High-Quality Streamable Free-Viewpoint Video

Alvaro Collet Ming Chuang Pat Sweeney Don Gillett Dennis Evseev David Calabrese
Hugues Hoppe Adam Kirk Steve Sullivan
Microsoft Corporation

Figure 1: *Examples of reconstructed free-viewpoint video acquired by our system.*

Abstract

We present the first end-to-end solution to create high-quality free-viewpoint video encoded as a compact data stream. Our system records performances using a dense set of RGB and IR video cameras, generates dynamic textured surfaces, and compresses these to a streamable 3D video format. Four technical advances contribute to high fidelity and robustness: multimodal multi-view stereo fusing RGB, IR, and silhouette information; adaptive meshing guided by automatic detection of perceptually salient areas; mesh tracking to create temporally coherent subsequences; and encoding of tracked textured meshes as an MPEG video stream. Quantitative experiments demonstrate geometric accuracy, texture fidelity, and encoding efficiency. We release several datasets with calibrated inputs and processed results to foster future research.

CR Categories: I.3.0 [Computer Graphics]: General

Keywords: multi-view stereo, surface reconstruction, 3D video, mesh tracking, geometry compression, MPEG

1 Introduction

Interest in free-viewpoint video (FVV) has soared with recent advances in both capture technology and consumer virtual/augmented reality hardware, e.g., Microsoft HoloLens. As real-time view tracking becomes more accurate and pervasive, a new class of immersive viewing experiences becomes possible on a broad scale, demanding similarly immersive content.

Our goal is to transform free-viewpoint video from research prototype into a rich and accessible form of media. Several system components must work together to achieve this goal: capture rigs must be easy to reconfigure and support professional production workflows; reconstruction must be automatic and scalable to high processing throughput; and results must be compressible to a data rate close to common media formats. Visual quality from any angle must be on par with traditional video, and the format must be renderable in real-time on a wide range of consumer devices.

In this paper, we discuss how we address these challenges to create an end-to-end system for realistic, streamable free-viewpoint video at significantly higher quality than the state of the art. Our approach does not require prior knowledge of the scene content. It handles challenging scenarios such as multi-figure captures, detailed hand and facial expressions, free-flowing clothing, animals, and a wide variety of props from golf clubs to ropes to furniture. The example captures in Fig. 1 and 17 illustrate a range of scenarios.

Our main contribution is an integrated capture and processing pipeline that attains fidelity, robustness, and scale suitable for professional content production. Within this pipeline, several key advancements enable the high-quality results:

- A novel multimodal reconstruction that leverages RGB, infrared (IR), and silhouette information to recover dense shape detail.
- An extension to Poisson surface reconstruction (PSR) [Kazhdan et al. 2006] that performs implicit function clipping to improve outlier removal and maintain silhouette accuracy.
- A method to automatically identify and preserve perceptually important areas (e.g., faces, hands) in performances.
- A parallelizable, non-rigid mesh tracking framework to automatically handle topology changes by segmenting the performance into a sequence of deforming keyframe meshes.
- An encoding method to encapsulate tracked textured meshes and audio information into a single low-bandwidth MPEG stream.

To support further research, we publish a dataset with thousands of FVV frames, each comprising 106 calibrated and synchronized camera images together with intermediate and final processing results.

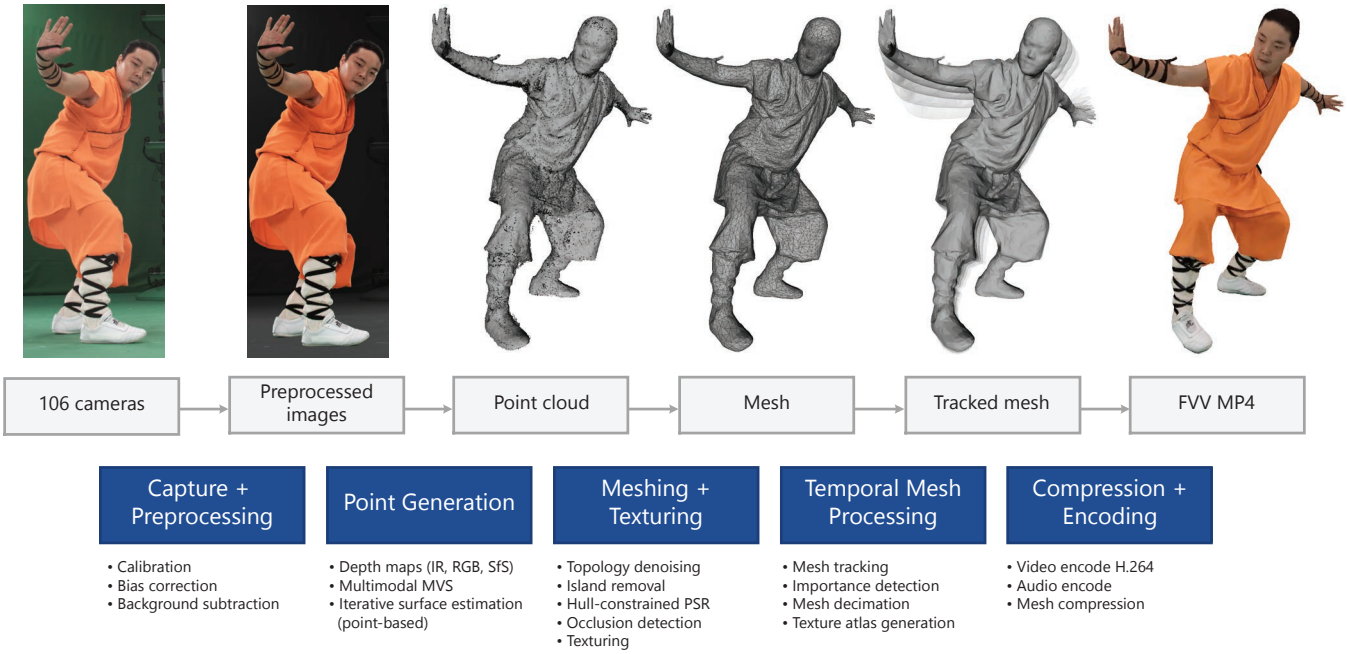


Figure 2: *Processing pipeline.*

2 Related work

We consider the problem of Free Viewpoint Video (FVV): a scene photographically captured in real-time that can be played back in real-time and appears like video from a continuous range of viewpoints chosen at any time in playback. In particular, we focus on a 360-degree volume captured and experienced from the outside looking in (as does the vast majority of the related work). The three main techniques proposed in this literature are: shape from silhouettes, freeform 3D reconstruction, and deformable models.

Shape from silhouette Silhouette-based approaches are relatively fast, easy to compute, and can provide a good approximation of the underlying shape, especially as the camera count grows.

One of the first silhouette-based systems is by Moezzi et al. [1997], motivated by fast reconstruction time and interactive playback. Matusik et al. [2000] improve the viewing experience with view-dependent rendering, and Franco et al. [2006] goes beyond polyhedral models to recover smooth shapes consistent with the input silhouettes. Ahmed et al. [2008b] generate temporally coherent meshes from per-frame shape-from-silhouette. To recover greater detail, Starck et al. [2007] enhance the visual hull with feature correspondences and computed the surface using graph cuts; Wu et al. [2011] enhance the visual hull with high frequency detail from shading variations. There are now commercial systems to produce FVV from silhouettes (e.g., 4DV [2007], DoubleMe [2014]).

Silhouette-based methods are fast and reliable, but quality can suffer as silhouettes alone cannot represent concavities such as self occlusions, eye sockets, or mouth openings, and can produce visible artifacts in non-captured views or textures with incorrect blends.

Freeform 3D surface reconstruction Kanade et al. [1997] pioneered FVV research with Virtualized Reality, using a dome of cameras to compute multi-baseline stereo and generate new views by triangulating merged depth maps. Narayanan et al. [1998] enhance the system by adding explicit surface reconstruction. Zitnick et al. [2004] compute dense depth maps, segment them into textured layers, and warp them to render virtual viewpoints. FreeD [2014] computes colored point clouds from still images for replays in TV broadcasting. Goldluecke et al. [2004] use space-time level sets to create temporal volumetric reconstructions. Labatut et al. [2007] compute a 3D Delaunay triangulation of the stereo points and reconstruct a surface by labeling the Delaunay tetrahedra as empty or occupied. Vlasic et al. [2009] optimize visual-hull meshes using normal maps estimated via active shape-from-shading. Similar to our work, Liu et al. [2010] compute 3D points from stereo and silhouettes, then refine and mesh to create a final surface.

Our work obtains higher-quality results by fusing silhouette, IR, and RGB information, and it constructs tracked meshes to enable efficient encoding of both geometry and texture for internet streaming.

Deformable models These systems exploit knowledge about the captured subject to parameterize and constrain the reconstruction process, and are often built to recover humans.

Carranza et al. [2003] pioneered fitting body models to images, deforming the precomputed shape of a specific performer to fit multiple silhouette images. Vlasic et al. [2008] deform a more general skeleton-driven template to fit visual hulls. Ahmed et al. [2008a] leverage calibrated lighting and reflectance variation to add temporally coherent detail over a smooth template. De Aguiar et al. [2008] add detail to a parameterized body model using silhouettes and multi-view stereo. Ye et al. [2013] leverage RGBD data to deform and texture Kinect-tracked skeleton templates. Huang et al. [2014] deform a human shape rigged using Pinocchio with a manually initialized skeleton.

Model-based systems produce very good results when the subject is well-represented in the parameterized space, especially when stereo information enhances the deformed model. The greatest challenges are requiring precomputed models of all content, and artifacts where the deformation cannot be represented in the parameterized space.

