated by both the irregular sampling pattern induced by the multiple cameras used for scene capture as well as the undersampling of partially-occluded surfaces.

To overcome these ambiguities, we build on the concept of space carving [Kutulakos and Seitz 1999], taking advantage of the free-space information implicit in the scene sampling rays. We combine solid constraints, derived from a point-based interpretation, with free-space constraints, derived from a ray-based interpretation, to accurately resolve silhouettes.

We define solid and free-space constraints as points on the surface of a tile. The silhouette of a tile is then the implicit surface that is equidistant from the two. By sampling this implicit silhouette, we can estimate coverage to yield alpha values. In doing so, we are able to faithfully represent even subpixel geometry, with contributions from multiple scene samples providing a stochastic supersampling of object silhouettes.

We iterate over all scene samples to accumulate constraints for all tiles as texture-space points. Solid constraints are determined by tracing rays from the center of the viewing region through each sample's endpoint (see Figure 7). The intersection closest to the endpoint is treated as a primary solid constraint. Additional intersections for which $|t_{hit} - 1| < thresh_{solid}$ yield secondary solid constraints, where t_{hit} is the relative distance to the point and $thresh_{solid}$ is a tuning parameter typically set to 0.01. These are used to mitigate cracks at intersecting tiles, similar to the approach developed by Decoret et al. [2003].

To obtain free-space constraints, we then trace rays through each scene sample from its ray's origin. We consider the order in which tiles would be composited along such a ray in final rendering. Free-space constraints result from intersections with tiles to be composited over the tile of the primary solid constraint. We further limit free-space constraints to only those for which $|t_{hit} - 1| > thresh_{solid}$ and also subtract these from the set of solid constraints.

By defining free-space constraints in terms of compositing order, we can model limitations of the method used for rendering the tiles. That is, if compositing is performed according to a fixed draw order, which can be desirable for reasons described in Section 6, then free-space constraints are derived as such. This approach works surprisingly well. As shown in Figure 7, our algorithm is able to generate appropriate cutouts when texturing pairs of intersecting tiles representing both convex and concave geometry. At tile intersections, where compositing order is reversed, both tiles usually have the same color, so compositing errors there do not result in visible artifacts.

Constraint positions are quantized in the texture-space of each tile and then encoded in a pair of supersampled bitmaps. Pixels containing both a solid and a free-space constraint are omitted from subsequent steps, removing the ambiguity of coincident constraints. These bitmaps are compact by construction, so our algorithm is able to stream through a large number of scene samples while maintaining a limited in-memory representation.

Finally, these per-tile constraints are resolved into alpha textures. We use multiple locations within each texel to sample the implicit silhouette, evaluating it by performing a nearest neighbor search of solid and free-space constraints. The resulting estimated fractional coverage is then used as the texel's alpha value.

5.2 Shading and Lighting

The shading information is either an RGB value that encodes outgoing radiance at the surface, computed from the original renderer's output colors, or is a surface normal and BRDF parameters, saved out by the renderer instead of color to make it possible to render a g-buffer [Saito and Takahashi 1991] of the scene for dynamic shading and lighting at runtime.

To compute these values, we also iterate over all of the scene geometry samples, using the (pre-quantization) solid constraints obtained as before. For each, we apply a customizable reconstruction filter (e.g. a Gaussian kernel), and accumulate the sample values into a weighted buffer allocated for each tile. After all scene samples have been processed, the sums of weighted values are normalized to yield final filtered values. Because we obtain solid constraints by consistently projecting scene point samples towards the center of the viewing volume (as opposed to projecting them from their respective generating cameras), our method avoids ghosting artifacts that can arise in classical projective texturing.

In order to provide reasonable values for regions that were undersampled during scene capture due to foreground occluders, we inpaint texels that did not receive any color or g-buffer contributions, diffusing nearby values by solving a discretized Laplacian equation. We optimize this process with a multigrid method, performing several Gauss–Seidel iterations at each level.

5.3 Content adaptive texturing

In addition to the constraints on geometric complexity described in Section 4, we also allow the user to specify a maximum size for the texture atlas. To allocate texels to tiles in a way that maximizes final image quality subject to this size constraint, we apply an optimization approach that minimizes overall distortion, making use of the fact that tiles with low-frequency textures can be represented with lower resolutions.

