

# Immersive Light Field Video with a Layered Mesh Representation

MICHAEL BROXTON\*, JOHN FLYNN\*, RYAN OVERBECK\*, DANIEL ERICKSON\*, PETER HEDMAN, MATTHEW DUVALL, JASON DOURGARIAN, JAY BUSCH, MATT WHALEN, and PAUL DEBEVEC, Google

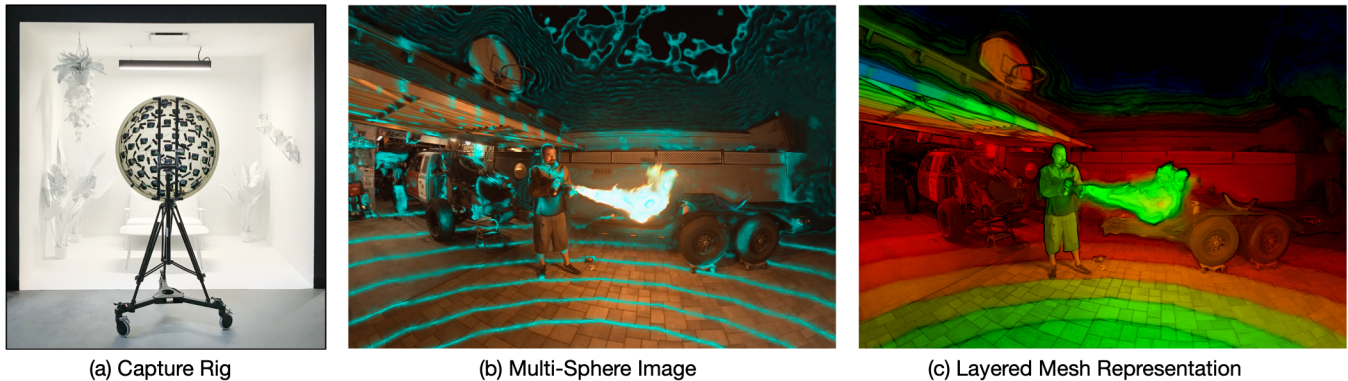


Fig. 1. Our light field capture rig and scene representations. (a) We record immersive light field video using 46 action sports cameras mounted to an acrylic dome. (b) Using deep view synthesis we infer an RGBA multi-sphere image (MSI) from the light field views. Every 10th spherical shell is highlighted. (c) We convert groups of MSI layers into Layered Meshes (each shown as a different color), which are texture atlased and compressed into light field video.

We present a system for capturing, reconstructing, compressing, and rendering high quality immersive light field video. We accomplish this by leveraging the recently introduced DeepView view interpolation algorithm, replacing its underlying multi-plane image (MPI) scene representation with a collection of spherical shells that are better suited for representing panoramic light field content. We further process this data to reduce the large number of shell layers to a small, fixed number of RGBA+depth layers without significant loss in visual quality. The resulting RGB, alpha, and depth channels in these layers are then compressed using conventional texture atlasing and video compression techniques. The final compressed representation is lightweight and can be rendered on mobile VR/AR platforms or in a web browser. We demonstrate light field video results using data from the 16-camera rig of [Pozo et al. 2019] as well as a new low-cost hemispherical array made from 46 synchronized action sports cameras. From this data we produce 6 degree of freedom volumetric videos with a wide 70 cm viewing baseline, 10 pixels per degree angular resolution, and a wide field of view, at 30 frames per second video frame rates. Advancing over previous work, we show that our system is able to reproduce challenging content such as view-dependent reflections, semi-transparent surfaces, and near-field objects as close as 34 cm to the surface of the camera rig.

\*Denotes equal contribution.

Authors' address: Michael Broxton, broxton@google.com; John Flynn, jflynn@google.com; Ryan Overbeck, rover@google.com; Daniel Erickson, danerickson@google.com; Peter Hedman, hedman@google.com; Matthew DuVall, matthewduvall@google.com; Jason Dourgarian, jdourgarian@google.com; Jay Busch, jbusch@google.com; Matt Whalen, mswhalen@google.com; Paul Debevec, debevec@google.com, Google.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

© 2020 Copyright held by the owner/author(s).  
0730-0301/2020/7-ART86

<https://doi.org/10.1145/3386569.3392485>

CCS Concepts: • **Computing methodologies** → *Image compression; Neural networks; Virtual reality; Image-based rendering; Computational photography.*

Additional Key Words and Phrases: view synthesis, light fields, image-based rendering, deep learning

## ACM Reference Format:

Michael Broxton, John Flynn, Ryan Overbeck, Daniel Erickson, Peter Hedman, Matthew DuVall, Jason Dourgarian, Jay Busch, Matt Whalen, and Paul Debevec. 2020. Immersive Light Field Video with a Layered Mesh Representation. *ACM Trans. Graph.* 39, 4, Article 86 (July 2020), 15 pages. <https://doi.org/10.1145/3386569.3392485>

## 1 INTRODUCTION

Our visual system is remarkable at perceiving the world around us from a single pair of eyes a few centimeters apart. Part of what makes it so effective is that moving our heads changes our perspective, allowing us to feel a greater sense of depth through motion parallax and a better sense of what the scene is made of by how light plays off of its surfaces. A virtual environment that is able to excite these same senses of motion parallax and view-dependent reflections can be far more immersive and realistic.

Many of today's augmented and virtual reality headsets provide positional tracking, enabling the user's view of the virtual scene to shift perspective properly as they move their head. If the scene is being rendered by a game engine, it is straightforward to feed the positional tracking data into the engine to yield proper motion parallax and view-dependent reflections. But if the scene is a real one recorded by an immersive video camera, changing the perspective is much more complicated. The majority of immersive video cameras deliver panoramic video from a single fixed point of view (e.g. Ricoh Theta Z1, GoPro Max, Insta360 One X) or omnidirectional stereo video fixed to a particular location in space (e.g. Yi Halo, Insta360

Titan, Facebook Surround 360). When these immersive videos are played back, the user can look in any direction, but the position of their view cannot be altered regardless of how the user moves their head. The result is not only less immersive but less comfortable: receiving such inconsistent cues from one's visual and vestibular systems is a cause of motion sickness, which can affect many people in VR headsets. And this problem is unavoidable: just turning one's head to look around produces noticeable shifts in perspective as the eyes translate with the front of the head.

To address this problem, efforts have been made to record immersive video with additional cameras and/or sensors so that the video can be rendered from a range of new perspectives. One strategy is to use multi-view stereo [Furukawa and Hernandez 2015] and/or depth sensors to estimate the 3D depth of the scene in each direction, so that the panoramic imagery can be reprojected with image-based rendering to a novel viewpoint. A drawback of this approach is that edges in the scene are typically not well preserved, and dis-occluded areas around object edges are difficult to fill in accurately. While the reconstructed 3D geometry can be displayed with textures [Hedman et al. 2017], this fails to reproduce view-dependent reflections, assuming instead that everything in the scene has a dull Lambertian reflectance. A different strategy [Milliron et al. 2017; Wilburn et al. 2005] is to record a sampling of the light field using an array of cameras distributed across a surface. When used with modern view interpolation techniques that include per-view depth estimation and view-dependent texture mapping, view-dependent reflections can be preserved [Debevec et al. 1998; Pozo et al. 2019]. However, there is a hardware complexity cost associated with denser camera arrays, and sparse samplings are prone to producing noticeable edge artifacts. For light field video, neither of these strategies has yet been paired with an especially efficient video compression scheme. As a result, these systems typically require specially-designed workstations filled with solid-state memory to play back immersive content.

The recently introduced interpolation techniques of [Flynn et al. 2019; Mildenhall et al. 2019; Zhou et al. 2018] use machine learning to convert sparsely-sampled light fields to Multi-Plane Images (MPIs), which are dense stacks of image planes with RGB color and alpha transparency components (RGBA). These MPIs offer several advantages, in that they are straightforward to render, yield good view reconstruction around edges, and are surprisingly effective at reproducing and smoothly interpolating specularities. (Specular reflections can be rendered on planes behind the object surface, with the surface itself made partially transparent to allow the specular reflections to shine through.) Flynn et al. [2019] applied MPIs to both light field stills and light field video with high-quality results. However, they did not address an immersive field of view, which would be difficult using planar perspective projections, and they did not report a compression and streaming technique necessary to store or deliver such content at scale. On the latter point, the relatively dense 3D volume of RGBA data comprising an MPI can be as large or larger than the original light field camera array data.

In this paper, we develop a practical immersive light field video solution by replacing the MPI planes with spherical shells. We also develop a compression and storage technique that packs them into streamable video and geometry. Instead of MPIs, we modify the training and inference scheme to solve for Multi-Sphere Images

(MSIs) (Figure 1 (b)) that wrap around the viewer with a set of concentric spherical polygon meshes with inward facing RGBA textures. We then use a new layer reduction technique to convert the relatively dense MSIs to a sparse set of Layered Meshes (LMs) (Figure 1 (c)) that consist of surface meshes largely corresponding to the surfaces in the scene. This smaller set of meshes still use RGBA textures and can therefore retain the versatility of MPI alpha blending, but at greatly reduced storage cost since LMs can be compressed using mesh compression and texture atlas. In this way, we can leverage 2D video codecs such as H.264 or VP9 to compress and transmit a compact, faithful representation of the original immersive light field video.

We show light field video results recorded using two 6 DOF camera arrays: the 16-camera system of [Pozo et al. 2019] and a new low-cost hemispherical light field video array comprised of 46 action sports cameras (Figure 1 (a)). We analyze how camera rig geometry affects the ability to represent near-field objects in the light field. We then introduce multi-sphere images as a scene representation for panoramic light field content and show how the MSI and camera rig geometry jointly determine the size of the interpolation volume where valid viewpoints can be rendered. Then, we describe a complete light field video encoding system that leverages deep learning based view interpolation, layer reduction, texture atlas, and conventional video compression. The resulting video stream can subsequently be decompressed and rendered on standard graphics hardware. We thus show the first practical, high-quality, immersive light field video solution that can be streamed and played back on standard AR and VR systems.

## 2 BACKGROUND

We build on significant bodies of work in multi-view stereo [Furukawa and Hernandez 2015], image based rendering [Shum et al. 2007], light field capture [Gortler et al. 1996; Levoy and Hanrahan 1996], and particularly recent advances in learning based view synthesis [Flynn et al. 2016]. Many of these efforts have focused on representing static scenes, but here we focus on approaches that record and play back video content. Such systems face unique challenges due in part to the large volume of data that must be processed and displayed. In designing a system for light field video, we identified the following necessary characteristics:

- (1) The ability to record six degree-of-freedom (6 DOF) content with a relatively sparse array of video cameras.
- (2) View synthesis within a viewing volume diameter of around 70 cm for a wide 180° or greater field of view, appropriate for comfortable viewing while seated.
- (3) Plausible rendering of view-dependent scene content including disocclusions around object edges and thin structures, semi-transparent objects, specular reflections, and mirror surfaces.
- (4) Visually stable results as the viewpoint moves in both space and time.
- (5) A compressible representation suitable for playback on consumer VR and AR hardware.

