ADAPTING SINGLE-VIEW VIEW SYNTHESIS WITH MULTIPLANE IMAGES FOR 3D VIDEO CHAT

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Anurag Uppuluri

December 2021

COMMITTEE MEMBERSHIP

TITLE:	Adapting Single-View View Synthesis with Multiplane Images for 3D Video Chat
AUTHOR:	Anurag Uppuluri
DATE SUBMITTED:	December 2021
COMMITTEE CHAIR:	Jonathan Ventura, Ph.D. Assistant Professor of Computer Science
COMMITTEE MEMBER:	Zoë Wood, Ph.D. Professor of Computer Science
COMMITTEE MEMBER:	Franz Kurfess, Ph.D. Professor of Computer Science

ABSTRACT

Adapting Single-View View Synthesis with Multiplane Images for 3D Video Chat

Anurag Uppuluri

Activities like one-on-one video chatting and video conferencing with multiple participants are more prevalent than ever today as we continue to tackle the pandemic. Bringing a 3D feel to video chat has always been a hot topic in Vision and Graphics communities. In this thesis, we have employed novel view synthesis in attempting to turn one-on-one video chatting into 3D. We have tuned the learning pipeline of Tucker and Snavely's single-view view synthesis paper [35] — by retraining it on MannequinChallenge dataset [20] — to better predict a layered representation of the scene viewed by either video chat participant at any given time. This intermediate representation of the local light field — called a Multiplane Image (MPI) — may then be used to rerender the scene at an arbitrary viewpoint which, in our case, would match with the head pose of the watcher in the opposite, concurrent video frame. We discuss that our pipeline, when implemented in real-time, would allow both video chat participants to unravel occluded scene content and "peer into" each other's dynamic video scenes to a certain extent. It would enable full parallax up to the baselines of small head rotations and/or translations. It would be similar to a VR headset's ability to determine the position and orientation of the wearer's head in 3D space and render any scene in alignment with this estimated head pose. We have attempted to improve the performance of the retrained model by extending MannequinChallenge with the much larger RealEstate10K dataset [38]. We present a quantitative and qualitative comparison of the model variants and describe our impactful dataset curation process, among other aspects.

ACKNOWLEDGMENTS

Thanks to:

- My advisor, Dr Ventura, and all the other faculty and staff members here at Cal Poly I am fortunate to interact with, faculty and staff at Fresno State, Sathyabama University and all the other schools I am an alumnus of, my family, and my friends for going above and beyond in helping me set sail in and even course through uncharted potential with oft-increasing frequency. It's as they say: you tend to a sapling until it takes firm root and starts bearing fruit; and I feel all these people have been doing all that and more all along, and I can't wait to give back in ways that far surpass my imagination.

- Richard Tucker, for his selfless and infectious enthusiasm in guiding our project.

- Robert Downey Jr. as Tony Stark in Iron Man (2008) saying, "Jarvis, sometimes you gotta run before you can walk", before pushing the limits of the Mark II armor.

- Andrew Guenther, for uploading this template.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

INTRODUCTION

From pertinent work meetings to casual conversations with family and friends, an ever increasing number of people use video chatting/conferencing applications such as FaceTime, Zoom, Google Meet, and Microsoft Teams, to name a few. One way of improving video chat experience is to bring in a feel of 3D by providing alternate views (images or frames) of each viewed scene, rendered at different viewpoints. To fortify the 3D experience each novel view would have to be rendered at the right angle such that it aligns with the viewpoint of the viewer. This would require taking the viewer's transient head pose[1] into account. In this way, we can seek to get an ideal feel of 3D by, essentially, simulating what happens when we move our heads. When we move our heads, what we see in terms of the extent of the foreground, the background, and everything in between changes based on the change in our head poses. These changes need to be reflected in rendered novel views. In this work, we attempt to emulate 3D video chatting via targeted high-quality novel view synthesis.

1.1 Motivation

Currently, synthesis of high-quality novel views — the basis of Image-Based Rendering (IBR) systems — is difficult to achieve end-to-end without some form of an intermediate representation of the structure (such as 3D world points) of the scene depicted by the given image(s). For instance, Google's Project Starline (Figure 1.1)

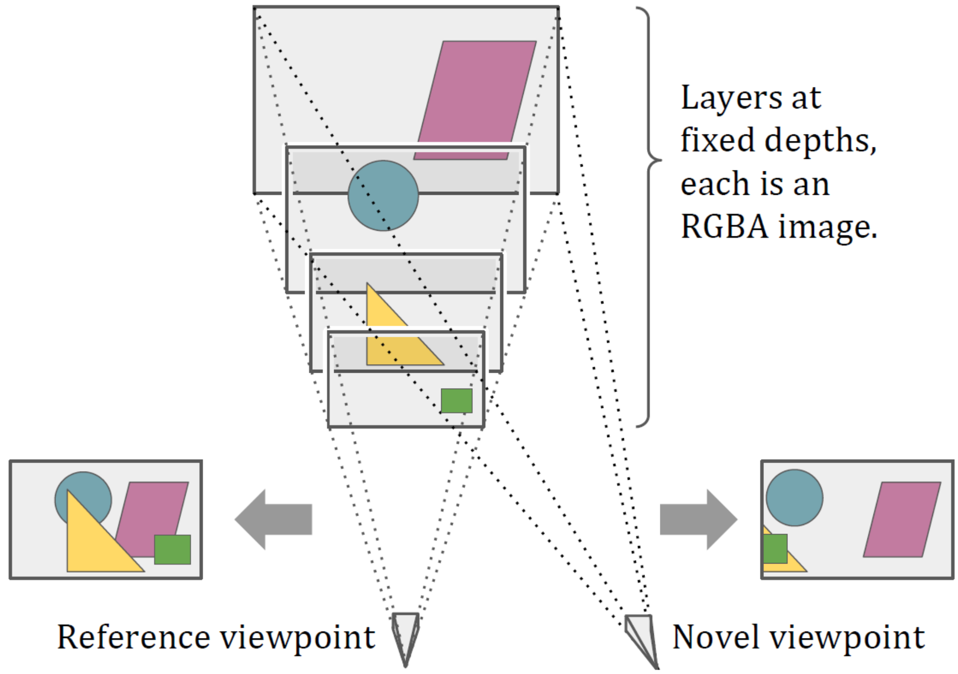
[1]Pose refers to the combination of any object's position and orientation in 3D world space, including cameras. In contrast, we only use the *orientation* of the viewer's head in the world as the head pose for viewed scenes to be rerendered at.

Project Starline uses a groundbreaking light field rendering system that is projected to improve glasses-free 3D / automultiscopic video chat experience by leaps and bounds.

Figure 1.1: Google's Project Starline [8]

uses a dense 3D representation to go from known views to novel views. One impressive variation of such an intermediate representation is called a Multiplane Image (MPI) — first reintroduced in Zhou et al. [38] (Figure 1.2). It is a volumetric representation that reprojects 2D points making up an image onto multiple 2D planes situated one behind the other at successive depths along the z-axis, according to the computed depth/disparity value(s)[2] at each point to be mapped. MPI planes are parallel to each other and also to a reference coordinate frame centered at a reference camera/viewpoint looking down positive z-axis (assuming a left-handed coordinate system). The reference camera can be that of the image itself or of a different view of the scene captured by the image. An MPI can thus be formulated as a set of RGBA layers $\{(C_1,\ \alpha_1),\ (C_2,\ \alpha_2),\ \ldots,\ (C_D,\ \alpha_D)\}$, where C_i refers to the RGB map of each layer $(C_i,\ \alpha_i)$ and α_i is the alpha map. D is the total number of depth planes used

[2]Since pixels can be smaller than or equal in size to points, there can be multiple RGBA and depth/disparity values corresponding to the multiple pixels/sub-pixels that might make up a 2D point on an image.

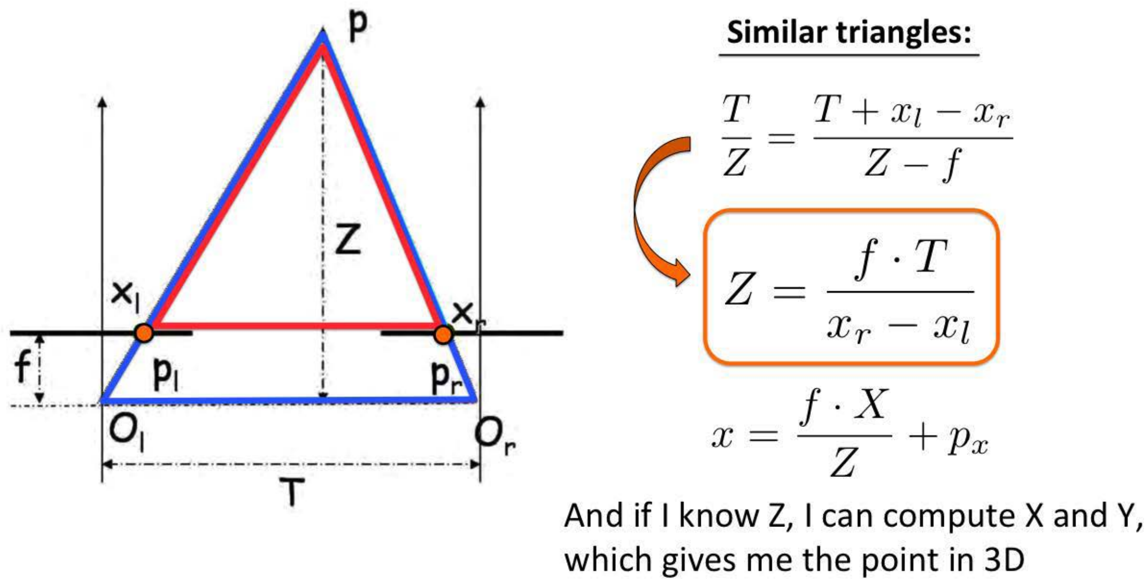


A given image is reprojected onto multiple fronto-parallel MPI planes within the view frustum of a common reference viewpoint that may or may not match with the given image's viewpoint. A novel image is synthesized by alpha-blending all layers of the MPI in back-to-front order. The layers are numbered in back-to-front order as well, with the farthest layer 1 being at depth d_1 and the nearest layer D being at depth d_D.