For each tile $t \in T$ we compute a set S_t of texture representations. We generate representations that correspond to all image sizes smaller than or equal to the initial texture size. Sizes are optionally quantized according to the block size of the texture compression format to be used.[2] This quantization ensures that tiles do not straddle block boundaries when atlassed, which is necessary for good compression quality. The optimization algorithm then chooses a single representation from each set.[3]

To evaluate the distortion $D(x_t)$ introduced by a texture representation $x_t \in S_t$, we first apply a low-pass filter and downsampling to get x_t. Then we upsample x_t back to the initial texture size using the same texture magnification algorithm as in the interactive renderer, i.e. `GL_LINEAR`. Finally, we compute a distortion function, $D(x_t)$ as the sum of squared differences between the initial texture and the upsampled representation.

The constrained optimization problem we wish to solve can be expressed as selecting the set of representations $\{x_t^*\}$ as

$$\{x_t^*\}_{t\in T} = \underset{\text{all choices of } x_t\in S_t, t\in T}{\arg\min} \sum_{t\in T} D(x_t) \qquad (3)$$
$$\text{subject to: Layout}(\{x_t\}_{t\in T}) \text{ is within the size bounds.}$$

We solve this problem using Lagrangian relaxation with the method of Everett [1963] to implement rate-distortion optimization.

[2] We typically use ASTC formats [Nystad et al. 2012] as they provide the necessary bit depth for the alpha-encoded silhouettes and have hardware support on mobile GPUs.

[3] Kraus and Ertl [2002] similarly simplified textures by downsampling until an error limit was reached, though they used a pre-selected error limit for all texture tiles rather than maximizing quality for a given texture resolution as we do.

Fig. 8. Gallery of test scenes. From left to right: a) The Landscape scene from PBRT, an environment with over 3.1 billion instanced triangles that takes hours to render in a ray tracer. b) Pagoda [Manufactura K4 2015], a complex outdoor environment designed to run on high-end desktop hardware. c) The Smithy scene [Unity Technologies 2015], an indoor environment with rich geometry close to the boundary of the viewing region. d) Half Dome from Google Earth, captured at desktop-level quality. e) The Colosseum captured from Google Earth.

6 REAL-TIME RENDERING

The final textured tiles are rendered using two triangles per tile, blending with over-compositing [Porter and Duff 1984]. Alpha blending is relatively inexpensive on the tile-based rendering architectures common in mobile GPUs and provides a mechanism for high-quality edge anti-aliasing, especially critical for VR rendering. While approximate order-independent-transparency methods have been demonstrated on mobile devices [Bjørge et al. 2014], the OpenGL extensions they rely on are not available on all GPUs. Hence, we rely on Painter's algorithm and render without backface culling, compositing back-to-front with a fixed quad ordering from the center of the viewing region.

Dynamic content is supported by rendering our quad tiles after any such objects have been rendered to the depth buffer. If it is necessary to composite dynamic alpha-cutout objects with our representation, alpha-to-coverage can be used with multisampling and a proper depth-buffer for visibility, albeit with decreased silhouette anti-aliasing quality.

7 RESULTS

We evaluated our algorithm with a collection of five scenes. Rendered images of them after simplification by our pipeline are shown in Figure 8. All are high-fidelity approximations of the original scenes and do not show temporal artifacts when the viewer moves (see the supplemental video for examples with camera motion).

All scenes were captured using 16–64 viewpoints at 1024^2 or 1536^2 resolution with high-res images up to 4096^2 for the center viewpoint. Unless otherwise specified, we processed all of these scenes with $L_{\text{tris}} = 72k$, $L_{\text{area}} = 3$, $L_{\text{peakOD}} = 5$, and a 4096^2 texture atlas size. Empirically, we have found this resource budget to be enough to represent most scenes without artifacts—even scenes with high depth complexity, where the greater geometric approximation error resulting from the compression of multiple layers, e.g. with foliage, is less perceptible. With these constraints, the total output size after gzip compression is under 20 MB—small enough to afford packing hundreds of scenes encoded using our approach in a single application.

For scenes captured from game engine content, Pagoda and Smithy, we disabled post processing screen-space effects, such as bloom and dynamic exposure, during capture. Although rendering these effects on top of our tile representation is possible, these effects are also disabled for all comparisons. Note that our scenes only contain small and/or distant specular objects, so all content can be effectively processed by our system which bakes these highlights as they appear from the center of the viewing region.

Table 1. Scene statistics. Note that input triangles are measured in millions, while output triangle are measured in thousands. Unity and Google Earth use culling and LOD mechanisms and therefore have view-dependent triangle counts. We provide measured ranges for Pagoda and Smithy and estimated averages for the two Google Earth scenes.