Although the works we mention below succeed in addressing some of these challenges, we believe that our system is the first to address all of them.

Early work on novel view-synthesis used explicit correspondence between input images to warp them to novel viewpoints. This was first demonstrated using synthetic input images with known depth maps [Chen and Williams 1993] and later using photoconsistency-based matching to establish correspondence between real photographs [McMillan and Bishop 1995]. To represent occlusions in the scene, Shade et al. [1998] introduced a *layered depth image* representation where each pixel represents multiple opaque surfaces with both depth and texture. Zitnick et al. [2004] used correspondence-based image matching to build a similar layered depth representation with alpha matting near object boundaries and successfully applied it to generating free viewpoint video. Later efforts have extended this to provide a much larger range of motion by fusing multiple depth maps into one animated texture mesh [Collet et al. 2015; Dou et al. 2016]. These works were focused on performance capture where the viewer is expected to look into the scene from the outside. Pozo et al. [2019] showed that such methods also work for outward-facing view synthesis, e.g. when the viewer is wearing a head mounted display.

However, most of these methods assume the scene being represented contains only opaque, Lambertian surfaces, and thus are unable to represent view-dependent phenomena such as specular highlights and semi-transparent objects. This has been addressed via methods where not all visual motion in the scene needs to be explained by a single depth value. Flow fields, for example, store the raw pixel-wise correspondence across cameras and also across time [Lipski et al. 2010], although their unconstrained nature introduces warping artifacts near occlusion boundaries and during viewpoint changes. Of course, this problem is greatly reduced if the viewpoint is fixed in advance, as it was in [Anderson et al. 2016], which nevertheless rendered immersive video using binocular disparity cues. Alternatively, a trained neural network can be used to correct artifacts and enhance visual quality of a viewpoint produced with depth-based rendering [Martin-Brualla et al. 2018]. Several other approaches [Sitzmann et al. 2019; Thies et al. 2019] learn a deep representation for visual information in the scene. These approaches, while promising, have not been demonstrated to scale well to high resolution video.

Image-based rendering approaches that employ view-dependent texture projection [Debevec et al. 1998] and/or view-dependent geometry [Davis et al. 2012; Heigl et al. 1999; Overbeck et al. 2018] can reproduce a degree of view-dependent reflection effects. This approach has recently been employed in video playback [Pozo et al. 2019], with earlier methods also showing video results [Buehler et al. 2001], albeit with a coarse proxy geometry shared by all images in the scene. Coarser geometry requires more densely sampled images. In the limit case this leads to capturing a dense 4D light field [Gortler et al. 1996; Levoy and Hanrahan 1996], which can faithfully represent scenes with arbitrarily complex materials and geometry. While this representation has been used to capture video [Wilburn et al. 2001], it's only practical over very narrow viewing baselines or coarse image resolution. This has been addressed by conventional video compression methods [Avramelos et al. 2019] or deep learning methods for super-resolution [Wang et al. 2017], but neither of these approaches produces a viewing volume sufficient for our needs. Perhaps the only example of a high quality, 1 m viewing

baseline light field video system is the Lytro Immerge prototype [Milliron et al. 2017]. However, this system was expensive, not fully documented, and required considerable storage and compute power to play back the video.

Recently introduced layered representations including Soft3D [Penner and Zhang 2017] and multi-plane images (MPIs) [Flynn et al. 2019; Zhou et al. 2018] have yielded attractive light field reconstructions as a volume of RGB plus alpha transparency data. These can be a very good approximation of a 4D light field since they naturally support transparency and approximate the motion of a wide variety of reflections in the scene. Such representations can support a much larger viewing baseline than the dense light fields above [Srinivasan et al. 2019], and even larger viewing baselines have been achieved by blending between multiple multi-plane images [Mildenhall et al. 2019].

We have found that multi-plane image representations are a good fit for the goals above. In particular, both Soft3D [Penner and Zhang 2017] and DeepView [Flynn et al. 2019] have demonstrated the temporal stability of a layered representation for video applications. In this paper we build upon the DeepView algorithm, but have to innovate in two areas to meet our goals. Firstly, we reparametrize MPIs to multi-*sphere* images (MSIs), which enable immersive fields of view. Secondly, we exploit the sparsity in the MSI volume and compress it into a lightweight Layered Mesh representation that can be decoded and rendered on standard hardware. These will be the core topics of the following two sections.

### 3 CAPTURE & RECONSTRUCTION OF SPHERICAL LIGHT FIELDS

We aim to design a light field capture system that supports a sufficiently large viewing volume and accurately reproduces objects that are relatively close to the surface of the camera rig. This supports our overall goal of heightening the 3D motion parallax effect and providing the viewer with a large volume to look around and move freely in without uncomfortable limits on their head motion. In this section we begin by analyzing the geometry of panoramic light field video capture.

#### 3.1 Camera Rig Geometry

First we consider how the the rig radius, camera spacing, and camera field of view (FOV) affect the size of the viewing volume and the closest object that can be reconstructed using view interpolation. For simplicity, we conduct our analysis using the circular 2D rig shown in Figure 2. Extending this analysis to a 3D spherical rig is straightforward.

The figure shows overlapping camera viewing frusta (blue wedges) for evenly spaced cameras on an idealized rig (black semicircle). View interpolation requires that objects be observed by at least two different cameras. We define the distance  $r_c$  (outer dotted circle) to be the closest possible distance where this is true. Notice that this *closest object distance*  $r_c$  depends on both the spacing between cameras on the rig surface and the cameras' FOV. Adding more cameras to the rig reduces the closest object distance and generally increases camera overlap at all distances greater than  $r_c$ , but at the cost of system expense and complexity. It may therefore be necessary to choose a camera spacing (and consequently, a closest

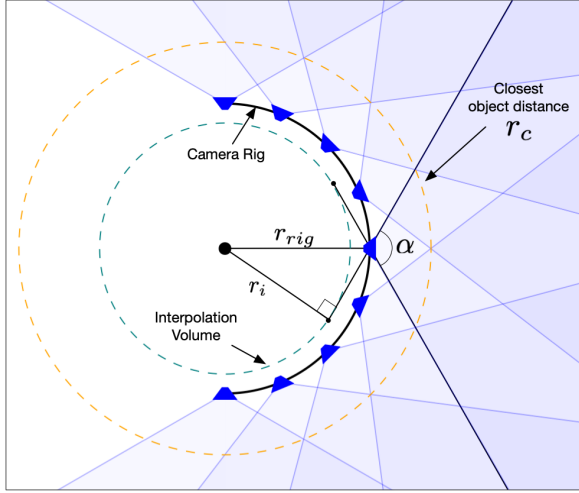


Fig. 2. The view frustra of cameras in a semicircular array, and how they determine the interpolation volume and closest object distance.

object distance) based on practical considerations such as camera cost, complexity of camera synchronization, or the total number of video files that can be easily downloaded and stored for each shot.

Figure 2 also shows the region we call the *interpolation volume* (dotted inner circle), whose radius  $r_i = r_{\text{rig}} \sin(\alpha/2)$  depends only on the FOVs of the cameras. This volume contains the intersection of rays projected backward from the cameras. At any position inside the interpolation volume, ray samples are available in all outward-facing directions, and the view synthesis task is one of interpolation between viewpoints rather than one of extrapolation.

We can use this 2D geometry construction to quickly estimate 6-DOF capture performance for our 3D camera rig and the rig of Pozo et al. [2019]. Our rig has a diameter of 0.92 m with cameras that have a  $120^\circ$  FOV and average inter-camera spacing of 18 cm. This results in an approximate interpolation volume size of  $r_i = 0.4$  m and a closest object distance of  $r_c = 0.66$  m<sup>1</sup>. This places the closest scene objects roughly at “arms length” when the viewer stands at the center of the viewing volume. The 16 camera rig of Pozo et al. is 0.48 m in diameter, with cameras that have a  $180^\circ$  FOV and average spacing of 21 cm. For this rig,  $r_i = 0.24$  m and  $r_c = 0.48$  m. Keep in mind, however, the closest object distance can also be limited by the depth of field of the rig cameras. We’ve found that the cameras we use produce sharp imagery for objects beyond 30 cm from the cameras, so while we can theoretically reconstruct objects 0.1 m closer than this, they may appear slightly blurred.

### 3.2 Multi-Sphere Images

Capturing the scene is only the first step towards a full 6-DOF light field video reconstruction. In this section we introduce our multi-sphere image representation and show how to construct it to be compatible with our capture geometry.

<sup>1</sup>We used these numbers from the 2D rig construction as design guidelines for our 3D rig. We have found that the 3D rig has a slightly smaller 0.7-m interpolation volume due to the smaller 90 degree vertical field of view of our cameras.

Multi-sphere images (MSI’s) are inspired by previous deep learning-based view interpolation algorithms that use multi-plane images (MPI’s) [Flynn et al. 2019; Mildenhall et al. 2019; Zhou et al. 2018]. These approaches train a neural network to produce an MPI from a set of sparse input viewpoints. Once trained, network inference is a relatively inexpensive operation. And unlike earlier deep learning methods such as [Flynn et al. 2016; Kalantari et al. 2016] that require running a network to generate each new view, an MPI is self-contained and can be used without a network to render novel views. Furthermore, MPI’s can be rendered on low-cost graphics hardware: only perspective projection and alpha compositing operations are required. However, MPI’s are fundamentally built upon a rectilinear projection and are therefore best optimized for forward-looking views from a position near their center of projection. To date, no work has demonstrated the use of MPI’s to render panoramic content. Here we describe how MSI’s can replace MPI’s for immersive, wide FOV applications.

An MSI consists of a series of concentric spherical shells, each with an associated RGBA texture map. Like the MPI, the multi-sphere image is a volumetric scene representation. MSI shells exist in three dimensional space, so their content appears at the appropriate positions relative to the viewer, and motion parallax when rendering novel viewpoints works as expected. As with an MPI, the MSI layers should be more closely spaced near the viewer to avoid depth-related aliasing. The familiar inverse depth spacing used for MPI’s yields almost the correct depth sampling for MSI’s, assuming depth is measured radially from the rig center. The spacing we use is determined by the desired size of the interpolation volume and angular sampling density as described in Appendix A.

**3.2.1 MSI Rendering.** For efficient rendering, we represent each sphere in the MSI as a texture-mapped triangle mesh and we form the output image by projecting these meshes to the novel viewpoint, and then compositing them in back-to-front order. Specifically, given a ray  $\mathbf{r}$  corresponding to a pixel in the output view, we first find all ray-mesh intersections along the ray. We denote  $\mathcal{C}_{\mathbf{r}} = \{c_1, \dots, c_n\}$  and  $\mathcal{A}_{\mathbf{r}} = \{\alpha_1, \dots, \alpha_n\}$  as the color and alpha components at each intersection, sorted by decreasing depth. We then compute the output color  $\mathbf{c}_{\mathbf{r}}$  by repeatedly over-compositing these colors. That is,

$$\mathbf{c}_{\mathbf{r}} = \sum_{i=1}^n \alpha_i c_i \prod_{j=i+1}^n (1 - \alpha_j). \quad (1)$$

We parameterize the MSI texture maps using equi-angular sampling, although other parameterizations could be used if it were necessary to dedicate more samples to important parts of the scene.