In comparison, our system does not require any prior knowledge of the content, though we can exploit it when available. We leverage silhouettes, IR and RGB multi-view stereo to create a highly detailed point cloud. We enforce silhouettes when meshing, and generate temporally coherent meshes and textures that can be encoded and played back smoothly in real time on low-powered devices.

Figure 3: *Typical stage setup. RGB and IR cameras and unstructured static IR laser light sources surround the capture volume.*

3 System overview

We present an overview of the end-to-end system in Fig. 2. To capture a performance, we use a greenscreen stage with high-speed RGB and IR cameras. To calibrate the stage, we capture: background images to assist in background segmentation; images of a calibration object to compute the camera parameters; and color calibration images to normalize pixel response across cameras for consistent texturing. We then capture the performance and preprocess the images to correct bias and segment out the background.

We compute a high-quality point cloud for each frame. We first process the images from stereo pairs and silhouette data to generate dense depth maps. We merge the depth maps using a multimodal multi-view stereo algorithm, and refine the resulting point cloud by locally fitting to the underlying surfaces. With this point cloud, we create a watertight mesh using a silhouette-constrained Poisson surface reconstruction. This mesh may contain spurious artifacts and disconnected components, which we clean up with topological denoising and island removal algorithms. We detect occlusions and compute surface coloring in the mesh by blending colors from visible cameras based on viewing angle and surface properties.

At this point we have a textured high-resolution mesh with different connectivity at every frame. We automatically select suitable keyframes and track them over sequences of frames to produce temporally coherent subsequences. We then determine which areas are perceptually important, and adaptively decimate and unwrap the meshes preserving important geometric and texture detail. We leverage mesh tracking to compute a temporally coherent mesh unwrapping and texture atlas. This is the raw FVV result, which we compress and encode in a single MPEG stream.

4 Capture and preprocessing

The current capture volume is a greenscreen stage encircled by synchronized high-speed video cameras, as shown in Fig. 3. The default configuration is 53 Sentech STC-CMC4MCL RGB cameras and 53 Sentech STC-CMB4MCL IR cameras covering a reconstruction volume of 2.8 m in diameter and 2.5 m high. Ten cameras are mounted in an array overhead, and 96 are mounted on 8 wheeled towers to support easy reconfiguration, different lighting conditions, and to maximize accuracy for a given scene size. We record 2048x2048 images at 30 Hz, though we can go up to 60 Hz for high-speed action. We add unstructured static IR laser light sources on the towers to provide strong IR texture cues. We surround the space with standard greenscreen to aid background segmentation, as a distinct background improves the accuracy of segmentation and thus the reconstructed shape. We use standard production lighting kits and typically choose a uniform lighting setup, but colored or imbalanced lights can be used to achieve specific looks.

We calibrate the stage before a performance to provide consistent orientation and ensure geometric accuracy and sharp textures. The calibration object is a wheeled octagonal tower with attached checkerboards, from which we compute the camera parameters with known scale, origin, and up vector. After each major lighting change, we adjust the white balance of each camera (using an integrating sphere as a constant D65 illumination source). We thus ensure photoconsistency across all video sources, as differences in black levels and color gains can lead to geometric and visual artifacts.

We control the capture stage through a central application that manages data synchronization, production metadata, audio recording, as well as preview tools and diagnostic views for directors. When capturing, the cameras write image data to standard PCs (six cameras per PC) which are then merged to a central disk array for processing.

More information on the configuration of cameras, including setup schematics, stereo pair configurations, pod/camera specifications, and frustum coverage can be found in the supplemental material.

5 Point generation

We introduce a multimodal 3D reconstruction method that adaptively combines RGB stereo, IR stereo, and Shape from Silhouette (SfSil) to improve reconstruction quality. Consider the capture in Fig. 4, which presents multiple challenges for point generation: untextured areas, hair, thin objects, etc. Most of the subject can be reconstructed using IR data (blue points), but non-IR-reflective areas (e.g., shoes, hair) cannot be accurately reconstructed. The RGB-based reconstruction (green points) recovers the shoes and hair better, but it is incomplete in other areas. The golf club is too thin to be reconstructed from either RGB or IR data, but its reconstruction from silhouette data (red points) is excellent. Our key insight is that no sensing modality can resolve every challenge, but that each modality is best suited to resolve different challenges.

Other works in the literature combine multiple modalities for 3D reconstruction, mainly photo-consistency and SfSil in volumetric methods (e.g., Hernandez et al. [2004], Sinha et al. [2005], Starck et al. [2007]). Song et al. [2010] combine stereo and silhouette data in a single point cloud, and reconstruct a mesh using Poisson surface reconstruction. In contrast with our work, their fusion of stereo and SfSil follows a fixed priority, in which stereo always takes precedence over SfSil. Our work is also related to depth map fusion approaches (e.g., [Campbell et al. 2008; Hiep et al. 2009]), though we fuse depth maps across modalities. PMVS2 [Furukawa and Ponce 2010] is regarded as the state of the art in 3D reconstruction. We compare our method to PMVS2 in Fig. 4.

We split the capture stage into logical groups of 2 RGB + 2 IR cameras that we call *pods*. Our algorithm is split into three components: in *Depth Map Generation*, we compute independent depth maps with confidences for each modality for each pod; in *Multimodal MVS* we select the best depth values in each pod and refine them with a combined RGB/IR score; and in *Iterative Surface Estimation* we refine the positions and normals of the oriented points by locally fitting the underlying surfaces. We perform a final photoconsistency check based on the combined RGB/IR score to cull outliers.

5.1 Depth map generation and confidence estimation

For each pod k, our goal is to compute dense depth maps D^k_{IR}, D^k_{RGB}, D^k_{SfSil} and confidence maps Φ^k_{IR}, Φ^k_{RGB}, Φ^k_{SfSil} for IR, RGB, and SfSil, respectively. We assume that each camera C in pod k has a similar unobstructed view of the scene. We define the reference view C^k_{ref} for pod k as the camera whose center is closest to the average camera center in pod k. We transform each depth map in pod k to C^k_{ref}.

We compute D^k_{IR} and D^k_{RGB} with GPU PatchMatch [Bleyer et al. 2011] because it produces near-state-of-the-art results efficiently, but

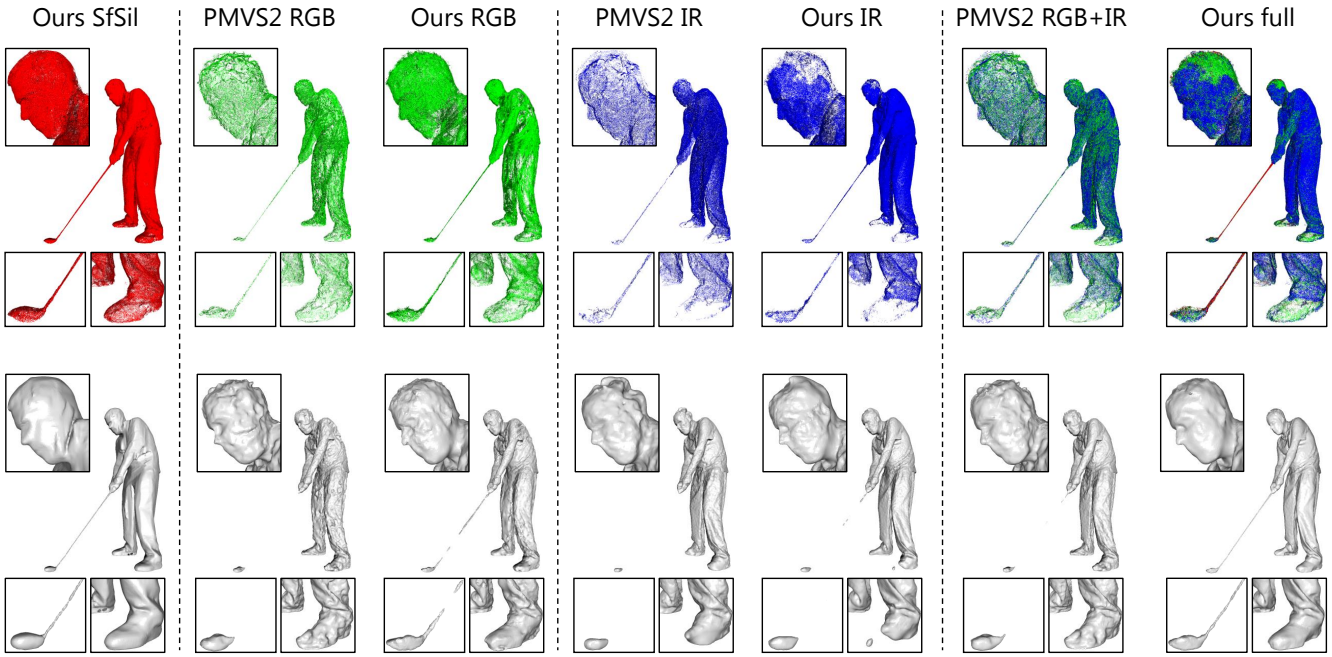


Figure 4: *Reconstructions of different modalities in PMVS2 and our method. All reconstructions (top) are meshed (bottom) with our silhouette-constrained PSR with the same parameters. The main body shape is best reconstructed using IR in both methods. The hair and shoes are not IR-reflective and cause artifacts in IR, so these regions are better reconstructed from RGB. The golf club seems to be properly reconstructed using RGB+IR in both PMVS2 and our method, but noisy depth values and inconsistent normals in such a thin object cause failures in surface reconstruction. Conversely, SfSil obtains good points and normals for such thin objects. Our final reconstruction (far right) combines the best of each modality and provides superior reconstruction quality on a wider range of scenarios and materials.*

our method also works with alternative dense stereo algorithms. We compute D^k_{SfSil} as in Song et al. [2010]: we first generate the visual hull octree as the intersection of all background masks; we mesh the octree with Marching Cubes; and we project the mesh into D^k_{SfSil} with a Z-buffered rasterization.