	Landscape	Pagoda	Smithy	Half Dome	Colosseum
Input triangles (M)	3,100	4.0–12.7	0.6–1.0	~ 15	~ 15
Output triangles (k)	72.0	68.8	70.8	72.0	72.0
Output texels (M)	16.8	16.8	16.8	16.8	16.8
Output texel occupancy (%)	58.8	62.8	79.1	76.3	72.4
Avg. overdraw (×)	3.0	3.8	3.1	3.5	3.2
Peak overdraw (×)	5.6	5.4	4.1	4.5	4.0
Memory (Mb)	13.6	16.0	16.7	14.1	13.9
PSNR (dB)	40.7	49.8	54.3	49.4	47.6
MSSIM	0.74	0.88	0.92	0.86	0.86
Processing time (s)	1,296	1,152	848	741	966
Average render time (ms)	6.7	5.8	5.2	5.3	5.1

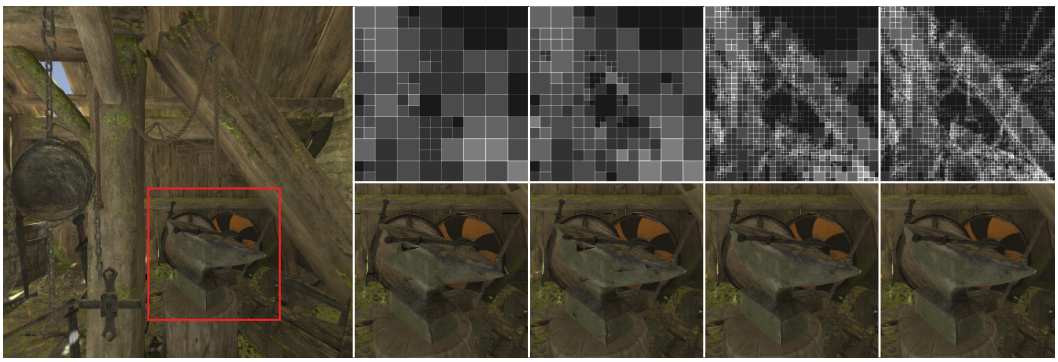

Fig. 9. The effect of approximating the Smithy scene with varying numbers of tiles. (a) an overview of the scene approximated with 127.4 k triangles. (b-e) The top row shows a visualization of selected tiles for different triangle counts, rendered with additive blending such that brighter regions indicate more layers of tiles and the bottom row shows the cropped region, rendered from an off-center position in the viewing region. Note how the tiling algorithm simultaneously solves for the size of the tiles and the number of tiles per cell, subdividing finely at silhouettes to avoid storing empty texels and rendering with unnecessary pixel fill. Cracks and missing geometry, visible at low triangle counts, disappear at 31.8 k triangles.

We evaluated performance with a minimal VR application running on a Pixel 2 phone with a Snapdragon 835 SoC and Adreno 540 GPU as a reasonable proxy for forthcoming 6DoF integrated headsets; see Table 1 for results. All scenes are rendered in under 7 ms per frame, low enough for VR applications and with headroom for additional dynamic content.

This performance is unsurprising given the final triangle count and actual overdraw for each scene, which we measured from each scene-capture viewpoint. Note that use of the entire triangle budget is not guaranteed by our algorithm, since additional tessellation also results in slight increases in fill rate due to the dilation of tiles. Peak fill rate is slightly exceeded for two scenes as a result of limitations in our approximate constraint, but in practice, these exceptions are not

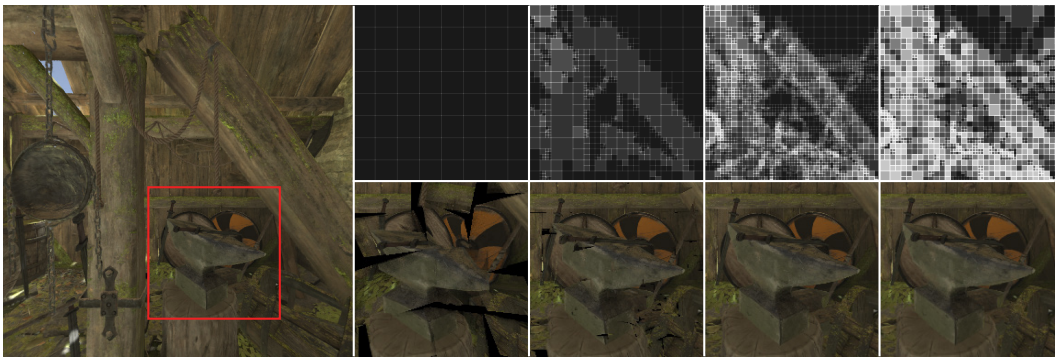

Fig. 10. The effect of varying L_{area} for the SMITHY scene. (a) an overview of the scene with $L_{\text{area}} = 3$ and 127.8 thousand triangles. (b-e) The top row shows the tile structure and overdraw for different overdraw limits and the bottom row shows the cropped region rendered from an off-center position in the viewing region. Cracks mostly disappear at $L_{\text{area}} \geq 3$ overdraw.