## 4 LIGHT FIELD VIDEO SYSTEM IMPLEMENTATION

A practical light field video system relies on tight integration between image capture, view interpolation, compression, and rendering. This section contains a detailed description of each of these sub-components in our system, beginning with the construction of our principal light field video rig. We then describe how we adapt the DeepView network to generate Multi-Sphere Images from the light field array data. Next, we convert the per-frame MSIs to Layered Meshes, whose RGBA textures we re-pack into a video texture

atlas. Rendering is done in a straightforward manner through de-compression and scan-converting the textured layers back-to-front.

#### 4.1 Capture Rig

Our principal camera array consists of 46 low-cost Yi 4K action cameras. We placed cameras at the vertices of a v3 icosahedral tiling of a 92-cm diameter hemisphere (see Figure 3). Each camera has a  $120^\circ \times 90^\circ$  field of view, yielding a wide-FOV light field that wraps more than  $180^\circ$  around the viewer. We matched the orientation of each camera with the horizon, as this has multiple benefits: It maximizes the horizontal FOV of the rig, ensures consistent rolling shutter distortions across all cameras and simplifies fabrication. While overlap in both space and time can be increased by explicitly optimizing for per-camera orientations [Poza et al. 2019], we found that our simple strategy results in ample coverage for reconstruction as we use many wide-FOV cameras in our rig. The hemisphere itself is fabricated from a 6-mm thick sheet of acrylic shaped using a draped plastic thermo-forming process. We drilled out 25mm lens openings and mounted the cameras using 3D printed brackets.

A single master camera controls the others via a 2-wire trigger/synchronization cable. Each camera uses its own built-in battery that offers roughly one hour of recording time. A 7-pin USB connection on each camera charges the array and transfers USB 2.0 data. This eliminates the need to physically remove batteries or micro-SD cards, and allows us to copy data from the array in parallel using a collection of high-bandwidth USB3 hubs. The Yi 4K software supports an “array mode” that enables time synchronization across each camera. Once a camera receives the “start recording” signal, it relies on internal timing for image capture. Synchronization tests performed by running the array in a room periodically lit by a camera flash indicate that the cameras begin recording within 4 ms of each other and the intra-array temporal drift stays within 1/2 of a 30 Hz video frame for up to 15 minutes.

We calibrate the camera extrinsics and intrinsics using standard structure from motion techniques [Hartley and Zisserman 2003]. We independently recalibrate every shot to account for jostled cameras between shots and the impact of temperature on camera intrinsics. The rig calibration is assumed constant through the shot, and we calibrate on four well-distributed frames from the captured sequence.

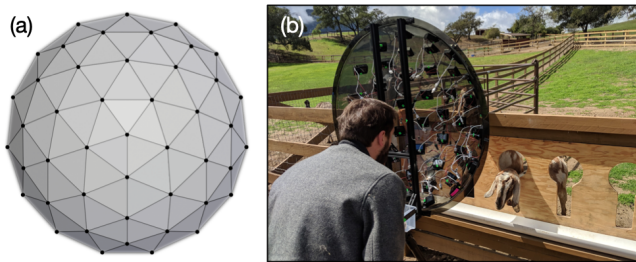


Fig. 3. Our camera array is built using a low-cost, semi-transparent acrylic dome. (a) Cameras are placed at vertices of an icosahedral tiling of a 0.92 m diameter hemisphere. This yields an average inter-camera spacing of 18 cm. (b) The plastic dome acts as a natural viewfinder, making it easy to compose a shot by looking through the surface of the camera rig.

We also exploit the nominal rig pose and calibration as a strong prior to both increase features matches and initialize the calibration solution. Finally, we align image exposures across cameras within the rig using a method similar to [Anderson et al. 2016].

#### 4.2 Learned View Interpolation

The central view synthesis problem we face is how to generate an MSI from a sparse set of input views. This is challenging because resolving depth and reasoning about occlusion often requires jointly considering points in the scene that are far apart from each other but nevertheless appear adjacent in the input views. While no method has investigated MSIs, the recent methods for producing MPis [Mildenhall et al. 2019; Srinivasan et al. 2019; Zhou et al. 2018] rely on 3D convolution with large receptive fields to provide sufficient context to reason about depth and visibility. In contrast, DeepView [Flynn et al. 2019] iteratively refines the MPI by successively incorporating and reasoning about the visibility between the input views and the MPI.

Passing information in this manner allows the network to be agnostic to the specific layer spacing and number of layers. This provides flexibility for training: we found that training at even one quarter of the final output resolution and one quarter the number of final image layers still produced good quality results. Additionally, using 46 input images during training would be prohibitively expensive, even at low resolution. However, DeepView uses repeated max-pooling layers [Qi et al. 2017a,b] to aggregate the features across the input images. This makes the trained network independent to the number of inputs, allowing us to train with a small number of views but evaluate using all of the camera array’s images.

Several changes to DeepView were required for our application. The most significant change was to adapt DeepView to work with multi-sphere images. The original algorithm warped images between the input views and MPI planes using homographies. For MSIs we instead need to warp between the input views and spherical shells. To this end, we implemented a custom Tensorflow operation that uses a commercial ray tracer [Wald et al. 2014] to compute warp fields between the MSI textures and input views. This operation also computes a validity mask to model visibility events, e.g. rays that miss the MSI shells. Note that since the MSI geometry is fixed, the model does not need to back propagate gradients through the ray tracer. We also added a simple sparsity loss when training to discourage the solver from producing dense volumetric data. It instead zeros out unseen parts of the MSI space. This produces fewer regions with positive alpha transparency to better aid layer reduction and image compression later in our pipeline. Specifically, for each position  $p$  in the MSI we collect all depth-wise alpha values in a vector  $\mathbf{M}_p$ . We then introduce an additional loss term,

$$\mathcal{L}_{sparse} = \sum_p \left| \frac{\mathbf{M}_p}{|\mathbf{M}_p|_2} \right|_1.$$

This loss prefers sparsity along depth layers but does not encourage the network to reduce the overall magnitude of the alpha values along a z column.

Finally, we adapted the training procedure for our rig geometry. We collected 130 training scenes, each consisting of captures from five shifted rig positions ( $\sim 5$  per scene each translated  $\sim 10$ -50cm

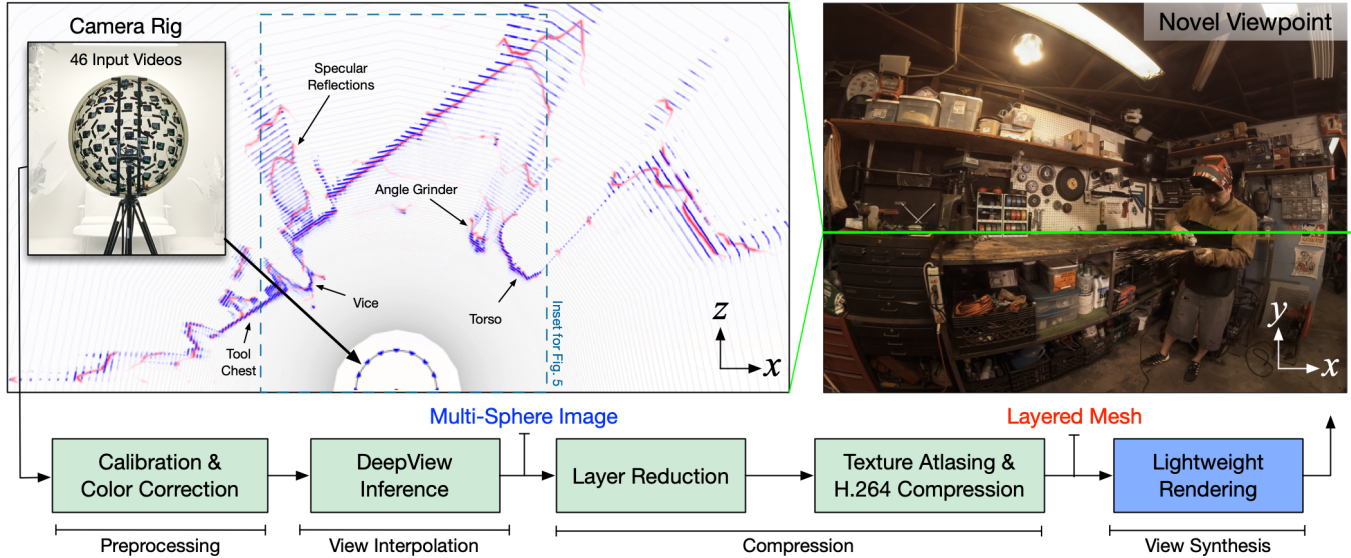


Fig. 4. Our Light Field Video System. Images captured by our camera rig are calibrated, color corrected, and then fed into a trained DeepView network to produce multi-sphere images (MSIs). Surfaces in the scene to the right are spread piecewise across many MSI layers (in gray), shown here in a top-down XZ slice. The alpha value for each MSI texel is shown in blue. Then, layer reduction consolidates this sparse content to a fixed number of Layered Meshes. The alpha channel for the LM texture maps is shown in red, and can be seen to closely “fit” to regions of the MSI with high alpha density. Subsequent texture atlasing and video compression results in a compact representation that can be readily played back on consumer graphics hardware.

relative to the initial position). We then calibrated and posed all rig positions within a scene together, again exploiting the nominal rig layout so that camera poses were consistent between positions. During training, one of the rig positions was used as input images while the rest provide nearby target views inside the interpolation volume for the network to learn to synthesize. For all the results shown, we trained with hexagonally-shaped clusters of seven adjacent cameras as inputs, which we found generalized well when running inference with all 46 cameras.

### 4.3 Converting MSIs to Layered Meshes

The MSIs we generate generally yield high quality view reconstructions throughout the interpolation volume (see Figure 8). However, the amount of data produced – over 100 high-resolution RGBA images per frame of video – would be challenging to stream as well as challenging to render on mobile graphics hardware with a limited amount of polygon fill per frame. Fortunately, our MSI volumes are relatively sparse, with positive alpha values falling mostly near the actual scene surfaces, and on virtual surfaces representing specular reflections. As a result, we can fit a set of polygonal Layered Meshes (LMs) with RGBA textures to the MSI that largely reproduce its appearance with considerably less data. This is a critical step, because the LM data is much more readily compressible using the methods we describe in Section 4.4. The process of converting an MSI to an LM follows four basic steps:

**4.3.1 Subdividing the MSI.** We subdivide the MSI layers into discrete depth ranges which we call *Layer Groups*, each with an equal number of consecutive layers. Figure 5(a) shows these subdivisions

for the inset data in the Figure 4. This approach is by construction temporally coherent because the split positions remain static throughout the video<sup>2</sup>. Using regular subdivisions also limits the depth complexity of any given LM layer and provides a clear upper bound on the worst case rubber sheet artifacts that can be introduced by each LM layer (see Figure 6). Furthermore, segmenting non-overlapping depth layers in this manner makes it easy to guarantee depth ordering during rendering. We chose to segment MSIs into 16 LM layers, as this was the highest attainable quality within our performance budget – more layers increased rendering overhead and decreased atlasing efficiency.