Figure 1.2: The Volumetric/Layered MPI Representation [38]

in the MPI. To render from an MPI one simply needs to alpha-blend all layers in back-to-front order, as explained in section 2.1. One popular instance of such depth planes used in an MPI is a set of 32 planes positioned at equidistant disparity, with the near and far planes being at 1m and 100m in 3D world space, respectively. Since disparity is inversely proportional to depth, the points on the nearer MPI planes are closer to the reference camera than the ones on the farther planes but they have greater disparity values associated with them than the farther ones.

Disparity refers the number of pixels that each point on a image shifts over by in any of its warped/transformed counterparts that can relate to it via a homography



In this birds-eye view down the y-axis, O_l and O_r are the optical/camera centers of the left and right images of a stereo pair; p_l and p_r are the 2D projections of the same 3D world point p onto the stereo pair; x_l and x_r are the x-coordinates of these 2D image points (y-coordinates are the same for corresponding points on a stereo pair); f is the common focal length of the stereo cameras; T is the horizontal translation or baseline of the stereo cameras; and Z is the perpendicular depth of p from the common reference coordinate frame of the stereo cameras and is inversely proportional to the disparity, $x_r - x_l$. Taking similar birds-eye perspectives down the z and x axes, we can compute the x and y coordinates of the 3D point as well.

Figure 1.3: Disparity used in Triangulating 3D Points [11]

(projective transform function). Disparity is required for triangulating the depth(s) at each point on the image with respect to its warped version(s). Triangulating depth and estimating the 3D scene structure is easier when two or more of the scene's images are subjected to either stereo or multi-view image rectification, respectively. Such image rectification procedures typically involve rotating and shifting the optical centers of each image so they became collinear and scaling — adjusting the focal lengths of the cameras of — the images themselves so they become coplanar. Rectified image sets are characterized by point displacements only in the horizontal/row-wise x direction. Properties of similar triangles can then be applied to the rectified images to get at the z-coordinate of each 3D world scene point most agreed upon by all the images containing the point's projections, after accounting for reprojection mismatches. Figure 1.3 shows the triangulation process for a stereo pair. This is akin to how the human visual system (including the eyes, their ganglia, the dorsal and ventral streams of the brain, and the visual cortex) is able to triangulate depth from binocular vision. The brain is backed by prior knowledge, heuristics, and biases (made apparent by optical illusions) that it is able to use to infer depth to some degree of approximation even with one eye closed. Since Artificial Neural Networks (ANNs) are basically trying to replicate and someday even surpass the workings of the human brain, we are actually trying to fill in for this prior knowledge acquired by the brain when we provide ANNs with copious amounts of data to learn from and devise their own heuristics out of. Therefore, we may only generate an MPI for an image when we are provided either with one or more shifted and/or rotated reprojections of the scene in the image or with the homographies for generating each of these transformed images from the original image. Otherwise, we would need to be supplied the sparse/dense 3D point cloud of the image's scene itself. In any case, the viewpoint parameters of all views involved are required as well.

1.2 Contribution

To give a gist of our work, it began by attempting to retrain Tucker and Snavely's state-of-the-art end-to-end fully-convolutional single-view view synthesis with MPIs CNN [35] on the MannequinChallenge dataset. We hypothesized — as was also hinted at in the paper — that such retraining would be sufficient to generate high quality MPIs of scenes involving close-up shots of people, typical of video chat settings. The original model is able to do the same for real estate scenes. We then went on to compare the inference results of this primary model variant with those of another variant trained on the MannequinChallenge dataset extended by the RealEstate10K dataset, taking the pretrained Tucker and Snavely model as baseline. This was so we could determine the best variant to apply to the domain of 3D video chat. Such application was conceived to be by way of a two-way rendering of appropriate novel views of concurrent dynamic scenes viewed by one-on-one video chat participants in both directions simultaneously. In the two-way pipeline, a novel view of a video frame would be rendered every time a change in head pose is detected in the participant in the opposite frame. To our knowledge, MPIs have not been used in 3D video chat so far. We publish the code used to fill in the missing parts of Tucker and Snavely's publicly available training and testing pipelines, along with highlights regarding curating and taking advantage of both datasets for view synthesis in video chats.

Chapter 2

RELATED WORK AND BACKGROUND

In this thesis, we have not created novel models or datasets but have rather curated preexisting datasets and retrained a state-of-the-art CNN. Data curation has been an essential part of our work as the datasets' YouTube videos are subject to modifications over time. These modifications are in terms of the videos being taken down from YouTube or the required 1280×720 pixel (720p) resolution versions of them becoming unavailable, etc. The curation process included action items like downloading and training only on 720p versions of the datasets' videos so as to minimize the chances of running into training errors, etc., as explained in section 3.2. As for simulating the 3D video chat experience itself, we linked-up the API of OpenFace 2.2 [7] — a preexisting head pose estimation model — to the MPI inference procedure so the MPI inference may generate novel views rendered at the head pose evaluated by OpenFace 2.2, as explained in section 3.3.

This chapter explores related work in two areas: MPIs and 3D video chat, while providing clarifications on background concepts along the way. The research papers of particular interest to us as far as the MPI component of our work is concerned are 2018's Zhou et al. [38] and 2020's Tucker and Snavely [35], which we consider to be our base papers. This is because we have attempted to adapt and apply Tucker and Snavely's work to the purposes of video chatting and their work directly draws from Zhou et al. We have also sought to differentiate 2016's DeepStereo [13] and Kalantari et al. [17] from Zhou et al. as it, in turn, is inspired by them and surpasses them performance-wise. As for progress in the field of 3D video chat, we have mentioned the state-of-the-art 3D video chat system: Google's Project Starline; among other projects.

2.1 Learning MPIs

Some of the major challenges in high-quality novel view synthesis include synthesizing pixels occluded in one or more of the provided views, disentangling and localizing ambiguous pixels at/near the boundaries of foreground and background objects, localizing pixels at transparent, translucent, reflective, or texture-less surfaces, etc. Moreover, whereas interpolating novel views at desired viewpoints lying within the convex hull of given viewpoints is easier to achieve than extrapolating significantly beyond the baselines (distances between camera centers) of input views, these challenges can emerge in either case. So far, it has been found that learning view synthesis is the way to go for tackling them all in one shot.

Before the Machine Learning (ML) boom in Computer Vision (CV) circles in 2012, convolutional filters had to be handcrafted and dexterously layered one atop the other before input views could be subjected to them and various types of features could be extracted in the process of rendering novel views. All the aforementioned view synthesis challenges had to be manually targeted by way of devising various combinations of these filters. This meant a high proportion of artifacts induced in novel views could be left unresolved. Since the time that the efficacy of CNNs in CV was proven by Krizhevsky, Sutskever, and Hinton [19] in 2012, to the delight of the CV community, the need to handcraft filters was obviated by ML models that learned to design all required convolutional filters on their own in their various hidden layers. These self-taught filters are defined by the weights and biases in each hidden layer neuron. The weights and biases constantly improve during training and the convolutional filters defined by them are specific to the datasets they are trained on, with some degree of generalizability to other datasets. If trained well under effective hyperparamter tuning, learned filters can evolve to surpass manual filters in addressing occlusion, transparency, reflection, and other image synthesis challenges.

View synthesis can lend itself to being a semi-well-posed to well-posed learning problem where two or more images of a scene can be shot and an ML model can be exposed to one of more of these images while being expected to predict one or more of the remaining views that have been withheld from it. The quantitative difference between the corresponding predicted and withheld (as ground truth) views will then be the loss that the ML training seeks to minimize. Since end-to-end view synthesis without an intermediate representation is still largely unrealized, the popular way to synthesize novel views is to learn an intermediate representation of the scene common to the input views and use this intermediate representation to render novel views. The MPI intermediate representation has proven to be one the most effective representations for this purpose with implications as significant as real-time high-quality spatially-consistent view synthesis.