Consider the confidence maps Φ^k_{IR}, Φ^k_{RGB}, Φ^k_{SfSil}, where each value is in range $[0, 1]$. For stereo, there exist many alternatives to compute confidences (see Hu et al. [2012] for a survey). We only need to determine which modality is more reliable, so we use a simple textureness measure $T^k_{\text{IR}}(p), T^k_{\text{RGB}}(p)$ by computing the average of the Sobel operator over a window centered at pixel p. High values of $T(p)$ indicate abrupt variations in intensity, and therefore more information for stereo methods. We normalize $T(p)$ using the negated Cauchy M-estimator, so that $\Phi^k(p) = 1 - 1/(1 + \left(T^k(p)/\alpha\right)^2)$.

We measure Φ^k_{SfSil} based on the range of valid depth values for a given silhouette pixel, which we call the valid ray length (VRL). Given that a silhouette pixel only intersects the visual hull in one point, the VRL is an estimation of uncertainty. Consider a silhouette pixel s in a background mask i, and $P(s, C_i, d)$ the 3D point from camera center C_i along the ray through s at a distance d. Then, $\text{VRL}(s)$ is the range of depth values d for which P is categorized as foreground when projected into all background masks for all pods. We normalize $\text{VRL}(s)$ with the Cauchy M-estimator, so that $\text{VRL}'(s) = 1/(1 + (\text{VRL}(s)/\beta)^2)$.

We compute $\text{VRL}(s)$ for every silhouette pixel in every image. We associate $\text{VRL}(s)$ to all nodes in the visual hull octree that project into s. We compute Φ^k_{SfSil} by projecting the confidences of the visual hull octree into C'^k_{ref} with a Z-buffered rasterization.

In our implementation, we learn the Cauchy normalization parameters α, β using grid search over a training set of 20 frames of known geometric shapes. The learned parameters result in SfSil confidences larger than stereo when VRL < 2 cm, and SfSil confidences smaller than average stereo confidences when VRL > 6 cm.

5.2 Multimodal MVS

In this section, we merge D^k_{IR}, D^k_{RGB}, D^k_{SfSil} into a single depth map per pod D^k, and refine it with an RGB/IR photoconsistency metric.

RGB/IR MVS photoconsistency We cannot directly compute a multi-view photoconsistency score because the data is not pixel-wise comparable across modalities. Instead, our metric is a confidence-weighted average of IR and RGB photoconsistency.

Consider the pairwise score $\text{NCC}(W_{p^i}(\pi_{\text{ref}}), W_{p^j}(\pi_{\text{ref}}))$ for a pair of patches W_{p^i} and W_{p^j} centered around p^i and p^j in images i and j, warped with a planar homography with respect to a reference plane π_{ref}. To extend this score to multiple views, Goesele et al. [2006] select a reference image and aggregate the score of p in each view with respect to the reference image. In our case, we need one reference image for each modality (IR and RGB) to compute the multimodal score. We also need a single reference plane across all images, so that all compared patches refer to the same physical area. To extend the score to multiple views, we aggregate all pairwise scores (with respect to the reference image in each modality) weighted by their normalized textureness confidences ($T^k_{\text{IR/RGB}}(p)$).

Joint IR/RGB optimization To optimize a 3D point P in MVS we require the following: initial position P; initial normal n; confidence ϕ; visibility set $V(P)$ (i.e., the set of images in which p is visible); reference images $I^{\text{RGB}}_{\text{ref}}$, $I^{\text{IR}}_{\text{ref}}$; and reference plane π_{ref}.

Consider pixel p in pod k with highest-confidence modality mod. We initialize its depth $D^k(p) = D^k_{\text{mod}}(p)$, and its confidence $\phi(p) = \Phi^k_{\text{mod}}(p)$. We initialize its normal $N^k(p)$ depending on mod: for IR and RGB, we estimate $N^k(p)$ via plane fitting on the depth map $D^k_{\text{mod}}(p)$ over a 3×3 window centered in p; for SfSil, we estimate $N^k(p)$ from the corresponding face normal in the visual hull mesh.

The choice of $V(P), I^{\text{RGB}}_{\text{ref}}, I^{\text{IR}}_{\text{ref}}, \pi_{\text{ref}}$ can be dynamic (with $V(P), I^{\text{RGB}}_{\text{ref}}, I^{\text{IR}}_{\text{ref}}, \pi_{\text{ref}}$ computed for every point depending on the

point normal and occlusions) or static (with constant $V(P)$ set to all cameras in pod k). In our experiments, a dynamic visibility set $V(P)$ across all cameras provided no visible improvement over a fixed $V(P)$ comprising only the cameras in pod k, while the latter is significantly faster. Therefore, we choose a fixed $V(P)$ for all points in pod k, as well as the reference images $I^{\text{RGB}}_{\text{ref}}$, $I^{\text{IR}}_{\text{ref}}$ closest to the pod center, and reference plane $\pi_{\text{ref}} = C_{\text{ref}}$.

Following [Furukawa and Ponce 2010], we optimize a set Θ of three parameters for each point P: its depth, and the two normal angles in spherical coordinates. Point P can only move along the ray passing through P and the center of its reference camera C^k_{ref}. We optimize Θ with our multimodal MVS score using a confidence-aware gradient descent, as we assume that the initialization is already fairly close to the optimum. Note that the textureness confidences in the multimodal MVS score depend on Θ and should be recomputed at every iteration. We leverage our confidence estimation to weight the default gradient descent step size as $\omega' = \omega_0(1 - \phi(p))$. This weighting scheme is particularly useful to ensure that high-confidence SfSil points are not modified by misleading IR/RGB information near depth discontinuities and edges. If the point moves outside the visual hull in an iteration, we terminate the optimization early and mark the point as invalid.

5.3 Iterative surface estimation

We combine depth maps and normals from all pods and refine their positions P_i and normals N_i using a global optimization to make them consistent with all views. Our approach is related to Moving Least Squares (MLS) projection [Alexa et al. 2001] in that we iteratively fit local planes that approximate the underlying surface. In addition to point positions, we augment MLS with a formulation that also considers colors and normals, updated at each iteration. Our algorithm iterates between the three steps described below.

1. Find point neighborhood. For each point P_i, we define its neighborhood $\mathcal{N}(P_i)$ as a ball centered around P_i which contains at least K nearest neighbors and has minimum radius r. The parameters K and r control the amount of smoothing.

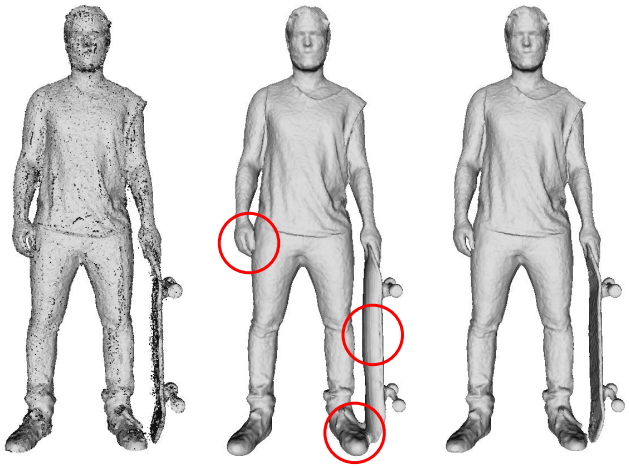
2. Estimate local plane. Given P_i and $\mathcal{N}(P_i)$, we estimate the best local plane S_i approximating the underlying surface at P_i according to a combination of metrics. Specifically, we estimate S_i as

$$\arg\max_{S_i} \sum_{j \in \mathcal{N}(P_i)} \phi(P_j) F_{\text{P}}(P_j, S_i) F_{\text{N}}(N_i, N_j) F_{\text{C}}(P_i, P_j), \quad (1)$$

where all scores are in range $[0, 1]$ (with 1 as the best possible score). The position score $F_{\text{P}}(P_j, \hat{S}_i)$ measures the Euclidean distance between P_j and the plane S_i, normalized as in Section 5.1. The normal score $F_{\text{N}}(N_i, N_j) = \max(N_i^\top N_j, 0)$ measures compatibility in normals: points with orthogonal normals are unlikely to belong to the same surface. The color score $F_{\text{C}}(P_i, P_j)$ compares the color of P_i and P_j in CIELAB space, normalized as in Section 5.1. $F_{\text{C}}(\cdot, \cdot)$ encodes the intuition that two points with dissimilar color are not likely to be in the same surface, similar to the adaptive support weights in [Bleyer et al. 2011]. We compute the color for P_i from the RGB reference image in the pod that generated P_i. Overall, we estimate surface S_i at P_i using those points in $\mathcal{N}(P_i)$ with similar color and normal, but not oversmoothing across large variations in color or curvature. We calculate S_i by drawing multiple random subsets of 3 points from $\mathcal{N}(P_i)$, estimating a hypothesis plane $\hat{S}_i$ from them, and keeping the $\hat{S}_i$ with best surface score according to Eq. (1).

3. Update point and normal. We set the position of P_i as the projection of P_i on S_i, and its normal N_i to the plane normal N_{S_i}.

We iterate steps 1-3 until there is limited variation in the estimated surfaces. In practice, two iterations are enough to calculate high-quality points and normals ready for meshing.



Input points | Poisson surface recon. | Hull-constrained PSR

Figure 5: *Surface reconstruction quality in Poisson surface reconstruction (PSR) and hull-constrained PSR. The top of the skateboard is not IR-reflective and is occluded in most camera views; the bottom of the shoes is not visible from any camera. Without constraining the surface to the visual hull, regular PSR produces bulging artifacts and fails to recover some of the sharp features such as the thumb.*

6 Meshing and texturing

We generate a textured simplified mesh from the point cloud in each frame. This involves several steps: reconstruction of a high-resolution triangle mesh, adaptive decimation to create a coarser mesh, parameterization over a texture atlas domain, and resampling of scene color into the texture. We focus on our novel contributions: adding implicit-function clipping to screened Poisson surface reconstruction, and using automatic face detection to improve the perceived quality of both geometry and texture.