**4.3.2 Computing Layer Depth maps.** In many cases, the MSI encodes continuous surfaces by alpha cross fading in depth. At the same time, it also uses saturated alpha values to represent strong discontinuities, and soft alpha values to encode subtle lighting effects and translucent surfaces. We found that the accumulation of alpha in over blending naturally handles all of these cases while remaining temporally coherent. Concretely, as shown in Figure 5(b), we collapse the MSI layers within each Layer Group into a single depth map using standard alpha compositing (Equation 1). However, instead of compositing RGB values we instead use the index of the layer, i.e. the layer disparity. We perform this over blend from the central viewpoint of the interpolation volume. This effectively computes the expected disparity within each Layer Group along rays that originate from the central viewpoint.

<sup>2</sup>In early experiments we found that adaptively changing the layer groups per frame based on scene content resulted in temporally unstable results. Keeping the same split boundaries throughout the entire video alleviates this problem. While this worked well in our system, an adaptive and temporally coherent layer splitting strategy is an interesting area for future research.

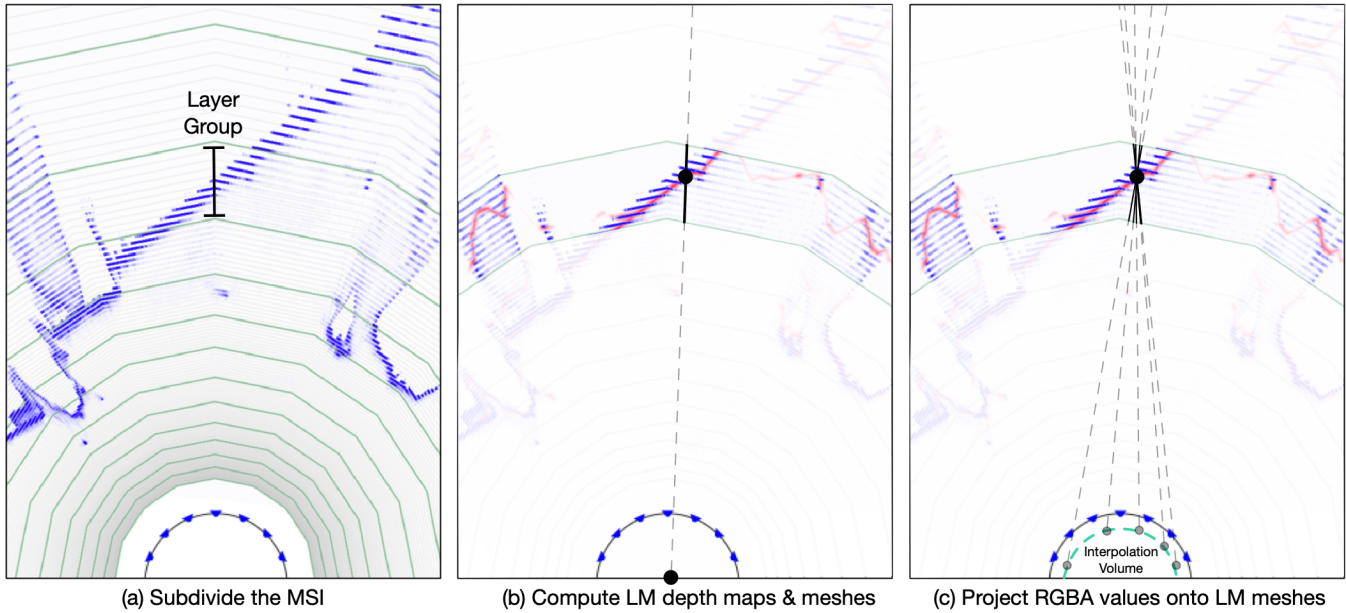


Fig. 5. Conversion from an MSI to an LM. (a) We first subdivide MSI layers (gray lines, alpha channel shown in blue) into Layer Groups (bounded by green lines). (b) Then, within each layer group we over blend MSI disparity and alpha to compute LM depth, and then convert each depth map to a triangle mesh. This is done along rays originating at central viewpoint. (c) Finally, within each Layer Group we project the RGBA values from the MSI layers onto the texture of the LM layer. Since no single viewpoint contains all the needed texture content for the LMs, we compute the Monte Carlo integral across rays originating at randomly sampled points on the surface of the interpolation volume. The alpha channel of the LM textures is shown in red in (b) and (c).

Fundamentally, representing the scene with depth maps introduces the possibility of rubber sheeting and stretched triangle artifacts whenever there is discontinuity between foreground and background layers. The benefit of MSIs is that they use alpha values instead of depth geometry to represent foreground edges, and we wish to preserve this as much as possible. The best we can do with a single layer of depth is to expand our depth maps around the edges

of foreground objects and rely instead on alpha values, inherited from the MSI, to cleanly represent discontinuities. In practice, we perform two alpha compositing passes to generate the Layer Group depth maps. The first pass initializes the depth estimate by compositing on top of a constant depth layer, which is the furthest depth in the layer group. The second pass extends the size of foreground objects by compositing the same depth map on top of a heavily dilated version of the first pass. This second pass helps to ensure that when a foreground edge and background are in the same mesh layer, we fall back to using the depth to represent the edge instead of alpha. This also has the benefit of reducing any depth aliasing that may be present in the MSI since the depth map construction averages together depth values from multiple MSI layers.

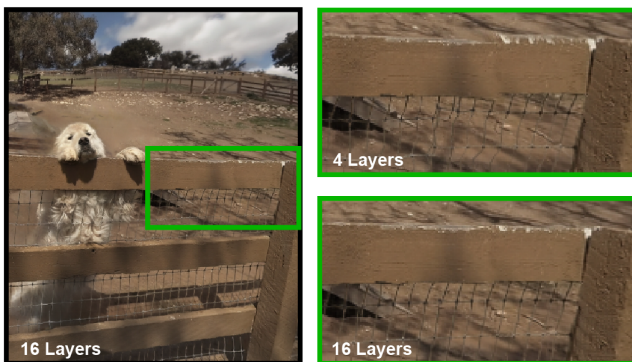


Fig. 6. Rubber sheet artifacts appear when using too few Layer Groups. Using just 4 LM layers (right top), the fence gets stretched to connect to the background and the chain links get warped. Using 16 layers (right bottom), the subdivisions are small enough to sufficiently separate foreground and background, and the rubber sheet artifacts are no longer visible.

**4.3.3 Computing Mesh Layers.** In order to efficiently render the LMs, we convert them from depth maps to low polygon meshes. For video playback, the mesh for each layer needs to be compact and also maintain a reasonable degree of temporal coherence across frames. To meet the first requirement, we need to simplify the mesh relative to the original depth map, but the second requirement discourages the use of complex mesh simplification algorithms that don't guarantee temporal coherence. Since most of the geometric detail and depth complexity in the scene is represented by the alpha channel, we can use the simple approach of generating a quad for each  $N \times N$  block of pixels in the depth map. We assign the depth of each quad vertex to the value of its depth map at its location. Naturally, there is a breaking point where, when  $N$  is too large, the mesh layers may separate in places where they should be connected.

This is especially noticeable along straight edges in the scene. We found  $N = 8$  to give the best trade-off between quality and the final triangle count for our data. This approach is temporally coherent because mesh connectivity remains consistent in every frame and each vertex only moves along the sphere’s radial direction.

**4.3.4 Projecting RGBA values onto Mesh Layers.** The final step in the layer reduction process is to project the MSI textures in each Layer Group onto the corresponding mesh layer. It is trivial to reproduce the appearance of a single viewpoint by over compositing the MSI textures into that viewpoint, and then projecting that result onto the corresponding LM mesh. However, we would prefer to generate the LM texture from multiple viewpoints, as more than one viewpoint is needed to capture the rich data in the MSI layers. Of course, for objects with high depth complexity a single LM texture cannot reproduce every possible ray encoded in the MSI layer group, so we are forced to compromise. We chose to average the rays across many randomly selected viewpoints to come up with the single color and alpha value at each point on the LM layer surface.

The approach we use is inspired by the prefiltering introduced in [Overbeck et al. 2018]. At each texel in every LM layer we perform a 2D integral along all potential lines of sight originating in the interpolation volume. As Figure 5(c) shows, we approximate the integral using Monte Carlo ray tracing, where each ray computes the alpha over-blend along the ray through the MSI Layer Group. Since our alpha values represent a density, we compute the alpha integral in log space and weight each RGB +  $\log(\alpha)$  sample by  $\log(\alpha)$ . This causes solid surfaces to heavily outweigh transparent surfaces in the integral. This approach guarantees that all view directions in the Layered Mesh receive a contribution from the MSI and that holes are filled.

To compute the RGB color  $c_t$  and the log-space alpha component  $\log(\alpha_t)$  for each texel  $t$  on a layered mesh, we compute Monte Carlo estimates for the integrals

$$\log(\alpha_t) = \lambda^{-1} \int_{V(t)} w(\mathbf{r}) \log(\alpha_r)^2 d\mathbf{r}, \quad (2)$$

and

$$c_t = \lambda^{-1} \int_{V(t)} w(\mathbf{r}) \log(\alpha_r) c_r d\mathbf{r}. \quad (3)$$

Here,  $V(t)$  is the set of all rays that originate in the interpolation volume and pass through the texel  $t$ ,

$$\lambda = \int_{V(t)} w(\mathbf{r}) \log(\alpha_r) d\mathbf{r} \quad (4)$$

is a normalizing constant, and we obtain the final alpha value as

$$\alpha_t = e^{\log(\alpha_t)}. \quad (5)$$

The functions  $\alpha_r$  and  $c_r$  are evaluated only through the MSI layers in the current Layer Group using Equation 1. In some cases, prefiltering can add noticeable blur seen in disoccluded areas from novel viewpoints. To counteract this, we use a function  $w(\mathbf{r})$  that more heavily weights viewing rays closer to the center view. This helps maintain the sharpness of the original MSI. We use a Gaussian with its peak at the center of the interpolation volume.

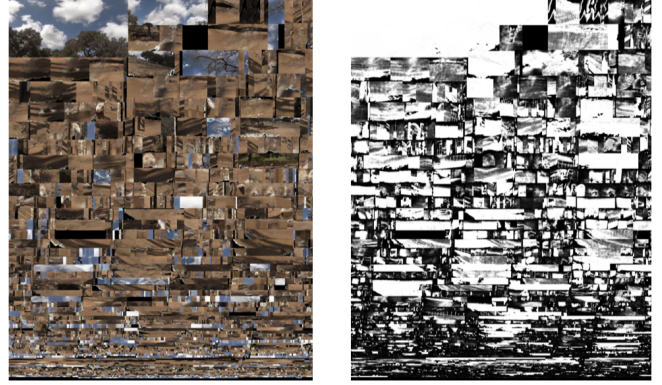


Fig. 7. An example texture atlas from the Dog scene in Figure 8.