2.1.1 Seminal Work

The roots of the MPI representation may be traced back to seminal papers such as 1996's Collins [10], 1998's Shade et al. [32], and 1999's Szeliski and Golland [34]. Collins perfected the concept of Plane Sweep Volumes (PSVs), Shade et al. introduced layered depth images, and Szeliski and Golland introduced the actual MPI representation itself. These groundbreaking techniques have also been compared in Scharstein and Szeliski [29].

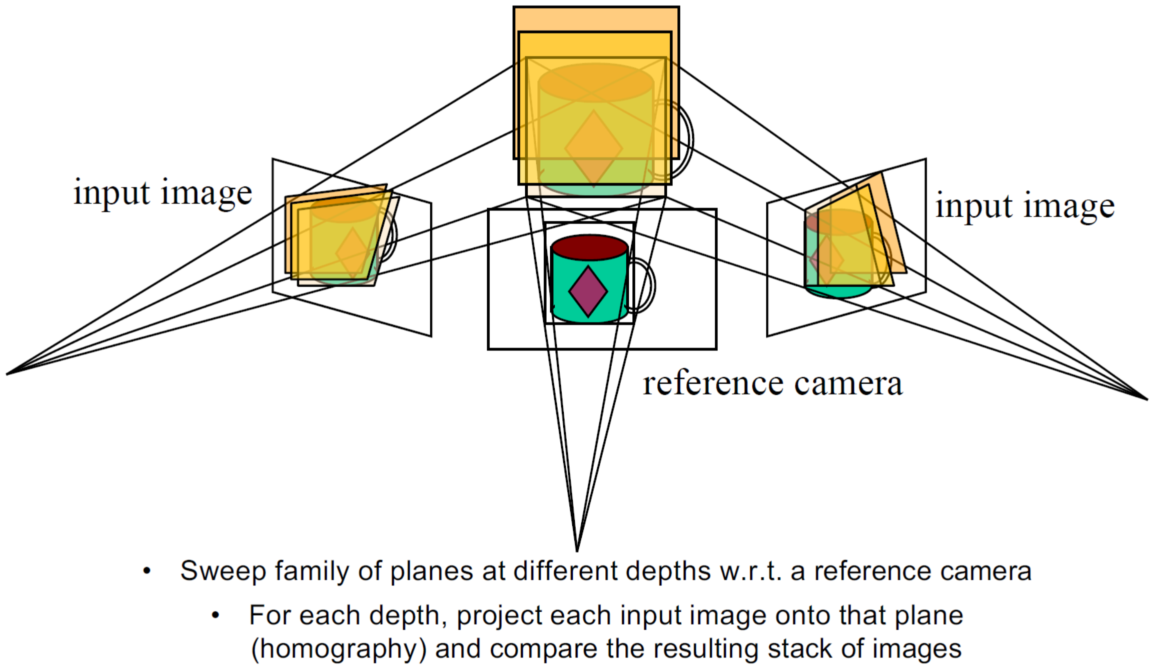


Figure 2.1: Plane Sweep Volume Representation [4]

Collins [10] applied the PSV representation to the problem of reconstructing the 3D scene from multiple views while simultaneously performing feature matching across all views sharing common features. Feature matching is the process of matching corresponding "features of interest" characterized by their repeatability across multiple views of the same world scene. Examples include keypoints, corners, edges, objects, etc. Matched views can be rectified and used for triangulating depth, etc., as mentioned in section 1.1. In the author's implementation, instead of going for a resource-intensive 3D representation that would require splitting the entire 3D scene space into voxels and reprojecting[1] all feature points from all views in such manner that the reprojected light rays passed through this uniformly partitioned space, he sampled the 3D scene space at various 2D planes along the depth (z) axis, as if capturing just one 2D plane sweeping though it at various instants in time. He partitioned the sweeping plane into cells and allowed each reprojected light ray to

[1]projecting to a target plane by unapplying and applying the homographies needed to project to the source and target planes, respectively, while accounting for surface normals, plane offsets, camera rotations and translations, etc., as described in subsection 2.1.3

vote for a group of cells that fell within a certain radius of the point of intersection of the light ray with the plane. This accounts for the fact that rays from corresponding feature points across all views may not converge most of the time due to reprojection errors. He then chose the z-coordinate of the sampled plane containing the cell with the maximum votes for a feature point to be the z-coordinate of the feature point in the world scene. The x and y world coordinates would be defined by this winning cell. The victor cell would also determine the 2D feature point correspondences simultaneously just by virtue of the converging rays being retraced to their respective originating views. PSVs, in their various reimplemented forms, have become almost synonymous of layered volumetric representations these days (Figure 2.1).

Shade et al.'s [32] Layered Depth Image (LDI) scene representation is similar to MPI scene representation in that both MPI and LDI consist of a series of fronto-parallel planes facing a chosen reference viewpoint and placed at varying depths from it. These planes contain the RGB information of the original pixels of the scene's image(s), segregated according to depth. MPI differs from LDI (and PSV) in that it has alpha masking effects at each layer, as it is generated with alpha transparency maps for each layer. Also, MPIs have fixed depths for each layer as opposed to the variable layer depths of LDIs (and PSVs). But in both cases, by virtue of layering, users are able to experience a simulation of what happens when they move their heads while looking at a scene in the world — they are able to look around foreground objects that occlude background ones.

Szeliski and Golland [34] first introduced the MPI representation for purposes of stereo matching with simultaneous RGBA estimation at each matched pixel. Stereo matching, otherwise called disparity mapping, uses feature matching techniques such as SIFT[2] in pixel-and-sub-pixel-wise disparity estimation for 3D scene reconstruction

[2]Scale-Invariant Feature Transform [21]

from rectified stereo images. The authors' framework was the first to extract high-accuracy depth, color, and transparency maps for several images at a time, operating even at sub-pixel levels. They were able to enforce sub-pixel accuracy and perform effective matte separation of foreground and background elements despite the usual 3D vision challenges such as occlusions, etc., because they came as close to modern ML reimplementations as possible. They implemented various loss functions such as a pixel-wise weighted photometric L_1 norm between the input and reprojected images, a per-pixel smoothness constraint on the RGBA values allowed in the reprojected images, etc. They then performed an iterative refinement of the estimated RGBA values with the help of a gradient[3] descent algorithm designed to optimize a combination of all these losses, but sans the explosive power of neural networks.

2.1.2 Influential Work

DeepStereo [13] was the first to apply CNNs in an end-to-end manner to novel view synthesis from diverse collections of indoor and outdoor imagery in the wild, given the availability of camera parameters[4] for each input image. Their paper describes why it would be unwise to expect a typical present-day CNN to synthesize any ground-truth target image without being provided with the pose of the view as well — the network would needlessly be learning epipolar geometry itself! Epipolar geometry — the geometry of binocular and multi-view stereo vision — gives us the epipolar constraint $x'^T F x = 0$ between all corresponding points x and x' on a stereo pair. Here, F is called the fundamental matrix and is derived from the intrinsic and extrinsic parameters of the stereo cameras involved. To circumvent such an indeterminable and expensive pixel-to-pixel training scenario, the authors had PSVs (Figure 2.1) come

[3]vector of partial derivatives of the function(s) to be optimized

[4]camera intrinscics such as focal length and principal point and camera extrinsics/pose such as position and orientation

to the rescue. They supplied all input views required to synthesize a target view as separate PSVs to their network. Each input plane sweep would contain all pixels of the respective input view reprojected onto a chosen number of planes at chosen depths in the usual "stack of acetates" manner, with the planes all having their viewpoints match with the target view's. The plane that each RGB pixel gets reprojected onto will also determine the availability of the pixel (as alpha values ranging from 0 to 1) to the surrounding voxels of the PSV. The plane sweep of each input view has the pose information of the view implicitly encoded in it just by virtue of its construction. Moreover, the plane sweeps of all input views of the same scene trivially enforce the epipolar constraint as all matching pixels across these originating input views may be located in the same depth-wise column of each plane sweep. Each of these depth-wise columns may then be computed upon by the network independently of other columns, in producing the corresponding synthesized target pixel. The network learns to predict the best weight and color for each reprojected pixel on all input planes, so it may perform a weighted summation of these estimated pixel colors and obtain a final predicted target pixel color. Such averaging has a smoothing effect over the color values of the synthesized target image. The error signal that is iteratively minimized by the training is given by the pixel-wise L_1 (absolute difference) loss between the actual target color $C^t_{i,j}$ and the synthesized target color $C^s_{i,j}$ at each pixel (i, j):

$$\mathcal{L} = \sum_{i,j} |C^t_{i,j} - C^s_{i,j}|$$

Kalantari et al.'s [17] model learns to interpolate novel views in the 8x8 central view grid of a Lytro camera containing a microlens array. It was the state-of-the-art learning-based view synthesis model prior to Zhou et al.'s [38] *stereo magnification*

MPI model. It is composed of disparity and color predictor components in the form of simple 4-layered sequential CNNs. The training signal it optimizes is given by the L_2 (squared difference) pixel reconstruction loss between each pair of original and interpolated target views.