6.1 Hull-constrained surface reconstruction

Screened Poisson surface reconstruction (PSR) [Kazhdan et al. 2006; Kazhdan and Hoppe 2013] is a commonly used technique for point cloud meshing, as it is both efficient and resilient to data imperfections. The basic approach is to transform the oriented points into a sparse 3D vector field, solve for the scalar function whose gradient best fits the vector field (i.e., a Poisson equation), and finally extract an isosurface of this scalar function.

Although PSR is guaranteed to produce watertight surfaces, "bulging" artifacts often arise in regions with missing data, as shown in Fig. 5. We address this issue by intersecting the reconstructed implicit solid with the visual hull, effectively constraining the reconstructed surface by the background masks.

Rather than modifying the scalar coefficients used within the Poisson solver, which are associated with smooth B-spline basis functions, we find it more efficient and accurate to operate on the final scalar values just before isosurface extraction. We project each octree cell corner as-needed into the background masks. If the cell is considered background in any mask, we set its scalar value to a negative number to enforce that it lies outside the resulting isosurface. As demonstrated in Fig. 5, the resulting meshes are free of the earlier bulging artifacts and also have sharper features. Alternatively, Shan et al. [2014] propose to add a soft constraint on free-space in PSR's energy formulation to achieve a similar effect.

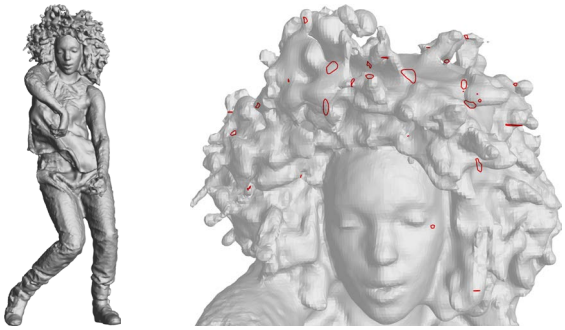


Figure 6: *In this frame, topology denoising closes 35 handles to simplify the mesh genus from 37 to 2. Non-separating cycles are illustrated in red (many of these cycles are normally occluded).*

6.2 Topology denoising

Surfaces reconstructed from noisy point clouds often have tiny topological artifacts (handles and tunnels) not present in the scanned objects [Guskov and Wood 2001]. These artifacts are nearly invisible, yet they raise the surface genus and can hinder the efficiency of geometry processing including mesh simplification and surface parameterization [Wood et al. 2004]. Most importantly, as discussed in Section 7.1, we find that robust characterization of surface genus provides a helpful heuristic in identifying keyframes for mesh tracking. We therefore have to remove topological noise.

We first considered the technique of Wood et al. [2004], designed for isosurfaces as in our pipeline. However, it assumes that the isosurface is created from a regular 3D grid; it has not yet been extended to the case of octrees. Instead, we adapt the scheme of Guskov et al. [2001], which iteratively examines the Euler characteristic ($\chi = \#\text{vertices} - \#\text{edges} + \#\text{faces}$) of small mesh neighborhoods. A neighborhood that spans a topological handle (i.e. $\chi \neq 2$) contains a non-separating cycle – a loop of edges that, when cut, still leaves the surface connected [Erickson and Whittlesey 2005]. If the length of this cycle is sufficiently short (10 cm in all our results), we perform the cut and close each of the two resulting mesh boundaries using a fan of triangles. Insertion of the new triangles may result in self-intersections, but we have not found this to be a concern for our application. Fig. 6 shows an example.

6.3 Adaptivity using importance functions

Rendering 3D objects in a streaming environment imposes a strict budget on geometric and texture complexity. To apportion mesh vertices and texture area, mesh simplification and parameterization tools commonly use low-level cost functions based on surface geometry, visual fidelity [Lindstrom and Turk 2000], or local surface curvature [Lee et al. 2005].

We find most helpful to include high-level contextual knowledge to help guide this process. For instance, viewers of a human performance are much more attuned to perceiving nuances in facial expression than subtle deformations in clothing. Our approach is to define a surface *importance function* based on automated object detection in the original input views. We currently look for human faces, but the method is generic enough to work with any image-based object detector.

To create the importance function, we first compute per-pixel importance values f_i in each input view i based on whether a pixel is part of a detected object (e.g., human face). For instance, the face detector identifies rectangles in the image and assigns f_i based on the detection confidence. Then, for each mesh vertex, we compute an importance value F using a simple median voting of the f_i at the pixels in which it is visible. This field F is illustrated in Fig. 7.

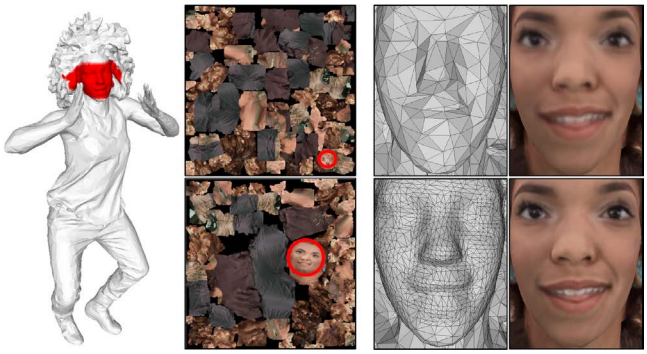


Figure 7: *Compared to default results (top row), introducing a surface importance field significantly improves the perceived quality of the reconstructed output frames (bottom row). The importance function, shown in red on the left, is computed automatically using face detection. The sub-images show how adaptive parameterization allocates 4x greater texture atlas area to the face, and how adaptive decimation retains greater geometric detail there.*

Adaptive allocation of polygons We simplify the high-resolution mesh using a variant of the Quadric Error Metric (QEM) [Garland and Heckbert 1997]. In particular, we scale the per-vertex QEM terms based on the F importance weights (typically by a factor of 100). Thus, vertices with higher importance are much more likely to be retained during simplification.

Adaptive allocation of texels We compute a texture atlas parameterization using UVAtlas [Microsoft 2011]. A parameter in UVAtlas for each mesh triangle is the Integrated Metric Tensor (IMT), a 2x2 positive-definite matrix that defines the inner product for computing surface distances. A larger IMT for a triangle results in a larger allocated area during unwrapping. For each triangle, we scale the identity IMT by the average of importance values F at its vertices.

6.4 View-independent Texturing

Diffuse texturing an object by stitching RGB images is a well-studied problem. It requires carefully registering images to geometry and blending of samples from different images. Image-to-geometry registration can be difficult in cases of inaccurate camera calibration or geometry reconstruction [Gal et al. 2010; Zhou and Koltun 2014]. Blending can be challenging due to lighting variation from different viewpoints [Lempitsky and Ivanov 2007; Chuang et al. 2009]. We find that, thanks to our controlled and well calibrated environment, direct image projection followed by normal-weighted blending of non-occluded images yields sufficiently accurate results. Note that although our setup has the potential to produce high-quality view-dependent textures, these are not yet suitable for internet streaming even when compressed using state-of-the-art methods [Casas et al. 2014; Volino et al. 2014].

7 Mesh tracking

Generating meshes independently per frame produces a surface geometry which is spatiotemporally coherent yet tessellated inconsistently. We need consistent tessellations to support compression.

Extracting a spatiotemporally coherent mesh out of a 4D capture is an active area of research. Template-based approaches address the problem by introducing an offline process to acquire/learn a static template mesh [Borshukov et al. 2005; de Aguiar et al. 2008; Gall et al. 2009; Li et al. 2009; Wand et al. 2009; Klaudiny et al. 2012; Zollhöfer et al. 2014]. They commonly assume that the subjects are topologically consistent throughout the capture, thus precluding the more complicated scenarios of surface interaction,

object fractures, prop insertion/removal, scene changes, etc. More recent methods adapt the template by amending the triangulation in the course of registration to record topological changes [Letouzey and Boyer 2012; Bojsen-Hansen et al. 2012]. It has not yet been demonstrated that such topological surgery can produce consistent atlas parameterization over a sparse number of templates for complex shapes. Finally, all these methods rely on the assumption that the template can be robustly deformed and registered with *all* frames by an ICP-like process without failure; they are not resilient to severe deformation, sudden fast motion, and elastic/free-form material.

To deliver streamable content, the restriction of a single mesh representation can be relaxed. Drawing the idea from ubiquitous I-frame based video compression, we reformulate the problem as finding a sparse set of *keyframes* whose meshes well abstract their neighbor frames. The approach is similar to [Klaudiny et al. 2012; Huang et al. 2014] where a pool of *keymeshes* is used to describe the capture. The difference is that by design we enforce continuous keyframe coverage to support better texture compression and random-access video seeking. Also, our approach is not specific to human subjects and requires no manual initialization.

Algorithm overview First, we estimate a feasibility score of each frame being a keyframe (Section 7.1). We choose the most promising non-tracked frame (according to the feasibility score) as keyframe i. We then perform a nonlinear registration to fit the keyframe meshes to neighboring frames (in both forward and backward directions), a process we refer to as mesh tracking (Section 7.2). During this process, we monitor the fitting error by measuring the Hausdorff distance [Aspert et al. 2002]. We associate successfully registered frames with keyframe i, and terminate the tracking for keyframe i when the error is above a certain tolerance. We repeat this process on the remaining frames until every frame is associated with a keyframe. The algorithm is summarized in Algorithm 1.