*Discussion.* As shown in Section 5, there is some loss of view interpolation quality in the conversion from MSI’s to LM’s. For light field still images, MSI’s may be the best representation. However, for generating a compact light field video that can stream and render in real-time, the conversion to the Layered Mesh representation is crucial. To bring the quality of LM’s fully up to the level of MSI’s, we would ideally use an end-to-end machine learning method that minimizes a perceptual error of the final view synthesis. However, the semantic non-overlapping structure of the layered meshes makes this a considerably more difficult inference process to parameterize. We leave end-to-end optimization of Layered Meshes as future work.

#### 4.4 Video Compression and Rendering

**4.4.1 Texture Atlasing.** We use a texture atlasing approach to consolidate the RGBA texture data from the Layered Mesh sequences. This allows us to better leverage standard 2D video codecs. We create the atlas over a group of video frames (i.e. a GOP, or *Group of Pictures*) to allow for efficient video and mesh compression. We first sparsify the meshes by computing the set of texels that are close to transparent over all frames in each GOP, removing any triangle that don’t intersect with at least one opaque texel. This reduces the size of the mesh and the total texels as can be seen in Table 1. We then divide the resulting meshes into near-rectangular sections and pack them into a flat texture space using the Skyline Bottom-Left algorithm described in [Jylänki 2010]. The atlas texture is then copied piece by piece from the relevant layered mesh images. The occupancy of the resulting texture area is given in Table 1.

Table 1. The number of mesh triangles and vertices before and after the removal of transparent triangles, and percentage of pixels covered by mesh triangles for the scenes in Figure 8.

Scene	before removal		after removal		Occupancy
	# tris	# verts	# tris	# verts	
Dog	605k	1.20M	151k	223k	84.7%
Flames	605k	1.20M	166k	231k	81.8%
Car	605k	1.20M	92k	139k	82.0%



Table 2. The RGB+A side-by-side atlas texture size in pixels and bit rates for compressed image and geometry video data for the scenes in Figure 8. The images are compressed using H.265 with a CRF value of 14, and meshes are compressed using the Draco geometry compression library.

Scene	Atlas Size	Bitrates		
		Texture	Mesh	Total
Dog	3240x5760	184Mb/s	81.2Mb/s	266Mb/s
Flames	3240x5760	249Mb/s	72.3Mb/s	322Mb/s
Car	3240x5760	72.7Mb/s	51.1Mb/s	124Mb/s

**4.4.2 Compression.** To compress the mesh geometry, we note that the vertex positions that remain after atlasing only vary along their radial direction across frames within a GOP. We leverage this redundancy by storing the mesh connectivity, normalized vertex positions, and atlas texture coordinates only once per GOP. Separately, we store the per-frame scalar inverse radial displacements of each vertex. This remaining mesh data is further compressed using the open-source Draco library [Draco 2019].

We compress the stream of texture-atlased images using the standard H.265 video encoding algorithm with a CRF value of 14. Maintaining the same atlas layout per GOP generates a more compressible atlas stream. Alpha channels aren't widely supported by the most common video codecs, so we concatenate the color and alpha channels side-by-side before compressing, as shown in Fig. 7. While this doubles the width of the atlas, the lack of chroma variance in the alpha channel makes it more efficient to compress. We found that encoding at a high quality with a CRF of 14 is important because compression artifacts in the alpha channel show up as shimmering around moving objects. The sizes of the final compressed data for the scenes in Figure 8 are shown in Table 2. To summarize, across a range of scenes, we achieve data rates of 124 Mb/s – 322 Mb/s.

**4.4.3 Rendering.** Our data representation is extremely simple and light-weight to decode and render. We decode both the mesh and image data and stream it to the GPU. The GOP meshes are reassembled by dividing the per-frame inverse radial displacements from the normalized vertex positions, and the texture atlas video stream is decoded using standard video decoder hardware. Rendering requires two texture taps: one for RGB and one for alpha. The LM's are rendered in back-to-front order, and we use GPU hardware blending operations to composite the resulting pixel values using a pre-multiplied over blend.

## 5 RESULTS

In this section we present evaluations and comparisons to show the visual quality of our light field video pipeline.

Figure 8 shows three light field video frames processed into MSI's, each of which is shown as a cross-fusible stereogram with a wide baseline to demonstrate the substantial viewing volume. The depth map visualizations seen at the right are made from the MSI's rendered from the center perspective. Note that these are composite images of the depths of all the individual spherical layers; we are not simply projecting textures onto a single depth map. The "Dog" scene rendered with a 30 cm baseline demonstrates the system's

ability to resolve thin structures in the wire fence mesh and near-field objects such as the dog's nose just 1 m from the center of the camera array. The "Flames" example rendered with a 60 cm baseline shows a successful result for a high dynamic range scene with a partially-translucent flame volume, represented successfully using the layer transparency channels. The person's shape is nicely separated from the background with crisp anti-aliased edges. The further reaches of the dark, textureless sky are reconstructed as having opacity on layers which are much closer than they should be; fortunately this error is difficult to notice within the viewing volume of the re-rendered imagery. The "Car" example rendered with a 60 cm baseline demonstrates challenging thin structures in the bare tree branches as well as nicely reproduced specular reflections on the car's hood and windshield. The network reproduces the specular behavior by making the car's surface partially translucent and placing the specular reflections on layers deeper behind the surface. The composite depth map visualization shows evidence of this in the form of apparent concavities in the car's hood and windows. These depth colors result from averaging together the depths of several layers with partial opacity.

Our accompanying papers video shows animated viewpoints through these (and other) scenes as the light field video is played back and occasionally paused. Our supplemental material includes an interactive viewer that runs in a web browser to inspect several other light field video frames. The local web viewer application allows each frame to be viewed from different viewpoints and layer-by-layer as a Multi-Sphere Image, a Layered Mesh, and a texture-atlased Layered Mesh.

**Processing Time.** We rely on cloud processing infrastructure to process videos in a reasonable amount of time. Thanks to heavy parallelization over hundreds to thousands of worker machines, we are able to fully process all videos, regardless of length, in less than a day. One sample run of our pipeline on 150 video frames took a total 4,271 CPU hours, or about 28.5 CPU hours per frame. This time is overwhelmingly dominated by the inference process to generate the MSI (see Subsection 4.2), which took about 25.4 CPU hours per frame. In actual wall-clock time, the full pipeline took about 17 hours. This run, and all of the results in the paper, generate high resolution MSIs at 1800×1350 resolution with 160 layers. Note that our system, including MSI inference, runs entirely on CPUs because we have easier access to CPUs than GPUs. However, there are optimization opportunities available if MSI inference were to be run on a modern GPU.

### 5.1 Comparing MSIs, Layered Meshes, and Texture-Atlased Layered Meshes

To evaluate our system we use held-out scenes from our training data shot from one rig position, and measure how well our system reproduces an unseen view from another nearby rig position. We quantify the difference between the synthesized and novel views across various stages using the LPIPS [Zhang et al. 2018] and SSIM [Wang et al. 2004] metrics. As shown in Table 3, there is a progressive (though acceptable) loss in quality as we progress from the MSI to the atlased Layered Mesh representation. Note that the SSIM scores in Table 3 are not comparable across datasets, due to

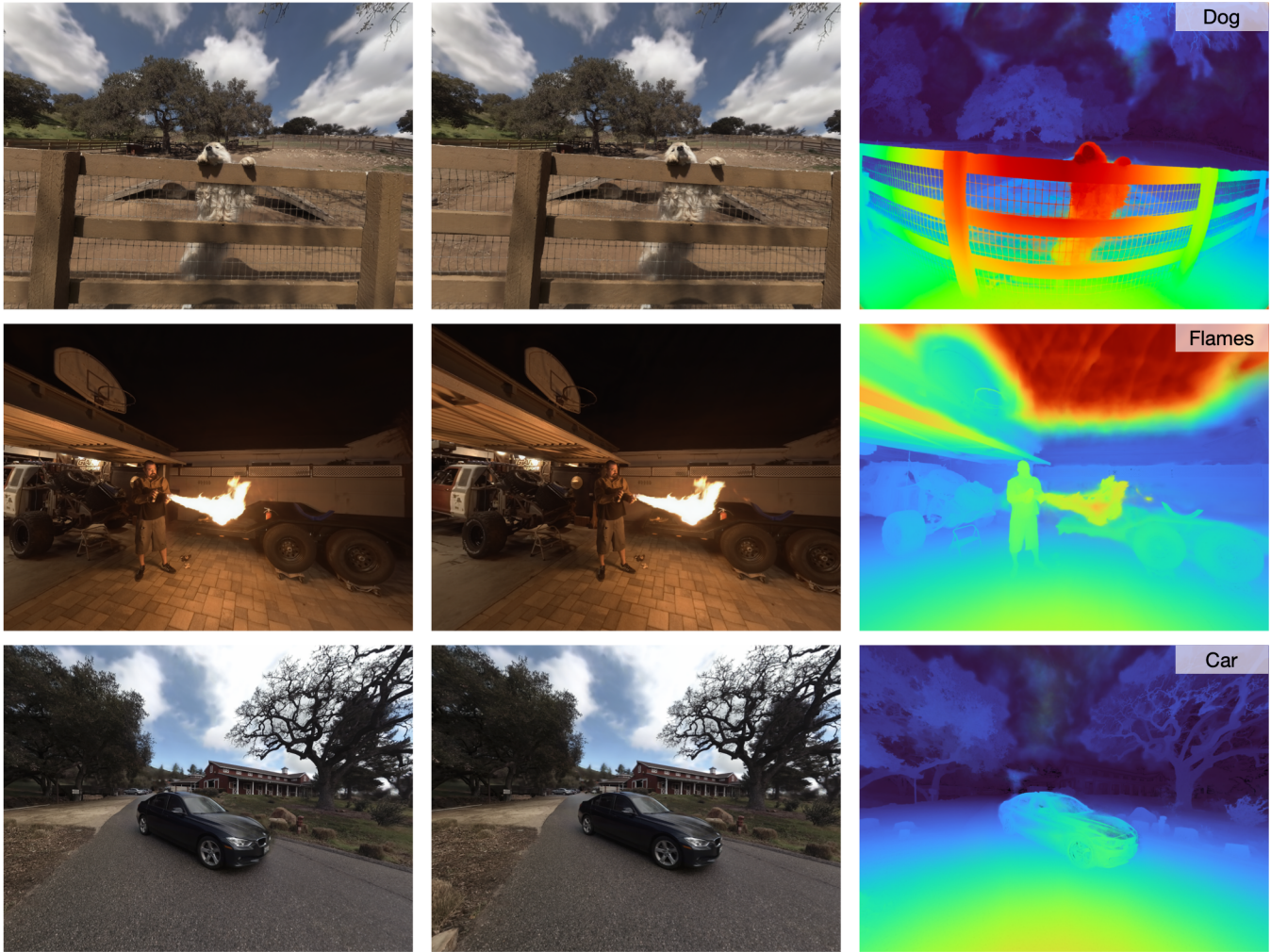


Fig. 8. Cross-fusible stereograms of three light field video frames processed by our complete pipeline, accompanied by colored depth visualizations produced from the MSI as in Section 4.3. The scenes are re-rendered with  $\pm 15$  cm,  $\pm 30$  cm, and  $\pm 30$  cm stereo baselines.

effects such as input image resolution, sharpness, visible parallax, scene materials, and the geometry of the scene. Bear in mind the difficult nature of our evaluation dataset when assessing these SSIM scores. Our rig has a large 18 cm camera spacing and our evaluation scenes contain difficult-to-reconstruct view-dependent materials and objects close to the camera rig.