Both DeepStereo and Kalantari et al. are unable to train on training images in their entirety. Instead, they extract patches of training images for their models to train on. This is because, unlike how Zhou et al.'s model is designed to predict a global scene representation once for a pair of views belonging to the same scene and render many novel views with it at near-real-time speeds, the former models are designed to predict each novel view in an end-to-end fashion independently of other novel views and so have to rerun their prediction pipelines every time, making novel view synthesis prohibitively slow for high-resolution and real-time applications. Moreover, when rendering nearby views, the former methods produce much more artifact-ridden, spatially-incoherent views compared to the views inferred by Zhou et al. What Zhou et al. has going for it in these scenarios is an implicit smoothness prior imposed by the common scene representation over the color and depth values being inferred for each synthesized nearby view.

What also comes close to the MPI representation is the layered representation of Penner and Zhang [25]. But then again, in all these prior methods, a unique scene representation is predicted in the reference coordinate frame of each target view to be rerendered, negatively impacting view synthesis efficiency. Other innovative MPI-related papers released subsequently to Zhou et al. and leading up to Tucker and Snavely's [35] *single-view* MPIs are 2019's Srinivasan et al. [33], Mildenhall et al. [23], and DeepView [12]. Srinivasan et al. improved the quality and increased the disparity and baseline ranges of predicted MPIs and rendered views, by bringing in a 3D CNN architecture, training on random-resolution views, and introducing an optical flow constraint over the appearance of occluded content in rendered views. Mildenhall et al.'s model converts an grid of irregularly sampled views into MPIs, i.e., mini-light-field representations, and blends such nearby local light fields to render novel views. They were able to establish a minimum density of sampled views required for robust rendering, which turned out to be 4000× less than the Nyquist frequency required to prevent aliasing. DeepView [12] replaced the update step[5] of the network's gradient descent algorithm with a CNN that learns the various gradient descent parameters instead. As a consequence, the network takes much larger strides along the direction of optimization and converges much sooner and with more accuracy than a network using standard gradient descent. However, these methods do not tackle the monocular-image approach for generating MPIs.

2.1.3 Base Papers

Zhou et al. [38] was the first to implement view extrapolation to significantly larger baselines (up to 8× input baselines) than prior work — a process they call stereo magnification. They use stereo pairs to learn an MPI (Figure 1.2) prediction network in the following manner:

- The camera parameters $c_1 = (p_1,\ k_1)$ and $c_2 = (p_2,\ k_2)$ of the stereo pair, $(I_1,\ I_2)$, are also needed for the prediction process, along with the target image I_t and its parameters c_t. Here, p's and k's refer to the camera extrinsics and intrinsics of the respective images.
- The viewpoint of one image of the stereo pair, I_1, is used as the reference viewpoint for the MPI to be predicted at. Hence, p_1 would be the identity pose $[I|\boldsymbol{0}]$.

[5]involving step size and other parameters such as priors/biases

Figure 2.2: Inferred MPI [38]

- The goal is to learn a network that generates an MPI representation (Figure 2.2) with inputs ($I_1,\ I_2,\ c_1,\ c_2$) such that when the MPI is rendered at viewpoint c_t, it would produce the target image I_t.

- As demonstrated by DeepStereo [13], an effective way to encode pose information for training is via a PSV (Figure 2.1). Hence, the input to their customized encoder-decoder type network includes a PSV version of I_2 ($\hat{I}_2$) with the planes all reprojected into the output MPI's viewpoint, c_1, and with the entire plane sweep concatenated internally and with an unaltered I_1 along the three color channels. The depth planes of $\hat{I}_2$ are also chosen to coincide with the ones of the output MPI.

- The 3D structure of the scene depicted by I_1 and I_2 is automatically learnt by the network by merely being able to compare I_1 with each of the reprojected images of I_2 in the input stack ($\hat{I}_2^1,\ \hat{I}_2^2,\ \ldots,\ \hat{I}_2^D,\ I_1$), where D is the total number of MPI depth planes. The depth at each pixel of any known or novel view of the scene must be the depth of the plane where I_1 and $\hat{I}_2$ concur.

- In order to reduce resource consumption due to network over-parameterization, the network's initial outputs do not consist of separate RGBA maps for each MPI layer but rather just a "background" image intended to capture pixels occluded in I_1 and a set of color blending weight maps and alpha maps for each MPI layer.

- The actual RGB values in each layer, C_i, are then easily computed by taking the per-pixel weighted average of I_1 and the predicted background image $\hat{I}_b$:

$$C_i = w_i \odot I_1 + (1 - w_i) \odot \hat{I}_b$$

  Here, $\odot$ is the Hadamard product and w_i refers to the RGB blending weights from the initial network output, specific to MPI layer i.

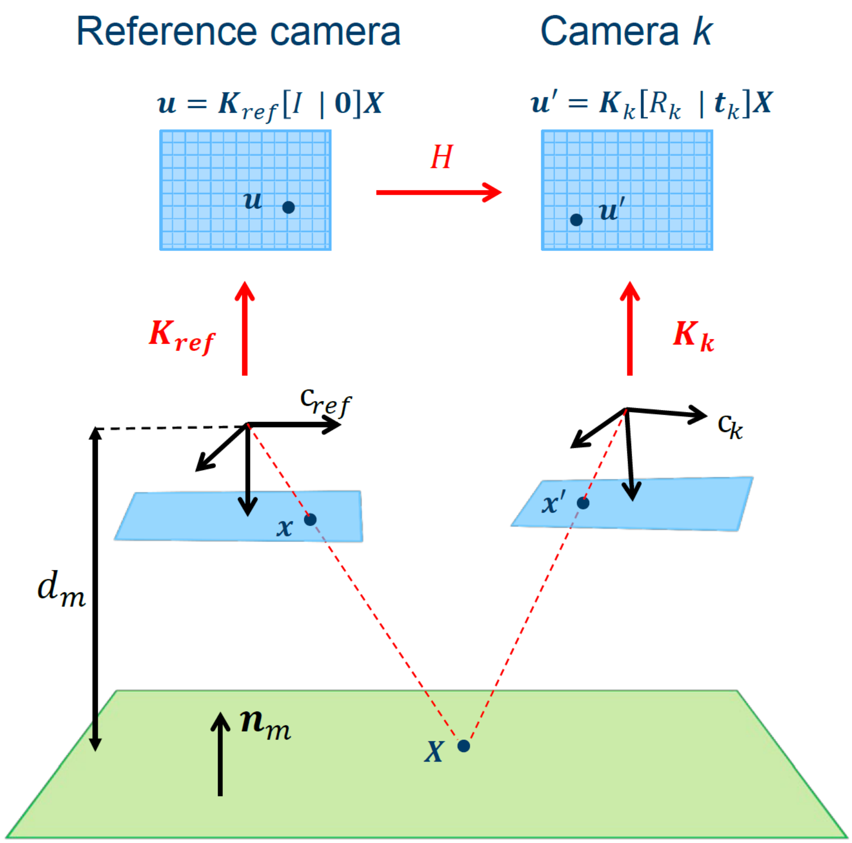
- $\hat{I}_b$ need not itself be a natural image as the network can selectively and softly blend each $\hat{I}_b$ pixel with I_1, based on respective layer α's and w's. Intuitively, I_1's contribution would be more in foreground layers than in the background ones; and conversely for $\hat{I}_b$.

The rest of the training pipeline consists of the rendering of the MPI at the target viewpoint, c_t, and the gradient descent algorithm involving a VGG perceptual (similar to LPIPS [37]) loss function between the rendered view and ground-truth target view. The perceptual loss is proven to be more robust than unmodified pixel reconstruction losses such as L_1 and L_2 norms. Adam gradient descent algorithm is used (similarly to Tucker and Snavely [35]) to optimize this loss. Adam [18] is better than regular stochastic gradient descent but is still not superior to DeepView's [12] implementation of learned gradient descent. Rendering an MPI first involves warping each

RGBA MPI layer onto the target camera's image plane using the standard inverse homography or reprojection operation [16], as illustrated in figure 2.3. But, anticipating usual reprojection mismatches, they resample each pixel to be warped by bilinear interpolation with respective four-grid neighbors. These rerendered MPI layers are then alpha-composited in back-to-front order to get the final predicted target view. All elements of the rendering process are differentiable.