Algorithm 1 Coherent Tessellation via Mesh Tracking

procedure CoherentTess($\{\mathcal{M}\}$) ▷ Input meshes
  $\{\Psi\} = \emptyset$ ▷ Temporal mesh sequences
  $\{\mathcal{S}\} \leftarrow$ keyframe feasibility for all frames ▷ **Section 7.1**
  Sort $\{\mathcal{S}\}$ into priority queue Q ▷ Key: i, Value:$\mathcal{S}_i$
  while $Q \neq \emptyset$ **do** ▷ Until we cover all frames
    $i \leftarrow Q.RemoveHighest()$ ▷ Most feasible frame
    $K_i \leftarrow \mathcal{M}_i$ ▷ Set as keymesh
    $\{\widehat{\mathcal{M}}\}_f = \text{MeshTracking}(i,+1)$ ▷ Forward Tracking
    $\{\widehat{\mathcal{M}}\}_b = \text{MeshTracking}(i,-1)$ ▷ Backward Tracking
    $\{\Psi\} \leftarrow \{K_i, \{\widehat{\mathcal{M}}\}_f \cup \{\widehat{\mathcal{M}}\}_b\}$ ▷ Save subsequences
  return $\{\Psi\}$ ▷ Return temporal sequences

procedure MeshTracking(i,δ) ▷ i: keyframe, δ: offset
  $\{\widehat{\mathcal{M}}\} = \emptyset$ ▷ Initialize subsequence
  while $Q.HasEntry(i+\delta)$ **do** ▷ Proceed if uncovered
    $\widehat{\mathcal{M}}_i \leftarrow K_i$ ▷ Initialize transient mesh
    $\widehat{\mathcal{M}}_{i+\delta} \leftarrow$DeformInto($\widehat{\mathcal{M}}_i$,$\mathcal{M}_{i+\delta}$) ▷ **Section 7.2**
    $e \leftarrow$ FittingError($\widehat{\mathcal{M}}_{i+\delta}$,$\mathcal{M}_{i+\delta}$) ▷ Compute error
    if $e < \epsilon$ **then** ▷ ϵ: error threshold
      $\{\widehat{\mathcal{M}}\} \leftarrow \widehat{\mathcal{M}}_{i+\delta}$ ▷ Save the tracked mesh
      $Q.Remove(i+\delta)$ ▷ This frame is covered
      $i \leftarrow i+\delta$ ▷ Move on to the next frame
    else ▷ Error above threshold
      return $\{\widehat{\mathcal{M}}\}$ ▷ Keymesh no longer usable
  return $\{\widehat{\mathcal{M}}\}$ ▷ Reach end of sequence

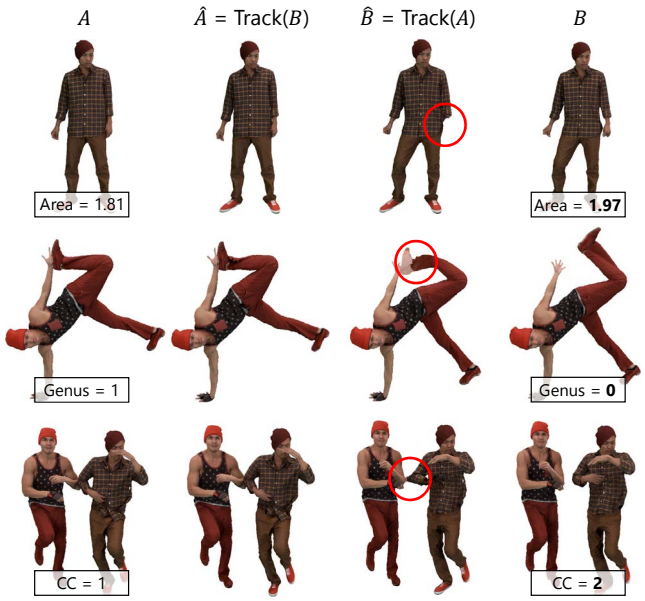


Figure 8: *Three examples illustrating our choice of heuristics for keyframe selection. Each shows how frame B is preferable as a keyframe because it can deform to approximate frame A with fewer artifacts than vice versa, and this corresponds to frame B having (1) greater surface area, (2) smaller genus, or (3) more connected components than frame A.*

7.1 Keyframe prediction

We wish to obtain the minimum set of keyframes that can register all other frames within some tolerance. This problem is similar to the famous set covering problem (which is NP-hard), except that we cannot know a priori how many frames can be registered by a candidate keyframe without actually performing the expensive nonlinear registration. Our solution entails a greedy algorithm that selects the most promising keyframes and enforces that each keyframe covers a continuous frame range.

We compute a feasibility score S_i for each frame i to find the most promising frames. The prediction is based on three key observations, illustrated in Fig. 8. First, we observe that it is often easier to deform a surface of larger area into one of smaller area (Fig. 8 (top)). Second, a lower-genus surface is likely to be deformed as a higher-genus surface, but *not* vice versa (Fig. 8 (middle)). Third, we observe it is easier to deform a surface with more connected components into one with fewer connected components, as the meshing algorithm might incorrectly merge independent subjects in contact (Fig. 8 (bottom)). The feasibility score combines these three observations as

$$\mathcal{S}_i = \sum_{c \in \mathcal{C}(i)} \left(1 + (g_{max} - g_c) + \frac{A_c}{A_{max}+1}\right), \qquad (2)$$

where $\mathcal{C}(i)$ is the number of connected components in the reconstruction; g_i, A_i denote the genus/area of the surface; and g_{max}, A_{max} denote the largest genus/area of components from all frames. Note that for robust estimation of genus, the meshes need to be filtered with the topological denoising process in Section 6.2.

7.2 Nonlinear registration

The described mesh tracking framework is designed to work in conjunction with any nonlinear registration algorithm. In this work, we implement the state-of-the-art non-rigid ICP algorithm of Li et al. [2009], though other approaches (e.g., [de Aguiar et al. 2008] and [Zollhöfer et al. 2014]) can be adapted as well.

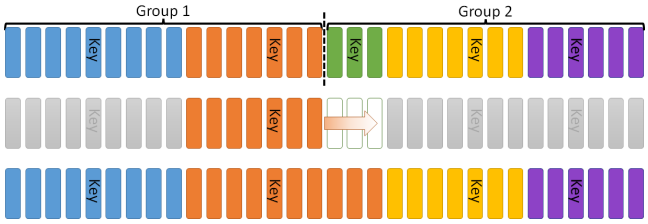


Figure 9: *To parallelize mesh tracking, we first partition frames into groups and process groups in parallel using Algorithm 1. To improve the possibly suboptimal result near group boundaries, we join the frames covered by the boundary keyframes (orange and green) and reapply Algorithm 1. The algorithm often completes using a single keyframe (bottom), thus effectively removing a keyframe (green).*

To fit a keyframe mesh to neighboring frames, we solve for a spatially-varying affine deformation field $f : M \to \mathbb{R}^3$ that minimizes the composite energy

$$E(f) = E_{\text{fit}} + \alpha_{\text{rigid}} E_{\text{rigid}} + \alpha_{\text{reg}} E_{\text{reg}}, \quad (3)$$

where E_{fit} measures how well the deformed surface fits the target surface, E_{rigid} aims to maximize rigidity of deformation, and E_{reg} accounts for smoothness of deformation. E_{reg} and E_{rigid} are computed with respect to the embedded deformation graph as in [Sumner et al. 2007]. The algorithm iteratively computes closest point correspondences for all vertices and updates the fitting constraint E_{fit} accordingly. When there is no significant improvement of energies between successive iterations, the rigidity and regularization constraints are relaxed by halving α_{rigid} and α_{reg}, in order to avoid local minima often observed during large-scale deformation.

We minimize Eq. (3) using the standard Gauss-Newton method from [Sumner et al. 2007], which requires solving the normal equations by (sparse) Cholesky factorization. We use SimplicialLLT [Guennebaud et al. 2010] for the task, and we precompute the nonzero factorization pattern to reuse it throughout the minimization. We have found that the normal matrix can occasionally be ill-conditioned, resulting in a factorization failure. To avoid this issue, we progressively add a scalar matrix $10^{r-8}I$ (where r is the number of retries) to the system until factorization succeeds.

7.3 Parallel implementation

Algorithm 1 is greedy and inherently sequential, and can become a processing bottleneck. We parallelize Algorithm 1 by splitting the frame sequence evenly into N groups and processing each group independently. To avoid redundant keyframes near group boundaries, we join the frames covered by the two keyframes nearest the boundary into a new group and reapply the greedy algorithm, in an attempt to merge the two keyframes. Fig. 9 demonstrates the idea using a simplified two-group example.

8 Compression and encoding

The final output is a streamable MPEG-DASH [ISO/IEC 23009-1 2014] file, the standard for adaptive streaming and playback. Our maximum target bitrate is 15.6Mbps, corresponding to Netflix's maximum video streaming bitrate. We store the texture and audio in the standard MPEG video and audio streams. We embed the mesh data as a custom Network Abstraction Layer (NAL) unit inside the video stream, because the support for custom streams is non-existent in the most widespread multimedia platforms (e.g., Microsoft Media Foundation). An additional advantage of this NAL unit mesh embedding is that third-party MPEG editing tools can modify the container without losing the mesh data.