## 5.2 Comparison to [Overbeck et al. 2018]

Figure 9 compares our approach to that of [Overbeck et al. 2018] on several light field stills from the SteamVR application *Welcome to Light Fields* (WLF). WLF captured high quality panoramic light field stills using a rotating camera gantry, with a relatively dense 1,000 images over the full sphere per scene. Overbeck et al. [2018] follow a more traditional light field rendering approach, projecting each of the images onto depth geometry and blending between them in real time. Their results are high-quality and immersive, but

since our system needs to record all views simultaneously, we are restricted to using fewer images from a much sparser camera array. To compare the pipelines more directly, we extracted the 46 image positions from the WLF light fields that most closely match the viewing positions in our hemispherical camera rig (Sec. 4.1).

In Figure 9, we compare our atlased LM and MSI representations with [Overbeck et al. 2018] with the 46 camera positions from our rig (WLF 46) and against their results with all of the images. We use two scenes from their paper: *Gamble House* on top and the *Living Room* on the bottom. When using only 46 images, [Overbeck et al. 2018] struggles to accurately model and render thin structures, such as the lamp’s arm in the *Gamble House* (the blue highlighted region) and on the tripod legs in the *Living Room* (the green highlighted region). Both our atlased LM and MSI representations faithfully capture these fine details. WLF with 46 images also struggles with some reflections. In these cases, their single-layer depth maps have



Fig. 9. Our MSIs and Atlased LMs compared to *Welcome to Light Fields* (WLF) [Overbeck et al. 2018] using a sparse 46-image subset of their views to match our rig (WLF 46), and also against their full-quality renderings using the full front hemisphere of their images (WLF 328 and WLF 595). With only 46 cameras, WLF 46 struggles to capture thin structures and some reflections. Both our atlased LMs and MSIs capture these effects, despite atlased LMs requiring 26 $\times$  less storage than WLF 328 and 47 $\times$  less than WLF 595.

to choose whether to follow the surface in the reflection or the reflective surface itself. For the glossy reflection under the vase in the Gamble House (the yellow highlighted region), the depth maps for WLF 46 alternate from representing the cabinet’s surface and the reflections within it, causing a distortion in the glossy reflection. Similarly, in the left of the mirror in the Living Room, the doorway appears ghosted in the reflection. In all our examples, our Atlased LM result is very close to the MSI result but appears slightly softer in some areas. This is where the LM geometry differs from the underlying MSI and the filter described in Section 4.3 smooths over the differences.

When [Overbeck et al. 2018] uses all 595 images in the Living Room and all 328 images from the Gamble house, its rendered quality is close to ours, however our data is compacted into a 16 layer LM packed into a single 3840 $\times$ 4320 pixel atlas for Gamble House and 2880 $\times$ 3240 pixel atlas for Living Room, whereas their system uses 328 images for Gamble House and 595 images for Living Room, all at 1280 $\times$ 1024 pixel resolution. That’s 47 times and 26 times the number of pixels to encode, respectively, making it far less feasible for video.

### 5.3 Comparison with [Pozo et al. 2019]

We also apply our method to an alternative camera rig geometry with a sparser sampling of cameras. We downloaded the publicly

available dataset from the Facebook Manifold camera [Pozo et al. 2019] and processed it with our pipeline. This data was captured with 16 cameras distributed over a 48 cm diameter sphere, which is significantly fewer than the 46 cameras over a 90 cm hemisphere we use for training and acquisition. In exchange for camera density, each camera in the Manifold rig captures more pixels per degree, a greater field of view, higher dynamic range, and provides a better signal-to-noise ratio.

As Figure 10 and the supplemental video both show, our system extends well to this data. In the crops, we show output views placed at different distances from the rig center, comparing our approach with the publicly available implementation of the Manifold camera renderer<sup>3</sup>. Our approach gives better view interpolation results in regions that are difficult to recover with traditional multi-view stereo and image-based rendering approaches. Our method achieves smooth anti-aliased occlusion boundaries and well-reproduced thin structures. This difference is particularly pronounced 20 cm away from the rig center, near the edge of the interpolation volume derived in Section 3.1. This available Manifold scene features relatively distant and diffuse geometry, and we believe that our performance gains could be more pronounced for a scene with near-field objects and specular reflections.

<sup>3</sup>[https://github.com/facebook/facebook360\\_dep](https://github.com/facebook/facebook360_dep)


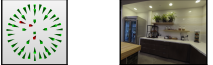

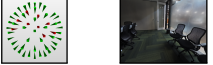
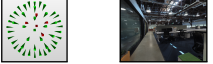
Scene	MSI		LM		Atlased LM	
	LPIPS	SSIM	LPIPS	SSIM	LPIPS	SSIM
	0.0552	0.9649	0.0788	0.9473	0.0818	0.9428
	0.0631	0.9638	0.0911	0.9403	0.0915	0.9406
	0.0804	0.9550	0.1140	0.9293	0.1145	0.9295
	0.0974	0.9532	0.1265	0.9389	0.1327	0.9390
	0.0687	0.9607	0.1192	0.9251	0.1199	0.9252

Table 3. Comparison of rendered vs actual images on different evaluation scenes at various stages of our process from MSI to Layered Mesh (LM) to Atlased Layered Mesh (Atlased LM). The input views, shown in green, are processed by our system and used to synthesize the held out views shown in red. We show the LPIPS [Zhang et al. 2018] (lower is better) and SSIM [Wang et al. 2004] (higher is better) scores.

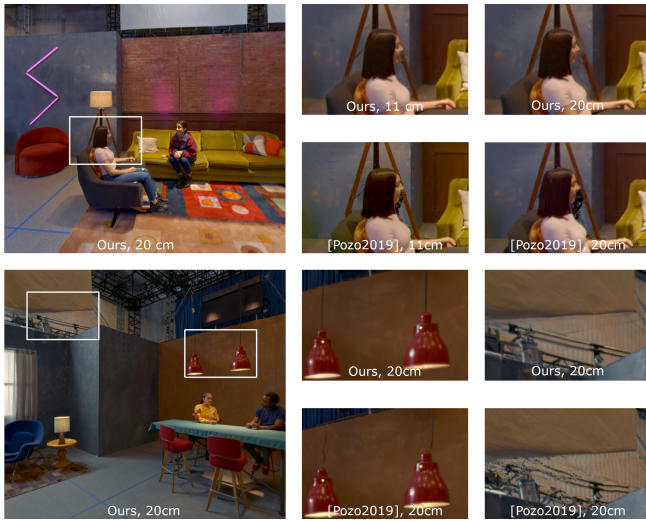


Fig. 10. Comparing atlased LMs with [Poza et al. 2019] on data captured with their camera rig. Each image is annotated with the distance from the rig center. The crops emphasize details that are challenging for traditional image based rendering methods, such as thin structures and occlusion boundaries.

In terms of storage representation, the Manifold scene stores high resolution geometry and texture for each frame and camera in the rig, requiring more than 9 GB of storage for this 50-frame clip; this would likely require a powerful desktop PC with multiple high-bandwidth solid state disks for smooth playback. In contrast, our compressed representation is much smaller (124 MB) and can be rendered in real-time on a consumer gaming laptop.

## 6 LIMITATIONS

Our system faithfully renders high quality views with dynamic and complex visual effects, such as fire (Figs. 1 and 8), sparks (Fig. 4 right), and reflections and fine geometry (see Figs. 9 and 8). However, we inherit the limitations of the underlying MPI representation upon which we build our work. There are some visual phenomenon which are not accurately represented by MPIs or MSIs, such as curved reflectors. Nonetheless, even in these difficult cases our optimization framework tends to subtly blur, or average over, problematic areas. See, for example the glossy reflection in the yellow highlighted insets in Figure 9 where our results tend to be smoother towards the edge of the counter top than the high-resolution WLF 328 results.

Motion-related blur is noticeable in low-light scenes such as the cave (see supplemental video). In low light conditions our cameras choose a 360 degree shutter to maximize the exposure time, which appears blurrier than expected even in the source videos. We believe that some temporal blurring also results from our inexpensive rig's not-quite-perfect sync (within 4 ms) and when the auto-exposing cameras choose different exposure times. MSIs reconstructed from such input images contain multi-camera motion blur that is the average of the individual motion blurs. While MSIs and LMs can represent consistent motion blur well, our network seems to translate heterogeneous motion blur into additional blurring.

Additionally, difficult dis-occluded regions sometimes appear blurry, for example around the person standing in the horse stall (see supplemental material). This results from the following approximations: First, due to GPU memory constraints we train with a local cluster of only 7 cameras, although we still perform inference with all 46 cameras. Thus, during training the network has a more limited ability to peek around objects and learn to distinguish foreground and background content. Second, as shown in the supplemental video and Table 3, there is a loss of quality when converting to LMs. Using the pre-filtering approach in Section 4.3.4, we make the loss in quality less noticeable by preferring blur in these regions.

Yet, in many difficult cases, e.g. in the Car scene (Fig. 8), we produce plausible results free of retinal rivalry. Also, sometimes in constant color regions, especially black, our network doesn't see enough detail to identify a smooth surface, and instead spreads the constant color across multiple layers of the MSI. Although these generally don't detract from the overall visual quality from inside of the viewing volume, they do reduce the compressibility of the MSI. A drawback of our training and processing pipelines are they require substantial cloud computing resources. It would not be practical to process a several-minute-long video on a single workstation using our current framework.

## 7 CONCLUSIONS AND FUTURE WORK

In this paper, we have presented the first end-to-end system for recording, processing, compressing, and rendering view-dependent immersive light field video at sufficiently low bandwidth for internet streaming. While our results are relatively free of visual artifacts, there are several avenues for future work to improve and augment the system. First, we could extend our rig to a full 360 degree field of view, which could be accomplished by building a copy of the rig and placing the two rigs back-to-back; our MSI inference already can accommodate fully panoramic spherical shells. Second,

we could improve the MSI-to-LM conversion to better adapt the mesh placement to where surfaces are in the scene, instead of allowing a single mesh surface per depth Layer Group. This would improve the appearance of object edges where the surface they are in front of might otherwise fall into the same depth Layer Group. Ideally, we would design and train a new version of the network to directly output Layered Meshes in an end-to-end system. Better mesh placement might allow the scene to be represented well with fewer meshes, which could facilitate streaming to mobile headsets such as the Oculus Quest and Magic Leap. Finally, it is interesting to consider how the inference network could be optimized to support streaming a live event, such as concert or sports match, to a distributed remote audience, with each person able to comfortably watch the scene from their individual point of view.