Zhou et al.'s methods are ingenious in a number of ways. They trained their model to predict novel views at varying distances from input views so as not to overfit to predicting only up to a limited number of baselines. They used assorted but apt convolutional layers such as dilated convolutions to bring back larger scene contexts at lower computational costs and fractionally-strided convolutions [27] with skip connections [6] from preceding layers to capture even the finer texture details. The use of VGG perceptual loss allowed them to retain these intricate micro textures together with macro object geometries in synthesized views. Also commendable is their meticulous RealEstate10K dataset creation process which was continued by Tucker and Snavely [35] in bringing the dataset to it's current state [38]. Knowing that state-of-the-art Structure from Motion (SfM) and bundle adjustment[6] algorithms such as COLMAP [30, 31] are not yet fully optimized for camera tracking in videos, they first subject candidate real estate YouTube videos to Simultaneous Localization And Mapping (SLAM) techniques such as ORB-SLAM2 [24] to obtain initial camera parameter estimates for all consecutive frames tracked. Consecutive, here, implies that each tracked frame's viewpoint is no farther than a certain percentage of the average of its two neighboring viewpoints. This process naturally breaks a video apart into clips with smoother camera motion. They then process all video clips obtained this

[6]initial scene reconstruction, camera calibration (including field of view estimation), and pose estimation for a candidate pair of scene views, followed by simultaneous iterative refinement of the 3D scene structure and all estimated camera parameters, using each additional view of the scene, as well, for feature matching



Here, the 3D point $\boldsymbol{X}$ on the MPI plane in the world is the *homogeneous* version (determined up to scale) of its projection $\boldsymbol{x}$ on the reference camera's image plane in camera coordinates, i.e., with the camera's image plane centered at the camera center, c_{ref}. More precisely, $\boldsymbol{X} = [X, Y, d_m]^T \sim \tilde{\boldsymbol{x}} = [X/d_m, Y/d_m, 1]$. This is because all MPI world planes are fronto-parallel to the reference camera and their equations can be given by $\boldsymbol{n_m} \cdot \tilde{\boldsymbol{x}} + a = 0$, where $\boldsymbol{n_m} = [0, 0, 1]$ is the plane normal and $a = -d_m$ is the plane offset from c_{ref}. The projection $\boldsymbol{u}$ on the reference camera's image plane in regular image coordinates is attained by applying reference camera intrinsics $\boldsymbol{K_{ref}}$ to $\boldsymbol{x}$. Since the MPI is not necessarily fronto-parallel to the target camera c_k, $\tilde{\boldsymbol{x}'}$ need not be $[X/d_m, Y/d_m, 1]$ even though $\boldsymbol{X} \sim \tilde{\boldsymbol{x}'}$ as well. $\boldsymbol{u'}$ and $\boldsymbol{K_k}$ similarly belong to the target camera, as does target camera pose (relative to reference camera) $[R_k|\boldsymbol{t_k}]$. The world plane *induces* the homography $H = \boldsymbol{K_k}(R_k - \boldsymbol{t_k}\boldsymbol{n_m}^T/a)\boldsymbol{K_{ref}}^{-1}$ between the image planes of c_{ref} and c_k, so we can go from $\boldsymbol{u}$ to $\boldsymbol{u'}$. To go from $\boldsymbol{u'}$ back to $\boldsymbol{u}$, we'd use H^{-1} [40].

Figure 2.3: Standard Inverse Homography or Reprojection [15]

way with COLMAP to get a sparse 3D point cloud reconstruction of the scene in each clip and a refined set of camera parameter estimates for all frames. As a final step, they *scale-normalize* each subsequence and its reconstructed camera parameters and 3D points in one shot by scaling the point cloud up or down so the nearest set of points is at a fixed distance from the cameras. Points clouds are discarded by Zhou et al. after scale-normalizing the dataset whereas they are used by Tucker and Snavely [35] to "scale-normalize", effectively, their entire single-view training process itself, for they don't have the luxury of inferring parameter and scene scale from more than one view at a time like how Zhou et al. does. SfM involves the estimation of the (generally sparse) 3D structure of a static scene from the multiple (usually unstructured) views of a (often uncalibrated) camera moving around the scene, accompanied by the simultaneous estimation of respective camera parameters. It is essentially a more generic version of Multi-View Stereo (MVS), which itself is an extension of stereo matching and requires known camera parameters to reconstruct (mostly) dense 3D points clouds. COLMAP is capable of both SfM and MVS. Both SfM and MVS can utilize bundle adjustment similarly to SLAM from the Robotics community. SLAM doesn't stop at bundle adjustment but rather proceeds to map out the entire terrain encountered by a robot by making connections between camera trajectories, viewed scenes, etc. [2].

Zhou et al. made some major observations in their various ablation studies. They found that their model trained better on their preferred MPI prediction format consisting of a predicted background image that is blended with the reference image (taken as foreground) using a set of predicted color blending weights, to form each layer of the MPI. This format beat other, more-expressive formats such as ones with an additional predicted foreground or with fully predicted MPI layers. They speculate that the network's somewhat diminished performance with the latter formats could be because of network over-parameterization, more utilization of synthesized layers rather than the original reference image, and perhaps even because of lesser camera movement between the synthesized layers for the network to efficiently learn depth complexity out of. Moreover, they were able to verify that the greater the number of MPI planes used, the higher would be the model's training performance and the quality of synthesized views. Their model presents considerable scope for improvement when it comes to accurately localizing and fixing the depths of multiple overlapping fine textures, avoiding "stacks of cards" edges in synthesized views when the disparity between the neighboring layers of an MPI exceeds one pixel, etc.

Tucker and Snavely [35] was the first to implement learning-based single-view view synthesis on videos in the wild. It is fascinating to see how they were able to achieve efficient single-view view synthesis — an objective coveted by Vision and Graphics communities. Moreover, there are numerous other perks to their model. It produces byproduct disparity maps that can be used in imposing a smoothness prior over synthesized viefws, in computing a global scale factor, etc. It learns to inpaint occluded content behind foreground objects without requiring ground truth 3D or depth, mainly due to their utilization of *scale-invariant* view synthesis for supervision. As mentioned previously in this subsection, although Tucker and Snavely extended RealEstate10K dataset by adopting the same methods as Zhou et al. [38], yet they had to incorporate scale-normalization/scale-invariance into their training in order to circumvent the global scale factor ambiguity that arises when attempting to infer scene geometry from monocular views. They accomplish this in the following manner (Figure 2.4):

- The sparse point cloud of the scene depicted by each group of sequential video frames, the lists of all 3D points *visible* from each frame, the camera parameters of each frame, and the video frames themselves are needed for training. All these input components result from the ORB-SLAM2, COLMAP, and scale-normalization procedures of Zhou et al.
- Pairs of source and target frames $(I_s,\ I_t)$ and respective camera parameters $(c_s,\ c_t)$ are randomly picked for training, along with the respective visible point sets of source frames. The sets of visible points are converted from world co-

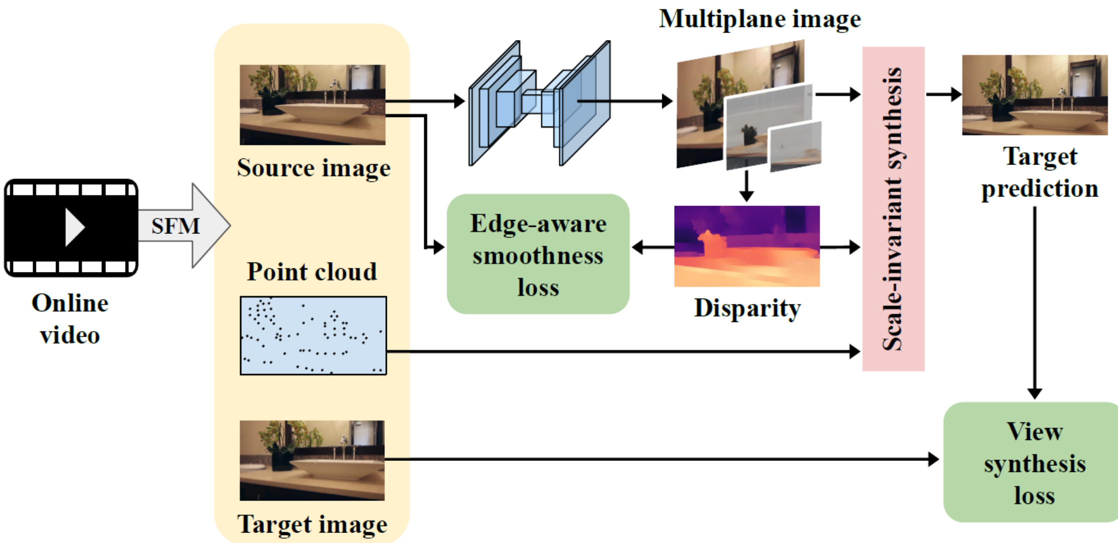


Figure 2.4: Tucker and Snavely's Single-View View Synthesis Pipeline [35]

ordinates to camera coordinates to get a final point set $P_s = \{(x,\ y,\ d),\ \ldots\}$ for each source frame, where the z-coordinate of each world point becomes the depth d of the world point from the source camera, and the mapped 2D points are denoted by the positions $(x,\ y)$ within the source image.

- Similarly to Zhou et al., Tucker and Snavely's chosen reference camera for the MPI planes (Figure 1.2) is c_s, and their preferred MPI prediction format consists of a predicted background image $\hat{I}_b$, a set of layer-wise predicted alphas, and a set of layer-wise color blending weights that (unlike Zhou et al.) are calculated from the alphas and not predicted by the network. Tucker and Snavely derives color blending weights w_i for each MPI layer i as $w_i = \underbrace{\prod_{j>i}(1-\alpha_j)}$, and final color values C_i for each layer as $C_i = w_i I_s + (1 - w_i)\hat{I}_b$.

- Similarly to Zhou et al., when rendering an MPI, Tucker and Snavely's warping function $\mathcal{W}$ uses bilinear sampling and standard inverse homography (Figure 2.3) to warp each layer from source viewpoint c_s to target viewpoint c_t: $C'_i = \mathcal{W}_{c_s,c_t}(\sigma d_i, C_i)$; $\alpha'_i = \mathcal{W}_{c_s,c_t}(\sigma d_i, \alpha_i)$. The only difference is that Tucker and Snavely's $\mathcal{W}$ scales the depths by a factor σ, which they compute separately for each training instance.