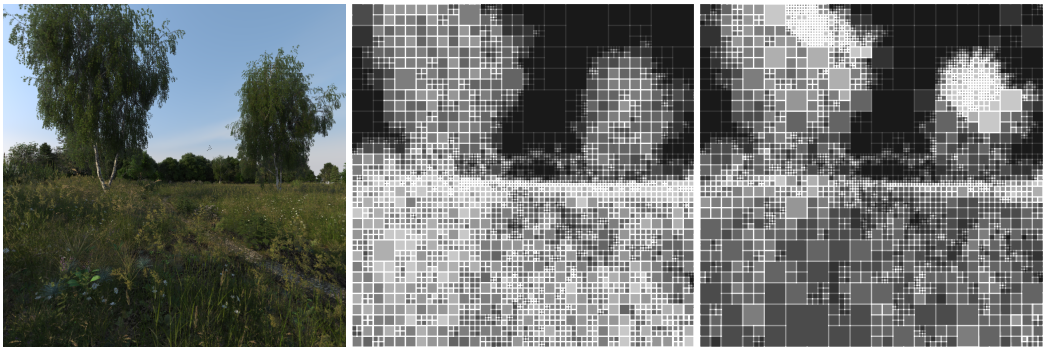

Fig. 11. The LANDSCAPE scene rendered from the center of the viewing volume (left), a visualization of tiles generated without peak fill rate constraints (middle), and a visualization of tiles generated with peak fill rate constraints (right). Notice how tiles are redistributed to allocate fewer layers to the grass and more layers to the trees, where rendering cost can be amortized with the sky.

very significant when compared to the overdraw resulting when no such constraints are placed, illustrated in Figure 11.

To evaluate the fidelity of our reconstruction, we rendered our output from the poses of each of the viewpoints used for scene capture. We then compared the differences between our re-rendered scene samples and the originally-captured scene samples with PSNR and MSSIM [Wang et al. 2004]. We found that most of the reconstruction error is due to distortion of silhouette boundaries when rendered from off-center viewpoints, as can be seen in Figure 12.

In Figures 9 and 10, we show the result of using a range of triangle limits and average overdraw constraints for the SMITHY scene. At below 31.8 k triangles and $L_{\text{area}} \geq 3$, artifacts begin to appear as cracks. See the supplemental video for better visualization of these examples with camera motion.

This effect can be seen in the plot of MSSIM in Figure 13. When the average overdraw constraint is decreased, quality degrades significantly, and at extremely low values the system may even fail to find a tiling satisfying those constraints. Note that at higher levels of L_{area}, the texture size

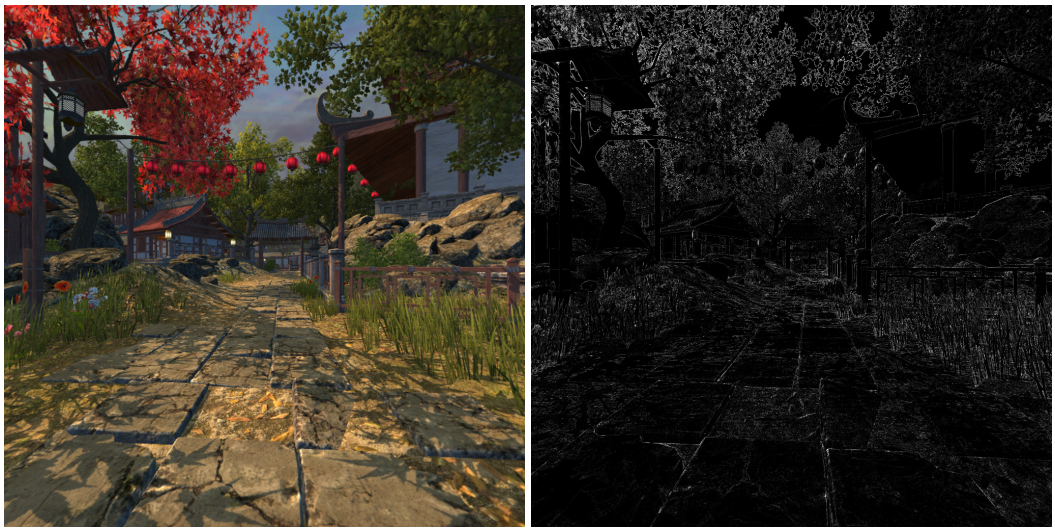

Fig. 12. Left: The Pagoda scene rendered from one of the scene capture viewpoints. Right: Visualization of MSE, tone-mapped to exaggerate error.