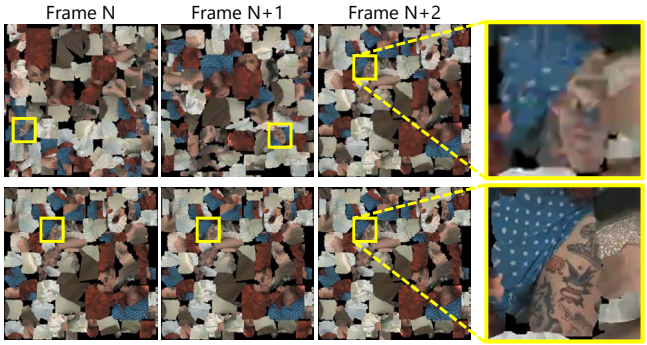


Figure 10: *Impact of mesh tracking on texture compression quality. We show three consecutive frames compressed at a fixed bitrate of 2 Mbps without mesh tracking (top), and with mesh tracking (bottom). The yellow inset, which corresponds to the same physical area in all frames, highlights the quality benefit of having a temporally consistent parameterization (thanks to mesh tracking).*

Audio and video compression We compress the audio and video streams using AAC and H.264, respectively. Each frame of the video stream is a texture atlas. The key to compact representation of video textures is mesh tracking; without it, each mesh would be topologically different and result in a different 2D parameterization. Fig. 10 shows the impact of a consistent parameterization in texture compression quality: in the top row, non-tracked meshes are unwrapped inconsistently in consecutive frames, and result in poor quality. In the bottom row, the tracked meshes are unwrapped consistently and result in much higher quality for the same bitrate.

Mesh compression Mesh tracking is also key for efficient mesh compression. There exist multiple methods in the literature to encode dynamic meshes with consistent topology (see [Yu et al. 2010] for a survey). In this paper, we show how a simple motion prediction scheme already achieves our bitrate goal when used in conjunction with mesh tracking. Alternative methods (e.g., [Vasa and Skala 2007]) should decrease this bitrate even further.

We split the mesh sequence into a series of keyframes and predictive frames. The keyframe meshes contain both geometry and connectivity information. We encode the geometric information (vertex positions and UV coordinates) quantized to 16 bits. We encode connectivity information as a delta-encoded, variable-byte tristrip. The predictive frames contain only delta geometry information. We use a linear motion predictor to compute the delta geometry, which we then quantize and compress with Golomb coding [Golomb 1966].

9 Implementation

Inputs and outputs Each recorded capture contains the following data: input video sequence from every camera; audio input; background capture (one frame per RGB camera); metadata (calibration, camera configuration, and processing parameters). The input data bandwidth is approximately 60Gbps at 30 fps (120 Gbps at 60 fps), requiring dedicated capture machines and dual 10-Gbps uplinks between the capture stage and our central storage.

The output is an MPEG-4 file with an embedded mesh stream as described in Section 8. By default, we set mesh resolution to 10K triangles in single-subject captures and 20K in multi-subject captures, with atlas video resolution of 1024x1024 pixels encoded at 4 Mbps. The average output bitrate is 9.5 Mbps. The output bandwidth is reduced by 3 orders of magnitude compared with the input.

Processing pipeline Our pipeline implementation aims to maximize (a) CPU and GPU parallelism on a single machine, and (b) multi-machine parallelism. We achieve both goals with task-

based parallelism and shared storage, considering data dependencies between tasks and launching independent tasks in parallel. Most stages can be parallelized per camera or per frame; the exceptions are mesh tracking (parallelized per group of frames) and the final encoding (not parallelized). The processing pipeline is distributed for multi-machine processing and, potentially, cloud processing.

Processing hardware To process the presented datasets, we use 61 Intel Xeon E5-2630, 2.3 GHz machines, each with dual 12-core processors, 64 GB of RAM, and an AMD Radeon R9 200 GPU.

10 Experiments

We evaluate the full system in terms of overall throughput, geometric quality, and texture quality. We use the five datasets illustrated in Fig. 11: DancingDuo (886 frames), Haka (173 frames), Dress (1157 frames), Kendo (740 frames), and Lincoln (508 frames) for a total of 3464 frames and 367K images. Three of these datasets are single-subject captures encoded at 10K triangles (Dress, Kendo, and Lincoln) and two are dual-subject captures encoded at 20K triangles (DancingDuo and Haka).

We also analyze the impact of the camera configuration on output quality, by running our pipeline on reduced configurations of our datasets as well as on publicly-available 3rd party datasets.

System throughput Table 1 shows the processing throughput for each dataset. On average, processing a frame on a single machine takes 28.2 minutes. With parallelization on the processing farm, throughput is improved to 27.7 seconds per frame. The average lifespan of a keyframe across all datasets is 31.7 frames, ranging from 72.6 frames in Lincoln (with relatively easy body motions) to 24.6 frames in DancingDuo (with multiple occlusions and intricate movement). We observe that these results follow the expected behavior: datasets with complex motion and fast movement such as DancingDuo require more keyframes (because the tracking algorithm cannot follow the action for long) and larger output bitrates (because there are more keyframes to encode). On the other hand, datasets with slower motion, such as Lincoln, can be tracked more consistently, resulting in fewer keyframes and smaller output bitrates.

Table 2 shows the relative processing cost of each stage, averaged over the five datasets. Point generation is the most time-consuming stage, requiring 49% of the overall processing time (5% for depth map generation, 31% for MVS, and 13% for iterative surface estimation). Mesh tracking, which is also time-consuming (11%), has significant room for improvement using a GPU-based implementation [Zollhöfer et al. 2014].

Texture quality We validate texture fidelity by comparing renderings of our final results against the original images via leave-one-out cross-validation. Specifically, we artificially remove a pair of RGB cameras from the capture and then compare a rendering of the reconstructed model against the removed cameras, as shown in Fig. 12. We repeat the experiment eight times, removing a different pair of RGB cameras each time. The reported statistics in Table 3 show an average over all frames and all experiments for each dataset.

We measure texture fidelity in terms of PSNR and structural similarity (MSSIM) [Wang et al. 2004], which closely correlates with perceptual quality. We compute both PSNR and MSSIM over the image intensity only on pixels where the 3D model is reprojected.

The main sources of error between the rendered frames and the original images are high-specularity (light-dependent) areas, edge misalignments, and blooming effects. Fig. 12 compares two examples of ground-truth views and their corresponding rendered views. In Fig. 12(bottom row), note that the major differences between the two views are due to lighting reflections, which cannot be accurately represented without view-dependent texturing. Another important

Figure 11: *Datasets: DancingDuo, Dress, Kendo, Haka, Lincoln*

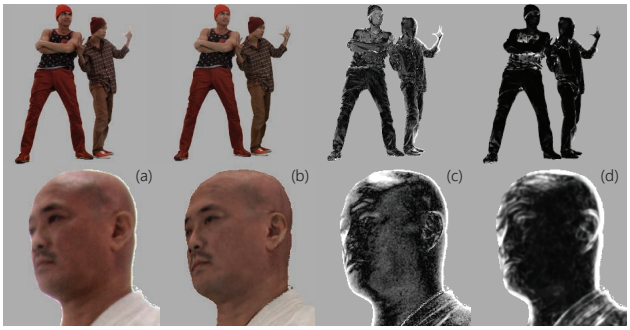


Figure 12: *Two tests of overall reconstruction accuracy using leave-one-out cross-validation. (a) ground-truth view (not present in reconstruction); (b) reconstructed view; (c) MSE; (d) SSIM.*

Scene	Frames	Avg points	Avg time/ frame (s)	Avg frames/ keyframe	Output Mbps
D.Duo	886	2.25M	29.1	24.6	12.1
Dress	1157	1.44M	27.7	27.5	8.6
Kendo	740	2.01M	26.0	35.2	8.3
Haka	173	2.70M	28.2	57.7	12.0
Lincoln	508	1.60M	27.5	72.6	7.9

Table 1: *Processing throughput and encoding performance analysis.*

Stage	Avg time (%)
Preprocessing	11%
Point generation	49%
Meshing	8%
Importance detection	2%
Mesh tracking	11%
Mesh decimation/unwrapping	4%
Texturing	7%
Compression and encoding	8%

Table 2: *Breakdown of processing time for each stage, averaged over the five datasets. The preprocessing step includes image I/O operations, bias correction, and background segmentation.*

	D.Duo	Dress	Kendo	Haka	Lincoln
PSNR (dB)	29.64	24.18	27.50	26.29	31.43
MSSIM	0.838	0.794	0.823	0.785	0.839

Table 3: *Measures of texture quality in cross-validation tests, averaged over all experiments (eight removed pairs of RGB cameras, one at a time) and all frames.*

source of error is blooming effects, where a camera captures the subject in front of a bright light source (see Fig. 14 for an example). This effect is particularly visible in Fig. 12(c, top), where the subject's hand disappears due to a blooming effect in the original image. The last source of error is due to edge misalignments caused by small geometry errors. These edge misalignments cause increased variability in PSNR scores compared to MSSIM scores. Dress, Kendo, and Haka contain features difficult to reconstruct accurately (hair, a fast-moving sword, and a Haka dance kilt, respectively), resulting in geometric errors and frequent edge misalignments. To improve

Figure 13: *Comparison of geometry and texture quality for different camera configurations, on Kendo dataset. See supplemental material/video for a more extensive comparison.*

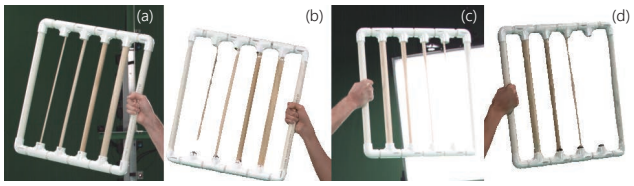


Figure 14: *Experiment to test geometric resolution, showing original views and reconstructions. In (a,b), the thinnest cylinder is partially reconstructed, whereas in (c,d), blooming from the rear light source causes the cylinder to be missed entirely.*

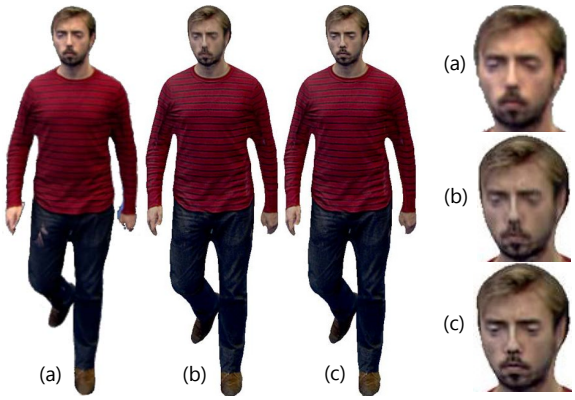


Figure 15: *Comparison of texturing schemes: (a) [Volino et al. 2014]; (b) Ours without adaptive allocation of triangles and texels; (c) Ours with importance-based adaptive allocation.*

upon the statistics in Table 3, we would need to faithfully render light-dependent effects, either with view-dependent texturing or with material estimation and relighting.