- To get the final rerendered target $\hat{I}_t$, the warped layers (C'_i, α'_i) are alpha-composited as usual:

$$\hat{I}_t = \sum_{i=1}^{D} \left(C'_i \alpha'_i \underbrace{\prod_{j=i+1}^{D} (1 - \alpha'_j)} \right) \tag{2.1}$$

Furthermore, the disparity map $\hat{D}_s$ of the source image can also be similarly synthesized from the MPI using the inverse depths d^{-1} of visible points P_s:

$$\hat{D}_s = \sum_{i=1}^{D} \left(d_i^{-1} \alpha_i \underbrace{\prod_{j=i+1}^{D} (1 - \alpha_j)} \right) \tag{2.2}$$

- DeepView [12] describes the under-braced terms in all previously mentioned formulae to be the *net transmittance* at respective depth planes i. They reason that the terms represent the fraction of the color/disparity that persists in layer i after getting attenuated through all prior layers.

- Learning the 3D scene structure from a single view is trickier than from multiple views, for only the relative pose between multiple views can implicitly resolve global scale ambiguity. But Tucker and Snavely's method is able to accept source and target inputs of unknown scale and still make rerendered images match ground-truth because they solve for the unknown scale factor as part of their MPI generation. They observe that RealEstate10K-dataset-derived inputs c_s, c_t, and P_s are consistent in scale for each training instance. They, therefore, compute σ to be the scale factor that minimizes the log-squared error between the predicted disparity map $\hat{D}_s$, bilinearly sampled at each position (x, y), and the point set P_s:

$$\sigma = exp\left[\frac{1}{|P_s|}\sum_{(x,y,d)\in P_s}(ln\hat{D}_s(x,y) - ln(d^{-1}))\right]$$

After σ is applied in warping with $\mathcal{W}$ as shown before, the rendered image no longer varies with the scale of the input viewpoints and point set, and can be used in the various loss functions.

- Their weighted aggregate loss function is given by

$$\mathcal{L} = \lambda_p\mathcal{L}^{pixel} + \lambda_s\mathcal{L}^{smooth} + \lambda_d\mathcal{L}^{depth} \tag{2.3}$$

Here, $\mathcal{L}^{pixel}$ is just the regular L_1 photometric distance between synthesize and ground-truth target views:

$$\mathcal{L}^{pixel} = \sum_{channels}\frac{1}{N}\sum_{(x,y)}|\hat{I}_t - I_t|$$

$\mathcal{L}^{smooth}$ is the *edge-aware smoothness loss* that prevents the gradients of the synthesized disparity map $\hat{D}_s$ from crossing a certain threshold (g_{min}, usually 0.05) whenever there is no edge detected in the source image, like so:

$$\mathcal{L}^{smooth} = \frac{1}{N}\sum_{(x,y)}\left(max\left(G(\hat{D}_s) - g_{min},\ 0\right) \odot (1 - E_s)\right)$$

where $\odot$ is the Hadamard product, G represents the L_1 norm of the gradient of an image summed over all three color channels, like so:

$$G(I) = \sum_{channels} ||\nabla I||_1$$

where Sobel filters are used to compute the gradient, and E_s represents a custom edge detector for the source image, which signals the presence of an edge whenever the gradient of the source image is at least a fraction (e_{min}, usually 0.1) of its own maximum value over the entire image, like so:

$$E_s = min\left(\frac{G(I_s)}{e_{min} \times max_{(x,y)} G(I_s)},\ 1\right)$$

and $\mathcal{L}^{depth}$ is a sparse depth loss given by the L_2 difference between the logs of the disparities derived using the predicted alphas (i.e., the synthesized disparity map) on the on hand and the input point set P_s and the other, like so

$$\mathcal{L}^{depth} = \frac{1}{|P_s|} \sum_{(x,y,d) \in P_s} \left(ln \frac{\hat{D}_s(x,y)}{\sigma} - ln(d^{-1}) \right)^2$$

where the computed scale factor σ that minimizes $\mathcal{L}^{depth}$ is itself included.

The network used is architecturally similar to DispNet [22]. In our work, in the process of recreating Tucker and Snavely's model and retraining it on video-chat-relevant scenes, we have reimplemented their weighted aggregate loss function, among other model features. We retained their chosen loss function weights of $\lambda_p = 1$ and $\lambda_s = 0.5$, except for picking λ_d to be 1 whereas their chosen λ_d value was 0.1. We also retained their choice of optimizer — Adam — but used a different learning rate 0.00001.

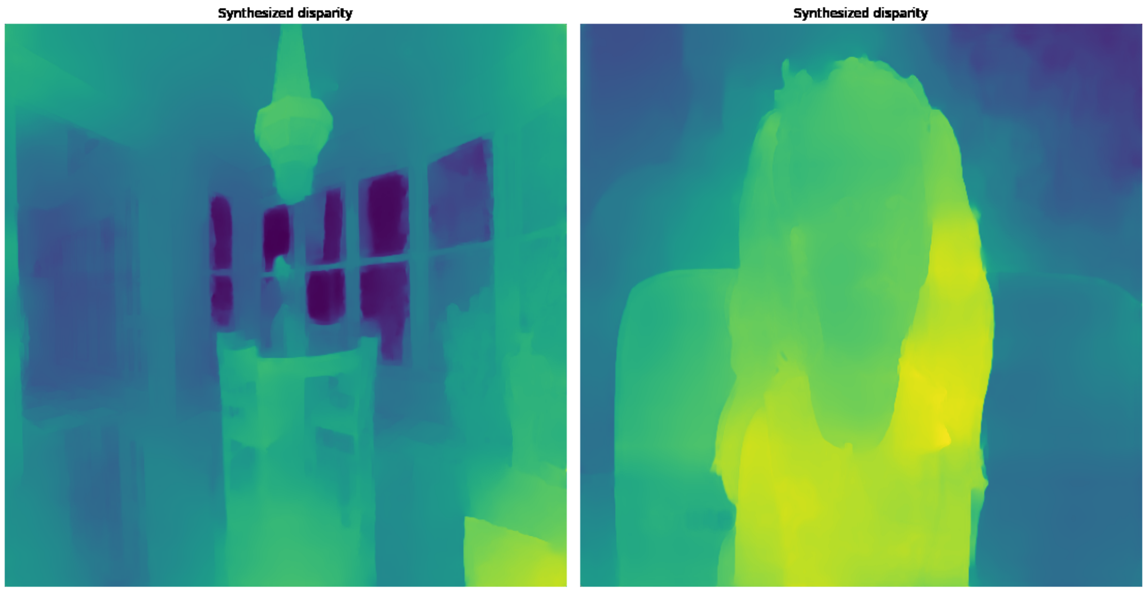
Even though there is still a lot of scope for improvement in performance with regard to model-induced artifacts reducing the quality of synthesized views, Tucker and

Snavely's authors share how the various aspects of the model contribute to it beating the state-of-the-art. They show that the scale-invariant nature of the model's supervision by view synthesis (i.e., usage of ground-truth target views) plays a major role in its success, followed by the edge-aware smoothness prior, and the chosen MPI format involving a predicted background. Another triumph of their model is that even though it does not use depth supervision at all, it is comparable to state-of-the-art depth prediction methods that use explicit depth supervision. Their project presents exciting future opportunities such as turning the model into a Generative Adversarial Network (GAN) [14] to possibly produce more extensive and realistic inpainting, and so on.

Chapter 3

METHODS

The objective of this work has been to freely rerender concurrent one-on-one video chat frames from the points of view of both participants bidirectionally and in real-time. This would help simulate the experience of conversing face-to-face with a person in the real world. We adopted Tucker and Snavely's [35] single-view MPI network, for it is the first state-of-the-art open-source single-view view synthesis network, and its popularity is eminent among various organizations since its release in 2020. When we initially ran the publicly available inference part of the network on a video chat frame, we found that the generated disparity map (Equation 2.2) was visually inaccurate. Comparatively (Figure 3.1), the inferred disparity map would be much more visually accurate whenever a real estate video frame was processed. The latter outcome is to be expected because Tucker and Snavely's model was originally trained on RealEstate10K [38] video dataset. Specifically, certain aspects of the synthesized views, such as image sharpness, would be brilliant for the real estate category of video frames by virtue of the model having been efficiently tweaked and extensively trained by the authors (given contemporary hardware limitations). Yet, synthesized video-chat-related frames alone would seem unnaturally concave/convex at arbitrary positions within each rerendered frame, not to mention the loss of perspectivity and the induction of random distortions occurring within the frame as well.



The disparity map on the left encodes a real estate scene and the one on the right, a video chat scene. The real estate map successfully shows appropriate heat/depth gradations from the hottest/closest armrest region on the bottom right to the coldest window regions toward the back of the scene. The video chat map, on the other hand, counterintuitively shows that the face of the girl in the scene is situated behind the body, and the couch in it is somehow disjointed.