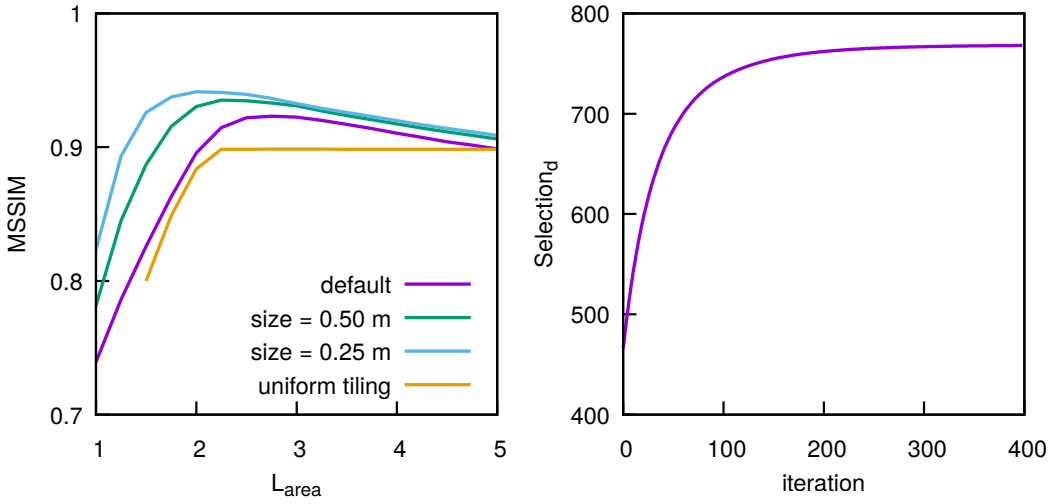

Fig. 13. Left: SSIM of rendered images for the Smithy as a function of the average overdraw constraint. The default setting is a $(1.0\text{m})^3$ viewing region. We show the effect of shrinking this to $(0.5\text{m})^3$ and (0.25m^3). We also show the same scene processed by tiling on a uniform grid, as opposed to the quadtree structure. Right: A graph of convergence of the dual variable, $\text{Selection}_d(\lambda)$, optimized during tile selection for the Smithy processed with default values.

constraint begins to limit quality. This is because permitting more overdraw results in scenes with more layers of tiles providing a better fit of the point sampled geometry. However, these additional layers require more texels to be faithfully represented. That our tiling algorithm cannot directly model effects of texture compression on reconstruction quality is a limitation of our approach.

Table 2. Processing time breakdown for the SMITHY scene on a 16-core workstation. Total time is 14 minutes and 8 seconds.

Stage	Time (s)	Time (%)
Point merging	70	8.3%
Candidate tile generation	325	38.3%
Tile selection	4	0.5%
Texture generation	71	8.4%
Content adaptive texturing	46	5.4%
ASTC compression	265	31.2%
Other	67	7.9%

We also show the result of varying the viewing region size in Figure 13. As expected, larger regions demand more resources (overdraw) to achieve the same reconstruction quality. So, for practical applications on resource-constrained systems, viewing region size is limited by this quality degradation, along with the limit that our algorithm is only well-defined for viewing regions that do not overlap with scene geometry.

To analyze the benefit of our quadtree-based tiling algorithm, we replace the quadtree with a uniform grid, corresponding to only the sixth layer of the default quadtree. As can be seen in Figure 13, this uniform tiling performs significantly worse than the full quadtree and even fails to find a solution at $L_{\text{area}} \leq 1.25$.

Table 1 also reports texture occupancy of our final atlases—the percentage of texels that are in fact used by the generated quads. Occupancy is high because the tiling step implicitly solves for geometry that minimizes empty texels around silhouettes and because the projectively-parameterized square textures can be packed side-by-side, without the inter-chart fragmentation that can arise from packing irregularly-shaped charts [Lévy et al. 2002]. Figure 14 shows the effect of varying the level of compression used for content-adaptive texture compression: our approach outperforms uniform texture subsampling.