We analyze the impact of the camera configuration on the final geometry and texture quality in Table 4 and Fig. 13. We compare four configurations on Kendo: 8 RGB + 8 IR cameras for all processing (which we refer to as 'LG/LT'); 8 RGB cameras for texturing, with geometry computed from the 106-camera setup ('HG/LT'); 53 RGB cameras for texturing, with geometry computed from the 8 RGB + 8 IR setup ('LG/HT'); and the full 106-camera setup ('HG/HT'). We draw two conclusions: 1) it is more beneficial to use accurate geometry with fewer texturing cameras than coarse geometry with more texturing cameras, since accurate geometry enables the 8 texture views to align properly on the surface, resulting in fewer artifacts; and 2) a higher density of cameras enables a better geometric and texture reconstruction, when processed with algorithms capable of leveraging the additional data volume and complexity.

Geometric resolution As a testing methodology to evaluate the resolution of geometric reconstruction, we capture 300 frames of a ground-truth object moving throughout the volume, and count the number of frames in which it is reconstructed correctly. The ground-truth object consists of 4 wooden cylinders of different diameters, ranging from 6.35 mm to 25.4 mm (Fig. 14). The 25.4 mm and 19 mm cylinders are reconstructed in 100% of the frames, the 12.7 mm cylinder 99.7% of the frames, and the 6.35 mm cylinder 3.3% of the time. For a thin object, a main source of error is blooming, which makes the object indistinguishable when viewed in front of a light source, as shown in Fig. 14(c). According to this test, the minimum object size we consistently reconstruct is 12.7 mm (0.5 inches).

	LG/LT	LG/HT	HG/LT	HG/HT
PSNR (dB)	24.26	25.36	25.80	27.50
MSSIM	0.764	0.775	0.787	0.823

Table 4: *For Kendo dataset, texture quality measures for different camera configurations (Low and High number of cameras for both Geometry and Texture reconstruction).*

	RMS (mm)	Hausdorff (mm)	Sil. (%)
[Budd et al. 2013]	5.67	39.3	0.05
Ours (1 keyframe)	2.27	21.2	0.02
Ours (3 keyframes)	2.06	17.5	0.02

Table 5: *Comparison of tracking results on dataset 'DanWalk' from [Casas et al. 2014]. RMS and Hausdorff errors are measured with point-to-plane distances at all vertices of the original and tracked meshes. Silhouette error is computed as the percentage of pixels failing the silhouette constraints.*

Experiments on external datasets We evaluate our system on the 'DanWalk' dataset of [Casas et al. 2014; Volino et al. 2014], containing 28 frames (8 RGB cameras evenly spread in a 360-degree configuration) with per-frame meshes and tracked meshes.

Table 5 compares our mesh tracking results with those of [Budd et al. 2013] (used by [Casas et al. 2014]). The results confirm that our tracking algorithm obtains comparable or better results even with a single keyframe mesh. The use of multiple keyframe meshes becomes essential when tracking more complex and longer motions, such as the ones showcased in our datasets. Fig. 15 compares our texture reconstruction results with those of [Volino et al. 2014]. Although our simpler texturing does not reproduce view-dependent effects, it achieves comparable quality on this dataset, at a higher compression rate (2 Mbps versus 80 Mbps for theirs). Fig. 15(c) shows the benefit of adaptive mesh decimation and texture unwrapping, preserving greater detail on the face. Note that we could not compare our point generation, as we use a dense stereo method (PatchMatch, see Section 5.1) designed for narrow baseline stereo, which is incompatible with the 45-degree baseline of this dataset.

11 Conclusions

We have presented the first system to create high-quality, streamable free-viewpoint video suitable for a broad range of end-user scenarios. Our capture and processing are automated and reliable, producing results whose quality consistently exceeds state-of-the-art methods (see Fig. 17 for examples).

Our approach does not require prior knowledge of the scene, which allows us to recover a wide array of subjects and performances. By fusing RGB, IR, SfSil, and importance information, we can recover critical surface detail and maintain it even after reducing triangle count by two orders of magnitude. The ability to track meshes into coherent subsequences lets us create stable texture atlases, which in turn supports compressing and encoding for wide consumption of FVV as a media type. The pipeline has been implemented as a robust distributed system, and we capture and process multiple performances each day.

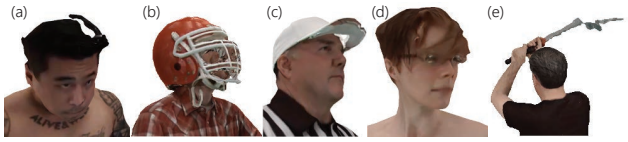


Figure 16: *Examples of poor reconstruction: (a) black hair has poor IR-reflectivity and lacks sufficient RGB detail; (b) helmet is reflective and has highly occluded geometry; (c) hat brim is thin and highly occluded; (d) glasses are reflective; (e) fast sword motion results in bad reconstruction.*

Future work Several types of content prove challenging and are active areas of work. We include examples of failure cases in Fig. 16. Three of the most important are specular materials, transparent/translucent objects, and hair.

Specular materials may cause errors in pixel correspondence during reconstruction due to view-dependent reflections. In some cases, our system can fall back to shape-from-silhouette reconstruction to produce a viable result, but often with noticeable artifacts. One approach with potential is to estimate basic material properties using all views and incorporate that knowledge into reconstruction.

Specularities also cause trouble for our pre-lit texture maps which cannot reproduce view-dependent visual response. We have found in practice that subject motion does change specular response enough to distract from the inaccuracy, but moving the view around paused frames is still problematic. We believe this should be feasible while maintaining low bandwidth as the specular components can be factored out and compressed quite efficiently.

Similarly, transparent and translucent surfaces (e.g., glasses, visors) are challenging to both reconstruction and rendering. A given pixel can have two or more valid depths, so stereo methods cannot recover them and conventional textured meshes cannot represent them. As before, one potential approach is to estimate basic material properties as part of the reconstruction process itself.

Hair presents a different challenge in that certain styles or motions cannot be faithfully represented using a triangle mesh representation. The structure of hair spans a broad spectrum, from dense hair that can be well approximated by a compact mesh to sparse hair or strands that points and meshes cannot reconstruct consistently across time or even for a single frame because individual hair strands are at or below the reconstruction resolution (e.g., Fig. 6). We are interested in approaches that leverage complementary representations such as billboards, curve segments, particles, etc. within a single capture.

Acknowledgements

We would like to thank: David Harnett, Chris Buehler, David Eraker, Leszek Mazur, Kanchan Mitra, Kestutis Patiejunas, Oliver Whyte, Simon Winder, Orest Zborowski, for their many contributions to this project. We also thank: Bill Crow, Lisa Hazen, Tri Le, Spencer Reynolds, David Vos, Jason Waskey, Dana Zimmerman for the quality, quantity, and creativity of our captures. Special thanks to Microsoft Research and affiliates, especially: Philip Chou, Shahram Izadi, Sing Bing Kang, Misha Kazhdan, Charles Loop, Rick Szeliski, Zhengyou Zhang, whose consultations were invaluable. Finally, we would like to thank Dan Casas and Marco Volino for their help in processing their dataset for comparison.

References

4D VIEW SOLUTIONS, 2007. `http://www.4dviews.com`.

AHMED, N., THEOBALT, C., DOBREV, P., AND SEIDEL, H. 2008. Robust fusion of dynamic shape and normal capture for high-quality reconstruction of time-varying geometry. In *Proc. CVPR*.

AHMED, N., THEOBALT, C., ROSSL, C., THRUN, S., AND SEIDEL, H. 2008. Dense correspondence finding for parameterization-free animation reconstruction from video. In *Proc. CVPR*.

ALEXA, M., BEHR, J., COHEN-OR, D., FLEISHMAN, S., LEVIN, D., AND SILVA, C. T. 2001. Point set surfaces. In *Proc. Conf. on Visualization*.

ASPERT, N., SANTA-CRUZ, D., AND EBRAHIMI, T. 2002. MESH: Measuring errors between surfaces using the Hausdorff distance. In *Proc. ICME*.

BLEYER, M., RHEMANN, C., AND ROTHER, C. 2011. PatchMatch stereo - stereo matching with slanted support windows. In *Proc. BMVC*.

BOJSEN-HANSEN, M., LI, H., AND WOJTAN, C. 2012. Tracking surfaces with evolving topology. *ACM Trans. Graph. 31*, 4.

BORSHUKOV, G., PIPONI, D., LARSEN, O., LEWIS, J. P., AND TEMPELAAR-LIETZ, C. 2005. Universal capture – Image-based facial animation for "The Matrix Reloaded". In *ACM SIGGRAPH Courses*.

BUDD, C., HUANG, P., KLAUDINY, M., AND HILTON, A. 2013. Global non-rigid alignment of surface sequences. *Int. J. Comput. Vision 102*, 1-3.

CAMPBELL, N. D. F., VOGIATZIS, G., HERNANDEZ, C., AND CIPOLLA, R. 2008. Using multiple hypotheses to improve depth-maps for multi-view stereo. In *Proc. ECCV*.