Figure 3.1: Disparity Heat Maps Synthesized by Tucker and Snavely's model [35] for Real Estate and Video Chat Frames

3.1 Approach

As a primary step (Figure 3.2), we attempted to increase Tucker and Snavely's depth prediction accuracy for video-chat-relevant frames containing close-up shots of people, so we may see a drastic reduction in the number of artifacts induced in synthesized frames. This involved curating and utilizing both RealEstate10K and Mannequin-Challenge [20] datasets. The latter contains video frames that resemble video chat scenes, as it is composed solely of scenes of people pretending to be mannequins while a camera moves around them, flowing seamlessly from scene to scene. Essentially, we performed transfer learning [28] with the pretrained weights of Tucker and Snavely's model by *fine-tuning/refitting* them to a dataset other than the one they were originally trained on. Secondly (Figure 3.3), we introduced the head pose detection submodule of OpenFace 2.2 [7] into the inference pipeline of Tucker and Snavely, so that "*viewee*" video frames may be rerendered at the head pose obtained from "*viewer*" frames. We considered a few state-of-the-art open-source head pose estimation models, including WHENet [39] for its speed and consistency, and ultimately chose OpenFace 2.2 because it works well with the Deep Learning (DL) framework used by Tucker and Snavely (TensorFlow 2.2) and can be installed in the same dockerized environment as COLMAP [30, 31] and the rest of the dependencies needed by our comprehensive pipeline.

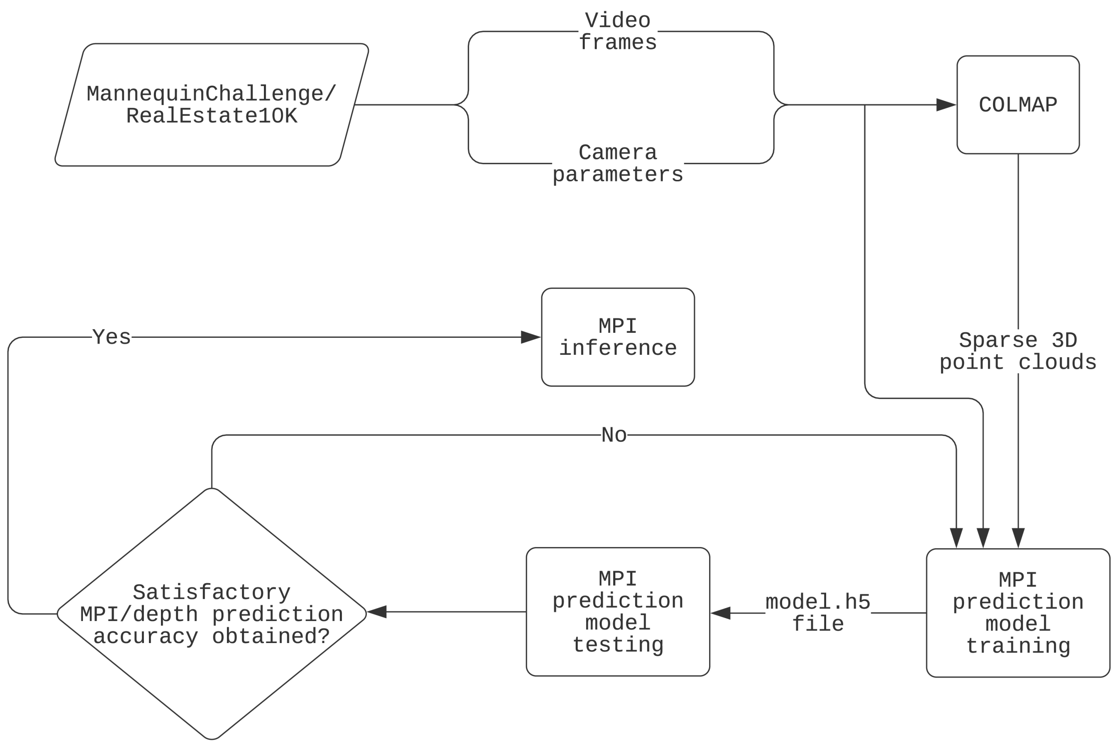


Figure 3.2: MPI Training Pipeline

Out of the non-exhaustive set of network components made publicly available by Tucker and Snavely [35], a comprehensive inference pipeline on Google Colaboratory (Section A.1) was one. It greatly helped us with our OpenFace integration and gave us the ability to visualize and present our results and demos in chapter 4 and everywhere else. They couldn't reveal certain other aspects of their codebase due to their proprietary natures. This prompted us to go about recreating Tucker and Snavely's DispNet-like model [22] first before retraining it on requisite datasets and repurposing it for video chat view synthesis. We recreated parts of the model from the code released (Section A.1) by the authors involving their network definition (convolutional layers, kernel sizes, etc.), and the code used by them for rendering views from new camera positions with homographies and related operations (Equation 2.1). We then put together other aspects of the network that called for a more involved recreation process like the data loader part and the loss functions (Equation 2.3). Requisite components of input data, including point clouds, had to be extracted and loaded in. One of the key features of Tucker and Snavely is to use sparse point cloud data to make the view synthesis loss scale-invariant (Subsection 2.1.3). To obtain such inputs, we processed both datasets with COLMAP and wrote a custom data loader. We took inspiration from Zhou et al.'s [38] stereo MPI paper for building the data loader, for the code they tailored to load in data (Section A.1) was refactored and reused by Tucker and Snavely as well. Their implementations of subsequence selection and random cropping proved useful.

We retrained the recreated network in two different ways. One group of model variants was fine-tuned exclusively on the video-chat-relevant MannequinChallenge video dataset [20] which is $\sim$96% smaller than RealEstate10K in training data, as of this writing. The other set of variants was retrained on a combination of both datasets by having the model pick same-sized batches of training data (Subsection 2.1.3) randomly and alternatingly from both datasets. We considered addressing this inherent data imbalance problem by making the model pick an appropriate proportion of RealEstate10K frames for every MannequinChallenge frame randomly selected, but ultimately voted against it in favor of resolving more pressing issues such as the training errors mentioned in section 3.3. We are grateful to the authors of Tucker and Snavely for forewarning us that there is a risk of overfitting to the much smaller

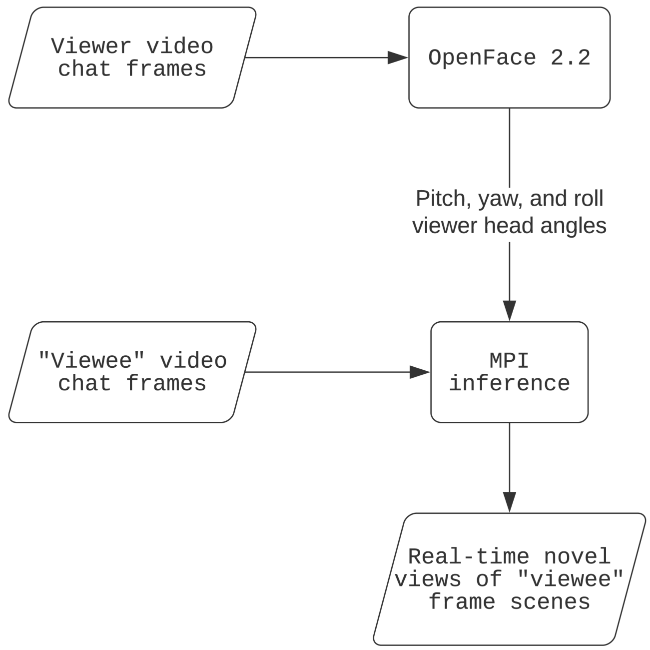


Figure 3.3: 3D Video Chat Rendering Pipeline

MannequinChallenge dataset, even though it was generally mentioned in both Zhou et al. and Tucker and Snavely that the stereo and single-view models were considerably generalizable to domains besides real estate footage. Hence, we felt the need for deploying the second set of variants to help access this risk. We could also have taken another transfer learning route of freezing all but the last few layers of the model to possibly reduce overfitting but we chose to unfreeze all layers in favor of making the variants wholly robust. The layers were thus free to learn and evolve based on the MannequinChallenge data they were newly exposed to. We stack up these variants to each other and also to the pretrained single-view model and compare their performances in chapter 4. Finally, after introducing the head pose estimation API of OpenFace 2.2 into the inference pipeline of the variants, we converted estimated head orientations into a form amenable to rendering with MPIs. This involved manipulating yaw, pitch, and roll head angles and the MPI helper functions provided by Tucker and Snavely went a long way in making this possible as well. We also visually verified for if the rerendered frames were getting seemingly aligned with the extracted head poses or not.