Finally, we present a breakdown of total processing time for the SMITHY scene in Figure 2, measured on a workstation with two Intel® Xeon® E5-2690 (2.9 GHz) CPUs, each with 8 cores, and 64 GB of main memory. All major steps of our pipeline are parallelized. Note that the time spent on computing coarse tiles is dominated by candidate generation. The point merging and texturing stages are both bottlenecked by image I/O for reading scene samples. Note too that our combinatorial tile selection algorithm is fast and converges quickly. We plot the convergence of its dual-ascent procedure in Figure 13.

8 CONCLUSION

We have described a robust set of algorithms that can take complex scenes, including offline-rendered environments with billions of primitives, and convert them to a format that can be accurately rendered with under a hundred thousand triangles on low-powered mobile GPUs. Throughout both our algorithms to generate coarse geometric proxies and our algorithms to generate textures for them, we make use of the expectation that the viewer will only view the scene from a fixed viewing region; this assumption is integral to the degree of simplification that we have demonstrated. We have found that this assumption works well for many VR experiences and believe that it is also applicable to many non-VR applications.

Our system has been open sourced and is available at https://github.com/googlevr/seurat.

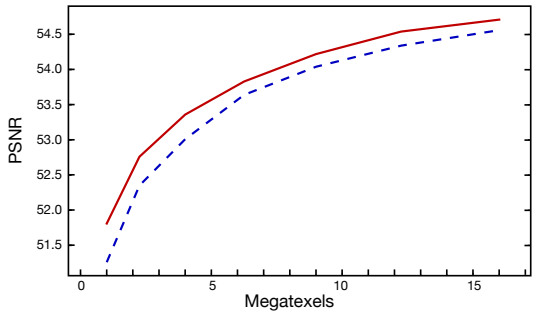

Fig. 14. PSNR of rendered images as a function of texture atlas size for the Smithy scene when using content adaptive texturing (solid red line) and not using it (dashed blue line). For this scene the number of texels may be reduced by 57, 30, 21 and 18 percent for an initial atlas size of 1, 4, 9 and 16 megatexels, respectively.

8.1 Future Work

There are a number of directions for further work to address limitations of our approach and to make it applicable to a wider variety of content.

For cases where a single viewing region has insufficient volume, we have found that generating multiple adjacent regions and then selecting between them based on the viewer position has potential, but seamless transitions between them are an open problem. Such an approach can be practical for applications with up to a few hundred locations, but not for large open worlds, for which a compression scheme that exploits the redundancy between nearby viewing volumes would be required.

View-dependent shading is only supported in our current implementation via the g-buffer approach described in Section 5, which means that the lighting and shading are limited by the capabilities of the GPU, rather than being based on high-quality shading computed by the original renderer. To address this limitation, we would like to investigate generating mobile-friendly compressed representations of the outgoing radiance field on scene surfaces. We note that the limited viewing region assumption makes this problem easier than it is in general: highly specular objects that are far away from the viewer exhibit limited shading variation, since the range of viewing directions for them is limited.

Finally, we would like to extend our approach to animated scenes, which would require temporally-coherent geometry generation and packing the textures such that a video codec can efficiently compress them. Addressing these issues would make it possible to view film-quality animated CGI movies in VR, which we believe will be an important use case for the medium.

ACKNOWLEDGMENTS

We thank early users of our simplification tool, including ILMxLAB, Seismic Games, and the Google Earth VR team, for their valuable feedback on practical use of the system. Thanks also to Jessica Liu for supporting the project and to Nathan Martz for his help and early enthusiasm. Finally, we thank Hugues Hoppe and our anonymous reviewers for their thoughtful feedback on the paper.

REFERENCES

Sameer Agarwal, Keir Mierle, and Others. 2018. Ceres Solver. (2018). http://ceres-solver.org

Alcon Interactive. 2018. Blade Runner: Revelations. (2018). https://play.google.com/store/apps/details?id=com.alconinteractive.brvr

Robert Anderson, David Gallup, Jonathan T. Barron, Janne Kontkanen, Noah Snavely, Carlos Hernández, Sameer Agarwal, and Steven M. Seitz. 2016. Jump: Virtual Reality Video. *ACM Trans. Graph.* 35, 6, Article 198, 13 pages.

C. Andujar, P. Brunet, A. Chica, I. Navazo, J. Rossignac, and A. Vinacua. 2004. Computing Maximal Tiles and Application to Impostor-Based Simplification. *Computer Graphics Forum* 23, 3 (2004), 401–410.

S. Behrendt, C. Colditz, O. Franzke, J. Kopf, and O. Deussen. 2005. Realistic real-time rendering of landscapes using billboard clouds. *Computer Graphics Forum* (2005).