## ACKNOWLEDGMENTS

The authors would like to thank Shahram Izadi and the greater Augmented Perception team within Google Daydream for their generous support for this project. We thank Clay Bavor, Steve Seitz, and Matt Pharr for their foundational support of the light fields effort at Google. We are also grateful to Peter Denny for keeping us organized and to Libor Janicek, the Gonda family, Alexa Meade, Grace Hoyt, and the Google Artist in Residence program for helping us shoot engaging light field content, and to Brian Cabral and Brent Schnarr for supplying us with data from the Manifold camera system. Finally, thanks go to Graham Fyffe, Noah Snavely, and Richard Tucker for numerous illuminating conversations during the course of this research.

## REFERENCES

- 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 (Nov. 2016), 13 pages. <https://doi.org/10.1145/2980179.2980257>
- Vasileios Avramelos, Glenn Van Wallendael, and Peter Lambert. 2019. Overview of MV-HEVC prediction structures for light field video. In *Applications of Digital Image Processing XLII*, Andrew G. Tescher and Touradj Ebrahimi (Eds.), Vol. 11137. International Society for Optics and Photonics, SPIE, 382–390. <https://doi.org/10.1117/12.2529137>
- Chris Buehler, Michael Bosse, Leonard McMillan, Steven Gortler, and Michael Cohen. 2001. Unstructured Lumigraph Rendering. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01)*. Association for Computing Machinery, New York, NY, USA, 425–432. <https://doi.org/10.1145/383259.383309>
- Shenchang Eric Chen and Lance Williams. 1993. View Interpolation for Image Synthesis. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques (Anaheim, CA) (SIGGRAPH '93)*. Association for Computing Machinery, New York, NY, USA, 279–288. <https://doi.org/10.1145/166117.166153>
- Alvaro Collet, Ming Chuang, Pat Sweeney, Don Gillett, Dennis Evseev, David Calabrese, Hugues Hoppe, Adam Kirk, and Steve Sullivan. 2015. High-Quality Streamable Free-Viewpoint Video. *ACM Trans. Graph.* 34, 4, Article Article 69 (July 2015), 13 pages. <https://doi.org/10.1145/2766945>
- Abe Davis, Marc Levoy, and Fredo Durand. 2012. Unstructured Light Fields. *Comput. Graph. Forum* 31, 2pt1 (May 2012), 305–314. <https://doi.org/10.1111/j.1467-8659.2012.03009.x>
- Paul Debevec, Yizhou Yu, and George Borshukov. 1998. Efficient View-Dependent Image-Based Rendering with Projective Texture-Mapping. In *Rendering Techniques '98*, George Drettakis and Nelson Max (Eds.). Springer Vienna, Vienna, 105–116.
- Mingsong Dou, Sameh Khamis, Yury Degtyarev, Philip Davidson, Sean Ryan Fanello, Adarsh Kowdle, Sergio Orts Escolano, Christoph Rhemann, David Kim, Jonathan Taylor, and et al. 2016. Fusion4D: Real-Time Performance Capture of Challenging Scenes. *ACM Trans. Graph.* 35, 4, Article Article 114 (July 2016), 13 pages. <https://doi.org/10.1145/2897824.2925969>
- Draco. 2019. [https://github.com/google/draco/](https://github.com/google/draco)
- John Flynn, Michael Broxton, Paul Debevec, Matthew DuVall, Graham Fyffe, Ryan Overbeck, Noah Snavely, and Richard Tucker. 2019. DeepView: View Synthesis With Learned Gradient Descent. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2367–2376.
- John Flynn, Ivan Neulander, James Philbin, and Noah Snavely. 2016. DeepStereo: Learning to Predict New Views From the World's Imagery. In *CVPR*.
- Yasutaka Furukawa and Carlos Hernandez. 2015. Multi-View Stereo: A Tutorial. *Foundations and Trends in Computer Graphics and Vision* 9, 1-2 (2015), 1–148. <https://doi.org/10.1561/06000000052>
- 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)*. Association for Computing Machinery, New York, NY, USA, 43–54. <https://doi.org/10.1145/237170.237200>
- Richard Hartley and Andrew Zisserman. 2003. *Multiple View Geometry in Computer Vision* (2 ed.). Cambridge University Press, New York, NY, USA.
- Peter Hedman, Suhil Alisan, Richard Szeliski, and Johannes Kopf. 2017. Casual 3D Photography. *ACM Trans. Graph.* 36, 6, Article Article 234 (Nov. 2017), 15 pages. <https://doi.org/10.1145/3130800.3130828>
- Benno Heigl, Reinhard Koch, Marc Pollefeys, Joachim Denzler, and Luc J. Van Gool. 1999. Plenoptic Modeling and Rendering from Image Sequences Taken by Hand-Held Camera. In *Mustererkennung 1999, 21. DAGM-Symposium*. Springer-Verlag, Berlin, Heidelberg, 94–101.
- Insung Ihm, Sanghoon Park, and Rae Kyoung Lee. 1997. Rendering of spherical light fields. In *Proceedings The Fifth Pacific Conference on Computer Graphics and Applications*. 59–68. <https://doi.org/10.1109/PCCGA.1997.626172>
- Jukka Jylänki. 2010. A thousand ways to pack the bin – a practical approach to two-dimensional rectangle bin packing.
- Nima Khademi Kalantari, Ting-Chun Wang, and Ravi Ramamoorthi. 2016. Learning-based View Synthesis for Light Field Cameras. *ACM Trans. Graph.* 35, 6, Article 193 (2016), 10 pages. <https://doi.org/10.1145/2980179.2980251>
- Marc Levoy and Pat Hanrahan. 1996. Light Field Rendering. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '96)*. Association for Computing Machinery, New York, NY, USA, 31–42. <https://doi.org/10.1145/237170.237199>
- C. Lipski, C. Linz, K. Berger, A. Sellent, and M. Magnor. 2010. Virtual Video Camera: Image-Based Viewpoint Navigation Through Space and Time. *Computer Graphics Forum* 29, 8 (2010), 2555–2568.
- Ricardo Martin-Brualla, Rohit Pandey, Shuoran Yang, Pavel Pidrlynskyi, Jonathan Taylor, Julien Valentin, Sameh Khamis, Philip Davidson, Anastasia Tkach, Peter Lincoln, and et al. 2018. LookinGood: Enhancing Performance Capture with Real-Time Neural Re-Rendering. *ACM Trans. Graph.* 37, 6, Article Article 255 (Dec. 2018), 14 pages. <https://doi.org/10.1145/3272127.3275099>
- Leonard McMillan and Gary Bishop. 1995. Plenoptic Modeling: An Image-Based Rendering System. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '95)*. Association for Computing Machinery, New York, NY, USA, 39–46. <https://doi.org/10.1145/218380.218398>
- Ben Mildenhall, Pratul P. Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. 2019. Local Light Field Fusion: Practical View Synthesis with Prescriptive Sampling Guidelines. *ACM Trans. Graph.* 38, 4, Article Article 29 (July 2019), 14 pages. <https://doi.org/10.1145/3306346.3322980>
- Tim Milliron, Chrissy Szczupak, and Orin Green. 2017. Hallelujah: The World's First Lytro VR Experience. In *ACM SIGGRAPH 2017 VR Village* (Los Angeles, California) (SIGGRAPH '17). Association for Computing Machinery, New York, NY, USA, Article Article 7, 2 pages. <https://doi.org/10.1145/3089269.3089283>
- Ryan S. Overbeck, Daniel Erickson, Daniel Evangelakos, and Paul Debevec. 2018. Welcome to Light Fields. In *ACM SIGGRAPH 2018 Virtual, Augmented, and Mixed Reality* (Vancouver, British Columbia, Canada) (SIGGRAPH '18). Association for Computing Machinery, New York, NY, USA, Article Article 32, 1 pages. <https://doi.org/10.1145/3226552.3226557>
- Eric Penner and Li Zhang. 2017. Soft 3D Reconstruction for View Synthesis. *ACM Trans. Graph.* 36, 6 (Nov. 2017), 235:1–235:11.
- Albert Parra Pozo, Michael Toksvig, Terry Filiba Schragar, Joyce Hsu, Uday Mathur, Alexander Sorkine-Hornung, Rick Szeliski, and Brian Cabral. 2019. An Integrated 6DoF Video Camera and System Design. *ACM Trans. Graph.* 38, 6, Article Article 216 (Nov. 2019), 16 pages. <https://doi.org/10.1145/3355089.3355555>
- Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. 2017a. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *CVPR*.
- Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J. Guibas. 2017b. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In *NIPS*.
- 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)*. Association for Computing Machinery, New York, NY, USA, 231–242. <https://doi.org/10.1145/280814.280882>
- Heung-Yeung Shum, Shing-Chow Chan, and Sing Bing Kang. 2007. *Image-Based Rendering* (1 ed.). Springer.
- Heung-Yeung Shum and Li-Wei He. 1999. Rendering with Concentric Mosaics. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '99)*. ACM Press/Addison-Wesley Publishing Co., USA, 299–306.

- <https://doi.org/10.1145/311535.311573>  
 Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Niessner, Gordon Wetzstein, and Michael Zollhofer. 2019. DeepVoxels: Learning Persistent 3D Feature Embeddings. In *Proc. CVPR*.
- Pratul P. Srinivasan, Richard Tucker, Jonathan T. Barron, Ravi Ramamoorthi, Ren Ng, and Noah Snavely. 2019. Pushing the Boundaries of View Extrapolation with Multiplane Images. *CVPR* (2019).
- Justus Thies, Michael Zollhofer, and Matthias Nieundefinedner. 2019. Deferred Neural Rendering: Image Synthesis Using Neural Textures. *ACM Trans. Graph.* 38, 4, Article Article 66 (July 2019), 12 pages. <https://doi.org/10.1145/3306346.3323035>
- Ingo Wald, Sven Woop, Carsten Benthin, Gregory S. Johnson, and Manfred Ernst. 2014. Embree: A Kernel Framework for Efficient CPU Ray Tracing. *ACM Trans. Graph.* 33, 4, Article Article 143 (July 2014), 8 pages. <https://doi.org/10.1145/2601097.2601199>
- Ting-Chun Wang, Jun-Yan Zhu, Nima Khademi Kalantari, Alexei A. Efros, and Ravi Ramamoorthi. 2017. Light Field Video Capture Using a Learning-Based Hybrid Imaging System. *ACM Trans. Graph.* 36, 4, Article Article 133 (July 2017), 13 pages. <https://doi.org/10.1145/3072959.3073614>
- 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.
- Bennett Wilburn, Neel Joshi, Vaibhav Vaish, Eino-Ville Talvala, Emilio Antunez, Adam Barth, Andrew Adams, Mark Horowitz, and Marc Levoy. 2005. High Performance Imaging Using Large Camera Arrays. *ACM Trans. Graph.* 24, 3 (July 2005), 765–776. <https://doi.org/10.1145/1073204.1073259>
- Bennett S. Wilburn, Michal Smulski, Hsiao-Heng Kelin Lee, and Mark A. Horowitz. 2001. Light field video camera. In *Media Processors 2002*, Sethuraman Panchanathan, V. Michael Bove Jr, and Subramania I. Sudharsanan (Eds.), Vol. 4674. International Society for Optics and Photonics, SPIE, 29 – 36. <https://doi.org/10.1117/12.451074>
- Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. 2018. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In *CVPR*.
- Tinghui Zhou, Richard Tucker, John Flynn, Graham Fyffe, and Noah Snavely. 2018. Stereo Magnification: Learning View Synthesis Using Multiplane Images. *ACM Trans. Graph.* 37, 4, Article Article 65 (July 2018), 12 pages. <https://doi.org/10.1145/3197517.3201323>
- C. Lawrence Zitnick, Sing Bing Kang, Matthew Uyttendaele, Simon Winder, and Richard Szeliski. 2004. High-Quality Video View Interpolation Using a Layered Representation. *ACM Trans. Graph.* 23, 3 (Aug. 2004), 600–608. <https://doi.org/10.1145/1015706.1015766>