CARRANZA, J., THEOBALT, C., MAGNOR, M. A., AND SEIDEL, H.-P. 2003. Free-viewpoint video of human actors. *ACM Trans. Graph. 22*, 3.

CASAS, D., VOLINO, M., COLLOMOSSE, J., AND HILTON, A. 2014. 4D video textures for interactive character appearance. *Comput. Graph. Forum 33*, 2.

CHUANG, M., LUO, L., BROWN, B., RUSINKIEWICZ, S., AND KAZHDAN, M. 2009. Estimating the Laplace-Beltrami operator by restricting 3D functions. *Symposium on Geometry Processing*.

DE AGUIAR, E., STOLL, C., THEOBALT, C., AHMED, N., SEIDEL, H.-P., AND THRUN, S. 2008. Performance capture from sparse multi-view video. *ACM Trans. Graph. 27*, 3.

DOUBLEME, 2014. `https://www.doubleme.me`.

ERICKSON, J., AND WHITTLESEY, K. 2005. Greedy optimal homotopy and homology generators. In *Proc. ACM-SIAM Symposium on Discrete algorithms*.

FRANCO, J., LAPIERRE, M., AND BOYER, E. 2006. Visual shapes of silhouette sets. In *Proc. Intl. Symp. 3D Data Processing, Visualization and Transmission*.

FREED, 2014. `http://replay-technologies.com`.

FURUKAWA, Y., AND PONCE, J. 2010. Accurate, dense, and robust multiview stereopsis. *IEEE PAMI 32*, 8.

GAL, R., WEXLER, Y., OFEK, E., HOPPE, H., AND COHEN-OR, D. 2010. Seamless montage for texturing models. *Comput. Graph. Forum 29*, 2.

GALL, J., STOLL, C., AGUIAR, E. D., THEOBALT, C., ROSENHAHN, B., AND PETER SEIDEL, H. 2009. Motion capture using joint skeleton tracking and surface estimation. In *Proc. CVPR*.

GARLAND, M., AND HECKBERT, P. S. 1997. Surface simplification using quadric error metrics. In *ACM SIGGRAPH*.

GOESELE, M., CURLESS, B., AND SEITZ, S. M. 2006. Multi-view stereo revisited. In *Proc. CVPR*.

GOLDLUECKE, B., AND MAGNOR, M. 2004. Space-time isosurface evolution for temporally coherent 3D reconstruction. In *Proc. CVPR*.

GOLOMB, S. 1966. Run-length encodings (corresp.). *IEEE Transactions on Information Theory 12*, 3.

GUENNEBAUD, G., JACOB, B., ET AL., 2010. Eigen v3. http://eigen.tuxfamily.org.

GUSKOV, I., AND WOOD, Z. J. 2001. Topological noise removal. In *Proc. Graphics Interface*.

HERNANDEZ, C., AND SCHMITT, F. 2004. Silhouette and stereo fusion for 3D object modeling. *Computer Vision and Image Understanding 96*, 3.

HIEP, V. H., KERIVEN, R., LABATUT, P., AND PONS, J.-P. 2009. Towards high-resolution large-scale multi-view stereo. In *Proc. CVPR*.

HU, X., AND MORDOHAI, P. 2012. A quantitative evaluation of confidence measures for stereo vision. *IEEE PAMI 34*, 11.

HUANG, C.-H., BOYER, E., NAVAB, N., AND ILIC, S. 2014. Human shape and pose tracking using keyframes. In *Proc. CVPR*.

ISO/IEC 23009-1, 2014. Information technology – dynamic adaptive streaming over HTTP (DASH) – Part 1: Media presentation description and segment formats.

KANADE, T., RANDER, P., AND NARAYANAN, P. J. 1997. Virtualized reality: Constructing virtual worlds from real scenes. *IEEE Multimedia 4*, 1.

KAZHDAN, M., AND HOPPE, H. 2013. Screened Poisson surface reconstruction. *ACM Trans. Graph. 32*, 3.

KAZHDAN, M., BOLITHO, M., AND HOPPE, H. 2006. Poisson surface reconstruction. In *Symposium on Geometry Processing*.

KLAUDINY, M., BUDD, C., AND HILTON, A. 2012. Towards optimal non-rigid surface tracking. In *Proc. ECCV*.

LABATUT, P., PONS, J.-P., AND KERIVEN, R. 2007. Efficient multi-view reconstruction of large-scale scenes using interest points, delaunay triangulation and graph cuts. In *Proc. ICCV*.

LEE, C. H., VARSHNEY, A., AND JACOBS, D. W. 2005. Mesh saliency. *ACM Trans. Graph. 24*, 3.

LEMPITSKY, V. S., AND IVANOV, D. V. 2007. Seamless mosaicing of image-based texture maps. In *Proc. CVPR*.

LETOUZEY, A., AND BOYER, E. 2012. Progressive shape models. In *Proc. CVPR*.

LI, H., ADAMS, B., GUIBAS, L. J., AND PAULY, M. 2009. Robust single-view geometry and motion reconstruction. *ACM Trans. Graph. 28*, 5.

LINDSTROM, P., AND TURK, G. 2000. Image-driven simplification. *ACM Trans. Graph. 19*, 3.

LIU, Y., DAI, Q., AND XU, W. 2010. A point-cloud-based multiview stereo algorithm for free-viewpoint video. *IEEE TVCG*.

MATUSIK, W., BUEHLER, C., RASKAR, R., GORTLER, S. J., AND MCMILLAN, L. 2000. Image-based visual hulls. In *ACM SIGGRAPH*.

MICROSOFT, 2011. UVAtlas. `http://uvatlas.codeplex.com`.

MOEZZI, S., TAI, L.-C., AND GERARD, P. 1997. Virtual view generation for 3D digital video. *IEEE Multimedia 4*, 1.

NARAYANAN, P., RANDER, P., AND KANADE, T. 1998. Constructing virtual worlds using dense stereo. In *Proc. ICCV*.

SHAN, Q., CURLESS, B., FURUKAWA, Y., HERNANDEZ, C., AND SEITZ, S. M. 2014. Occluding contours for multi-view stereo. In *Proc. ECCV*.

SINHA, S. N., AND POLLEFEYS, M. 2005. Multi-view reconstruction using photo-consistency and exact silhouette constraints: a maximum-flow formulation. In *Proc. ICCV*.

SONG, P., WU, X., AND WANG, M. Y. 2010. Volumetric stereo and silhouette fusion for image-based modeling. *The Visual Computer 26*, 12.

STARCK, J., AND HILTON, A. 2007. Surface capture for performance-based animation. *IEEE Computer Graphics and Application 27*, 6.

SUMNER, R. W., SCHMID, J., AND PAULY, M. 2007. Embedded deformation for shape manipulation. *ACM Trans. Graph. 26*, 3.

VASA, L., AND SKALA, V. 2007. CoDDyaC: Connectivity Driven Dynamic Mesh Compression. In *Proc. 3DTV*.

VLASIC, D., BARAN, I., MATUSIK, W., AND POPOVIC, J. 2008. Articulated mesh animation from multiview silhouettes. *ACM Trans. Graph. 27*, 3.

VLASIC, D., PEERS, P., BARAN, I., DEBEVEC, P., POPOVIĆ, J., RUSINKIEWICZ, S., AND MATUSIK, W. 2009. Dynamic shape capture using multi-view photometric stereo. *ACM Trans. Graph. 28*, 5.

VOLINO, M., CASAS, D., COLLOMOSSE, J. P., AND HILTON, A. 2014. Optimal representation of multiple view video. In *Proc. BMVC*.

WAND, M., ADAMS, B., OVSJANIKOV, M., BERNER, A., BOKELOH, M., JENKE, P., GUIBAS, L., SEIDEL, H.-P., AND SCHILLING, A. 2009. Efficient reconstruction of nonrigid shape and motion from real-time 3D scanner data. *ACM Trans. Graph. 28*, 2.

WANG, Z., BOVIK, A. C., SHEIKH, H. R., AND SIMONCELLI, E. P. 2004. Image quality assessment: From error visibility to structural similarity. *IEEE Trans. Image Proc. 13*, 4.

WOOD, Z., HOPPE, H., DESBRUN, M., AND SCHRÖDER, P. 2004. Removing excess topology from isosurfaces. *ACM Trans. Graph. 23*, 2.

WU, C., VARANASI, K., LIU, Y., SEIDEL, H.-P., AND THEOBALT, C. 2011. Shading–based dynamic shape refinement from multi-view video under general illumination. In *Proc. ICCV*.

YE, G., LIU, Y., DENG, Y., HASLER, N., JI, X., DAI, Q., AND THEOBALT, C. 2013. Free-viewpoint video of human actors using multiple handheld Kinects. *IEEE Trans. on System, Man & Cybernetics 43*, 5.

YU, F., LUO, H., LU, Z., AND WANG, P. 2010. 3D mesh compression. *Three-Dimensional Model Analysis and Processing*.

ZHOU, Q.-Y., AND KOLTUN, V. 2014. Color map optimization for 3D reconstruction with consumer depth cameras. *ACM Trans. Graph. 33*, 4.

ZITNICK, C. L., KANG, S. B., UYTTENDAELE, M., WINDER, S., AND SZELISKI, R. 2004. High-quality video view interpolation using a layered representation. *ACM Trans. Graph. 23*, 3.

ZOLLHÖFER, M., NIESSNER, M., IZADI, S., REHMANN, C., ZACH, C., FISHER, M., WU, C., FITZGIBBON, A., LOOP, C., THEOBALT, C., AND STAMMINGER, M. 2014. Real-time non-rigid reconstruction using an RGB-D camera. *ACM Trans. Graph. 33*, 4.

Figure 17: *Gallery of example results. (All renderings are from viewpoints that differ from the original stage views.)*