Marius Bjørge, Sam Martin, Sandeep Kakarlapudi, and Jan-Harald Fredriksen. 2014. Efficient Rendering with Tile Local Storage. In *ACM SIGGRAPH 2014 Talks (SIGGRAPH '14)*. Article 51, 1 pages.

Kevin Boos, David Chu, and Eduardo Cuervo. 2016. FlashBack: Immersive Virtual Reality on Mobile Devices via Rendering Memoization. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '16)*. 291–304.

David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. 2004. Variational Shape Approximation. *ACM Trans. Graph.* 23, 3 (Aug. 2004), 905–914.

Cyril Crassin, Fabrice Neyret, Sylvain Lefebvre, and Elmar Eisemann. 2009. GigaVoxels: Ray-guided Streaming for Efficient and Detailed Voxel Rendering. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games (I3D '09)*. 15–22.

Philippe Decaudin and Fabrice Neyret. 2009. Volumetric Billboards. *Computer Graphics Forum* 28, 8 (2009), 2079–2089.

Xavier Décoret, Frédo Durand, François X. Sillion, and Julie Dorsey. 2003. Billboard Clouds for Extreme Model Simplification. *ACM Trans. Graph.* 22, 3 (July 2003), 689–696.

Hugh Everett. 1963. Generalized Lagrange Multiplier Method for Solving Problems of Optimum Allocation of Resources. *Operations Research* 11, 3 (1963), 399–417.

Marshall L. Fisher. 2004. The Lagrangian Relaxation Method for Solving Integer Programming Problems. *Management Science* 50, 12 (2004), 1861–1871.

Anton L. Fuhrmann, Eike Umlauf, and Stephan Mantler. 2005. Extreme Model Simplification for Forest Rendering. In *Proceedings of the First Eurographics Conference on Natural Phenomena (NPH'05)*. Eurographics Association, 57–67.

Michael Garland and Paul S. Heckbert. 1997. Surface Simplification Using Quadric Error Metrics. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '97)*. 209–216.

Enrico Gobbetti and Fabio Marton. 2005. Far Voxels: A Multiresolution Framework for Interactive Rendering of Huge Complex 3D Models on Commodity Graphics Platforms. In *ACM SIGGRAPH 2005 Papers (SIGGRAPH '05)*. 878–885.

Google Inc. 2018. Expeditions. (2018). https://play.google.com/store/apps/details?id=com.google.vr.expeditions

Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. 1996. The Lumigraph. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '96)*. 43–54.

J. Huang, Z. Chen, D. Ceylan, and H. Jin. 2017. 6-DOF VR videos with a single 360-camera. In *2017 IEEE Virtual Reality (VR)*. 37–44.

H. Ishiguro, M. Yamamoto, and S. Tsuji. 1990. Omni-directional stereo for making global map. In *Proceedings Third International Conference on Computer Vision*. 540–547.

Stefan Jeschke and Michael Wimmer. 2002. Textured Depth Meshes for Real-Time Rendering of Arbitrary Scenes. In *Eurographics Workshop on Rendering*, P. Debevec and S. Gibson (Eds.). The Eurographics Association.

Alexander Kharlamov, Iain Cantlay, and Yury Stepanenko. 2007. Next-Generation SpeedTree Rendering. In *GPU Gems 3* (first ed.), Hubert Nguyen (Ed.). Addison-Wesley Professional.

Babis Koniaris, Ivan Huerta, Maggie Kosek, Karen Darragh, Charles Malleson, Joanna Jamrozy, Nick Swafford, Jose Guitian, Bochang Moon, Ali Israr, and Kenny Mitchell. 2016. IRIDiuM: Immersive Rendered Interactive Deep Media. In *ACM SIGGRAPH 2016 VR Village (SIGGRAPH '16)*. Article 11, 2 pages.

Babis Koniaris, Maggie Kosek, David Sinclair, and Kenny Mitchell. 2017. Real-time Rendering with Compressed Animated Light Fields. In *Proceedings of the 43rd Graphics Interface Conference (GI '17)*. 33–40.

Martin Kraus and Thomas Ertl. 2002. Adaptive Texture Maps. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware (HWWS '02)*. Eurographics Association, 7–15.

K.N. Kutulakos and S.M. Seitz. 1999. A Theory of Shape by Space Carving. In *The Proceedings of the Seventh IEEE International Conference on Computer Vision*. 307–314.