## A COMPUTING INTER-LAYER SPACING FOR MULTI-SPHERE IMAGES

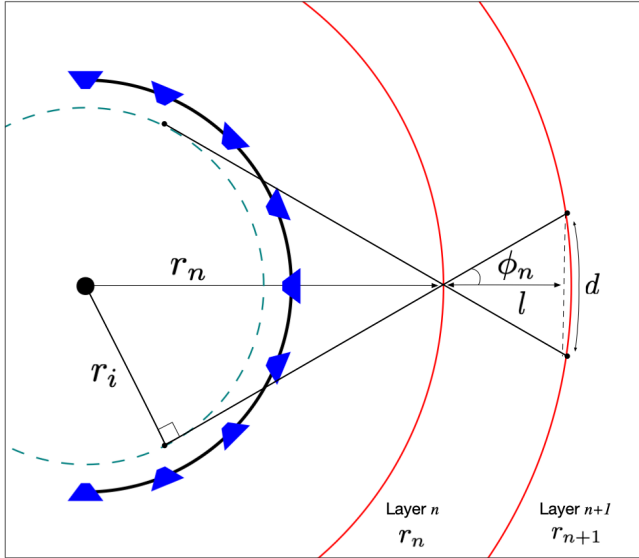


Fig. 11. Derived layer spacing bounds for MSIs.

In order to render views that do not alias or “tear apart” as the viewer moves from side to side in the interpolation volume, the

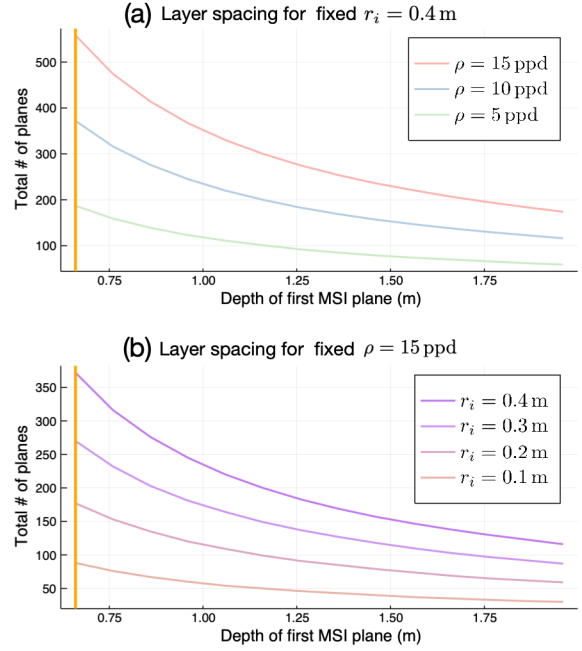


Fig. 12. Comparison of layer spacing bounds for MSIs.

maximum disparity seen by the viewer between any two adjacent layers must be  $\leq 1$  pixel. Here our goal is to determine the correct MSI layer spacing that achieves this inter-layer disparity (ILD) condition. Our analysis is similar to conditions for the spacing of MPI planes proposed in [Flynn et al. 2019; Zhou et al. 2018] and further refined using Fourier analysis in [Mildenhall et al. 2019].

Consider the geometric construction in Figure 11. Two adjacent spherical MSI layers are shown with radii  $r_n$  and  $r_{n+1}$ . In order to render views without aliasing, the ILD condition must hold for any ray that passes through the interpolation volume whose radius is  $r_i$ . It suffices to concern ourselves with the rays that pass through two adjacent layers at the steepest possible angle, as these are most likely to violate the condition stated above. Two such rays are shown, although due to the radial symmetry of the rig any pair of rays tangent to and on opposite sides of the interpolation volume could be chosen for this analysis. We see that the steepest ray angles at layer  $n$  that intersects with the interpolation volume has  $\phi_n = \pm \sin^{-1}(r_i/r_n)$ . If each spherical layer has a texture with uniform angular sampling  $\rho$  pixels per radian with respect to the center of the rig, each layer texture will have  $2\pi\rho$  pixels around its full circumference. The ILD condition therefore requires that the arc length  $d$  should be less than or equal to the spacing between texture pixels at layer  $n+1$ ; i.e.,  $d = r_{n+1}/\rho$ . For any reasonable high texture resolution,  $d \ll r_{n+1}$ , so by the small angle approximation we can treat the arc length  $d$  as being equal to the chord length (dotted line), and the distance  $l \approx r_{n+1} - r_n$ . After some algebraic manipulation, we arrive at the follow expression for the incremental layer spacing

for a multi-sphere image:

$$r_{n+1} = r_n \left( \frac{\tan(\phi_n)}{\tan(\phi_n) - 1/(2\rho)} \right) \quad (6)$$

From Equation 6 we see that MSI layer spacing depends on only two variables: (1) the radius  $r_i$  of the interpolation volume, and (2) the angular sampling rate  $\rho$  of the MSI layer textures. Starting at “near” shell radius  $r_0$  and iterate using Equation 6 until a “far” shell radius of  $r_{\max} = \infty$  is reached, we can compute the total number of shells required to satisfy the ILD condition. Figure 11 shows how many planes are required as a function of  $r_0$ , for different choice of  $r_i$  and  $\rho$ . We see that doubling angular sampling rate or doubling the size of the interpolation volume leads to a doubling of the number of depth planes required. This generally follows the scaling properties of MPIs that are described in [Mildenhall et al. 2019]. However, one advantage of using MSIs is that they have no such field of view dependence since they naturally provide equi-angular sampling in all viewing directions.

## B SPHERICAL LIGHT FIELD GEOMETRY OF THE MSI

Here we show how spherical light field geometry [Insung Ihm et al. 1997; Shum and He 1999] can be used to understand how to properly construct multi-spherical images that match the near object distance and interpolation volume size of our capture geometry. Figure 13(a) shows a spherical light field parameterization where each ray is indexed via a pair of coordinates  $(\theta, \phi)$ . The azimuth angle  $\theta$  is measured relative to the center of the interpolation volume, and it defines *the position* where a ray intersects a sphere at radius  $r_{rs}$ . Here, we also measure *the angle*  $\phi$  of an incoming ray relative to the normal vector. The inset in Figure 13(a) shows a  $(\theta, \phi)$  ray space diagram illustrating the ray “bundles” sampled by nine different cameras. The shape of the curves changes depending on the radius  $r_{rs}$  of the sphere we are projecting rays onto. When  $r_{rs} = r_{rig}$  the diagram contains vertical lines (black) corresponding to the fan of rays collected at each discrete camera position on the rig. However, if we plot those same ray bundles for  $r_{rs} = r_i$ , the lines elongate into s-curves (teal) that cover a full 180 degrees in  $\phi$ . This agrees with our earlier definition of the interpolation volume as the region with rays sampling all outward-facing viewing directions. Finally, if we choose the closest object distance  $r_{rs} = r_c$  (orange), we see camera rays curve in the other direction and flatten in the  $\phi$  dimension. We can see at this distance there is overlap of two cameras across the full range of azimuth positions since each coordinate on the azimuth axis ( $\theta$ ) contains two curved lines.

Figure 13(b) shows the ray space coverage of the MSI with a truncated 225 degree FOV similar what we use in our real system. Several color coded viewpoints are shown, each representing a potential novel view. The inset in this drawing is a ray space diagram parameterized at the typical distance where we place the first MSI sphere in our real system; i.e.,  $r_{rs} = 0.8$  m. The red parallelogram contains the valid rays for the MSI that both (1) satisfy the inter-layer disparity condition from Appendix A, and (2) pass through all the MSI layers. The MSI in this example was constructed to have a sufficient number of layers, following the analysis in Appendix A, thereby perfectly matching its interpolation volume to that of our rig cameras (dark blue curves). Rays needed to render the novel viewpoints are shown

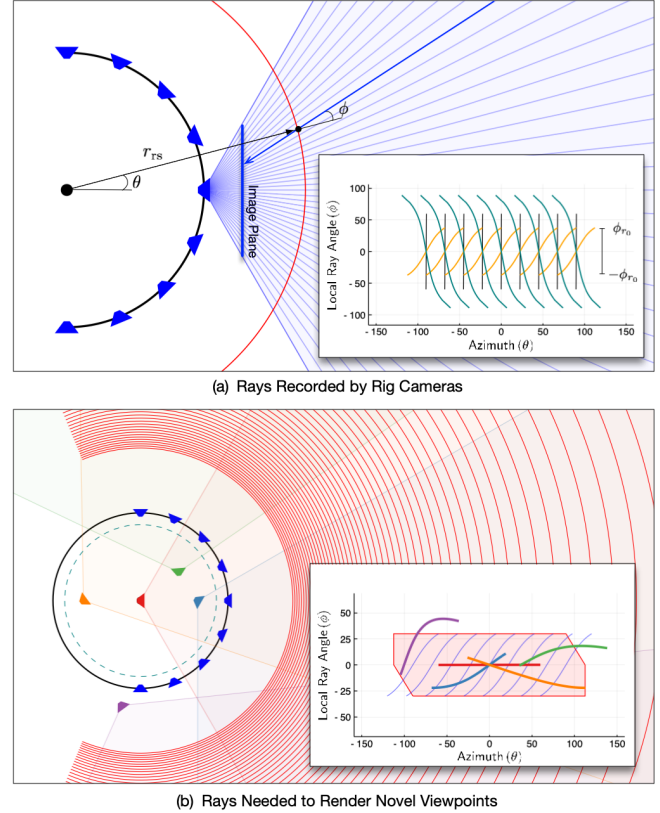


Fig. 13. Geometry of a spherical light field. (a) A ray space diagram parameterized by two angles  $(\theta, \phi)$  shows rays collected by each camera. Changing the radius  $r_{rs}$  where the ray space is parameterized reveals rig capabilities at the edge of the interpolation volume (teal), on the surface of the rig (black), and at the closest object distance (orange). (b) The ray space for a multi-sphere image, depicted as a parallelogram, shows the set of valid rays that can be used to synthesize novel viewpoints. Several novel viewpoints are shown, including some that require rays outside the MSI ray space.

with bold, colored curves in the diagram. We see that not all views can be rendered successfully. The green view sees “off the edge” of the MSI where there is simply no data available. The purple view is outside of the interpolation volume and some portion of its FOV will be filled with extrapolated data likely containing artifacts. In either case, this shows the limit of the LMI head volume both in terms of position and viewing direction. This is useful, for example, for knowing when to gracefully “fade out” invalid viewpoints in an interactive 6 DOF viewer, or for choosing valid example images while training a deep image synthesis network.