Dylan Lacewell, David Edwards, Peter Shirley, and William B. Thompson. 2006. Stochastic Billboard Clouds for Interactive Foliage Rendering. *Journal of Graphics Tools* 11 (1 2006), 1–12.

Marc Levoy and Pat Hanrahan. 1996. Light Field Rendering. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '96)*. 31–42.

Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérome Maillot. 2002. Least Squares Conformal Maps for Automatic Texture Atlas Generation. *ACM Trans. Graph.* 21, 3 (July 2002), 362–371.

A. Lodi, S. Martello, and D. Vigo. 2004. Models and Bounds for Two-Dimensional Level Packing Problems. *Journal of Combinatorial Optimization* 8, 3 (9 2004), 363–379.

B. Luo, F. Xu, C. Richardt, and J. H. Yong. 2018. Parallax360: Stereoscopic 360° Scene Representation for Head-Motion Parallax. *IEEE Transactions on Visualization and Computer Graphics* (2018).

Manufactura K4. 2015. Asia – Far East Environment. https://assetstore.unity.com/packages/3d/environments/asia-far-east-environment-21298. (2015). Accessed on 2018-01-22.

Harald Niederreiter. 1992. *Random Number Generation and Quasi-Monte Carlo Methods.* Society for Industrial and Applied Mathematics.

J. Nystad, A. Lassen, A. Pomianowski, S. Ellis, and T. Olson. 2012. Adaptive Scalable Texture Compression. In *Proceedings of the Fourth ACM SIGGRAPH / Eurographics Conference on High-Performance Graphics (EGGH-HPG'12)*. Eurographics Association, 105–114.

S. Peleg, M. Ben-Ezra, and Y. Pritch. 2001. Omnistereo: panoramic stereo imaging. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 23, 3 (Mar 2001), 279–290.

Eric Penner and Li Zhang. 2017. Soft 3D Reconstruction for View Synthesis. *ACM Trans. Graph.* 36, 6, Article 235 (Nov. 2017), 11 pages.

Hanspeter Pfister, Matthias Zwicker, Jeroen van Baar, and Markus Gross. 2000. Surfels: Surface Elements As Rendering Primitives. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '00)*. 335–342.

Matt Pharr. 2017. Mobile VR: Challenges and Opportunities. (2017). http://pharr.org/matt/mobilevr.pdf ACM I3D Keynote.

Thomas Porter and Tom Duff. 1984. Compositing Digital Images. *SIGGRAPH Comput. Graph.* 18, 3 (Jan. 1984), 253–259.

Liu Ren, Hanspeter Pfister, and Matthias Zwicker. 2002. Object Space EWA Surface Splatting: A Hardware Accelerated Approach to High Quality Point Rendering. *Computer Graphics Forum* 21, 3 (2002), 461–470.

C. Richardt, Y. Pritch, H. Zimmer, and A. Sorkine-Hornung. 2013. Megastereo: Constructing High-Resolution Stereo Panoramas. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*. 1256–1263.

Takafumi Saito and Tokiichiro Takahashi. 1991. NC Machining with G-buffer Method. In *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '91)*. 207–216.

Gernot Schaufler. 1998. Image-based Object Representation by Layered Impostors. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST '98)*. 99–104.

Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, and Paul Haeberli. 1992. Fast Shadows and Lighting Effects Using Texture Mapping. In *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '92)*. 249–252.

Jonathan Shade, Steven Gortler, Li-Wei He, and Richard Szeliski. 1998. Layered Depth Images. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '98)*. 231–242.

Unity Technologies. 2015. The Blacksmith Environments. https://assetstore.unity.com/packages/3d/environments/asia-far-east-environment-21298. (2015). Accessed on 2018-01-22.

Alex Vlachos. 2016. Advanced VR Rendering Performance. (2016). https://alex.vlachos.com/graphics/Alex_Vlachos_Advanced_VR_Rendering_Performance_GDC2016.pdf GDC Talk.

Zhou Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. 2004. Image Quality Assessment: From Error Visibility to Structural Similarity. *Trans. Img. Proc.* 13, 4 (April 2004), 600–612.

Andrew Wilson and Dinesh Manocha. 2003. Simplifying Complex Environments Using Incremental Textured Depth Meshes. In *ACM SIGGRAPH 2003 Papers (SIGGRAPH '03)*. 678–688.

Daniel N. Wood, Daniel I. Azuma, Ken Aldinger, Brian Curless, Tom Duchamp, David H. Salesin, and Werner Stuetzle. 2000. Surface Light Fields for 3D Photography. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '00)*. 287–296.

Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, and Markus Gross. 2001. Surface Splatting. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01)*. 371–378.