3.2 Data

Both Mannequin Challenge [20] and RealEstate10K [38] datasets were created by roughly the same group of researchers hailing from Google. They involved the same ORB-SLAM2, COLMAP, and scale-normalization procedures of Zhou et al. [38] (Subsection 2.1.3). Hence, both datasets consist of the same kind of metadata in text files pertaining to the downloadable videos. Each text file begins with the video's YouTube link on the first line and continues with the details of each COLMAP-processed video frame from the second line onward. Frame details include the timestamp (in microseconds), camera intrinsics, and camera extrinsics. As mentioned in subsection 2.1.3, COLMAP is a 3D scene reconstruction pipeline. It attempts to recover the 3D scene structure from even those unstructured 2D images of the scene that do not come tagged with any prior knowledge of camera intrinsics, extrinsics, and nature of objects captured. The extracted scene structure is either in the form of sparse 3D points along with the camera parameters for each input 2D image or in the form of dense 3D points with associated color information. COLMAP's pipeline can be given by: feature detection → pairwise feature matching → correspondence estimation → incremental structure from motion (Figure 3.4). Fortunately, absolute camera poses are not required by the model; only the relative ones made available with the help of COLMAP in these text files are. Our scripts to download and curate all these videos were facilitated by our compilation of a comprehensive Docker container ensuring robustness in code reusability and transferability. Resolving version compatibility issues among our project dependencies such as COLMAP and OpenFace 2.2, both in the Docker container and in Google Colaboratory proved paramount to the successful running of our experiments. All our scripts, notebooks, sample renderings, demos, and most other aspects of our code for this project can be found in our GitHub repository (Section A.1).

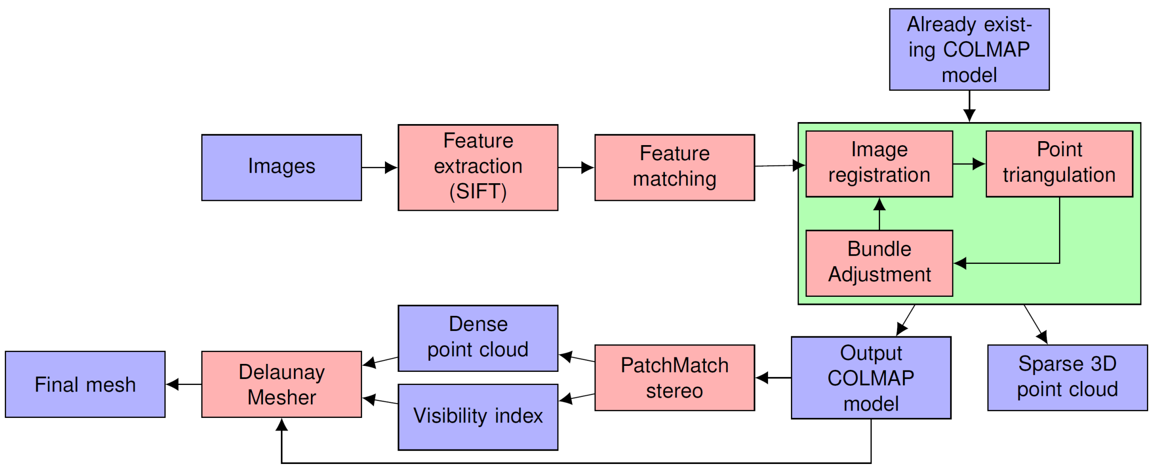


Figure 3.4: Photogrammetry Workflow used in COLMAP [26]

Although our training and testing scripts are designed to crop all incoming video frames to 512×512 pixels, we ensured that we downloaded all videos with youtube-dl at 720p resolution. This uniformity was so we could reduce the number of sources of arbitrariness in the initial process of replicating Tucker and Snavely's [35] work. Linking youtube-dl with the download management utility aria2 [5] proved very useful in bolstering youtube-dl's download speed by optimizing resource utilization. We then targeted addressing youtube-dl download errors. There would inevitably be several partial and/or skipped downloads for various reasons ranging from the videos being taken down from YouTube over time to fixable errors intrinsic to youtube-dl. Moreover, some videos were unavailable in their 720p versions and were discarded by us with the aim of maintaining consistency. In favor of maintaining the pristine versions we chose not to manually convert the varying resolutions to 720p. Although differently scaled videos should theoretically not pose any problem to training or to 3D point cloud generation with COLMAP, we opted again to go with uniformity and consequent ease of reproducibility for one and all.

We were finally able to procure 66,861 RealEstate10K videos with 9,095,528 frames and 2,364 MannequinChallenge videos with 117,811 frames, for processing. But not all downloaded videos could be processed. For instance, only ∼60000 RealEstate10K videos were actually COLMAP-processed and used for training. This is because the rest of the videos did not meet COLMAP processing requirements. And, it would have taken 200 days to process all 66,861 videos with COLMAP with CPUs alone. Fortunately, we were able to avail the benefits NVIDIA Tesla V100 GPUs (rated the best server models in 2020) at Cal Poly and could bring down the processing time to 25 days. In these ways, we obtained the required points clouds and frames for both training and testing.

3.3 Implementation

We attempted to generate accurate MPI representations for close-up targets such as heads and upper bodies, and improve the pixel accuracies of views synthesized from these MPIs. After putting together the data loader to feed the datasets and point clouds into the network, we recreated loss functions from the textual descriptions in the single-view MPI paper. As mentioned in subsection 2.1.3, we likened our training process to Tucker and Snavely [35] with regard to various aspects such as the use of TensorFlow 2.2, ADAM solver, a pixel loss weight of 1, a smoothness loss weight of 0.5, etc. We experimented with choices of learning rate and depth loss weight but generally picked 0.00001 and 1, respectively, contrary to the 0.0001 and 0.1 used in Tucker and Snavely. We reduced the learning rate because we were fine-tuning the pretrained model rather than training from scratch. The requirement that we had to have view synthesis quality as supervision was fulfilled by taking a frame one frame apart from each chosen training frame as target ground truth. We trained for a number of steps rather than for a number of epochs. Our data loader randomizes batch picking not only for testing but also for training. Moreover, we have not yet been able to go beyond the model experimentation stage. Exposing the model to a wide variety of frames is the way to go in this stage. For the model to be training sequentially on all frames clip by clip, and covering entire datasets multiple times in multiple epochs, it should be free of any errors that impede its progress toward convergence. We have not been able to bring our model up to that stage yet.

We used wandb.ai [9] for experiment tracking and it proved to be a valuable tool for our entire process. It helped us spin different variants of the model, chiefly characterized by their being trained either on MannequinChallenge alone or on a combination of both datasets. As with some notable attempts at model training in the community, we encountered Not a Number (NaN) gradient errors that took a good chunk of our resolution efforts in this work, but ultimately could not be resolved. NaN losses signal that the issue of vanishing/exploding gradients may be present. In this work, NaN gradients could only be reduced in their frequency of occurrence from once in several hundred steps to once in several thousand steps. wandb.ai helped immensely in resuming not just the training runs themselves but also the activity of logging training metrics right from the point where the run broke off due to a NaN error. What also helped bring down the frequency of encountering NaNs, we believe, was the fact that we removed all those videos from the training/testing process that had at least one frame with a point cloud composed of less than two 3D points. Our Linux command to locate such point cloud `.txt` files (Section A.2) would take about 3 hours to sift through a set of 2500 point cloud directories with one `.txt` file per video frame. Replacing `cumprod` used in several places in the single-view MPI source code with `safe_cumprod`, as suggested to us by one of the authors of the single-view paper, also helped reduce the frequency of encountering NaNs. One of the issues that we were able to completely resolve was the occasional throwing of `ValueErrors` by our data loader. We also attempted to redress the rendered artifacts mentioned in section 3.1 and determine if real-time, high-quality view synthesis was indeed possible without game engines.

We used customized training loops with TensorFlow's `tf.GradientTape` context [3]. However, we found that the gradient calculation (Section A.2) would take about one minute! We were using a batch size of 8 at that time on an NVIDIA V100 GPU. But the authors of the single-view MPI paper informed us that their gradient calculation would take less than a second even on a single worker. They then correctly diagnosed our issue to be that we were doing everything in *eager mode*, which would lead to the accumulation of a lot of overhead. They suggested that using Keras's `model.fit`, or using the old estimator system of TensorFlow, or just wrapping things in `tf.function` should allow the critical parts to run in graph mode and be faster. They also suggested that things were probably too big to fit on our GPU. The authors had used a batch size of 4. We ultimately adopted the use of `tf.function` wrapper as well as a batch size of 4 and were able to complete implementing our training and testing pipelines.

e comma-delimited (.csv) format. To preserve these features, save it in an Excel file format. Don't show

	KG	KH	KI	KJ	KK	KL	KM	KN	
_Z	eye_lmk_Z	pose_Tx	pose_Ty	pose_Tz	pose_Rx	pose_Ry	pose_Rz	x_0	x_1
.5	612.1	39.2	-143.3	687.2	-0.094	0.012	0.164	610.7	
.2	606.9	36.1	-143.2	679.5	-0.123	-0.006	0.151	608.7	
.2	594.8	33	-143.3	673.1	-0.133	-0.013	0.131	603.1	
92	591.5	30.5	-144.2	671.4	-0.132	-0.007	0.11	599	
.5	588.9	27.9	-144.8	669.3	-0.124	-0.006	0.091	594.7	
.5	589.1	25.7	-145.2	668.2	-0.125	-0.003	0.073	591.3	
.3	590.9	23.4	-145.2	665.5	-0.119	0.005	0.052	587.6	
.2	593.8	21.3	-145	663.6	-0.116	0.01	0.031	583.8	
.9	599.4	19.5	-144.9	662.6	-0.113	0.018	0.012	580.2	
.1	605.7	17.7	-144.4	660.8	-0.114	0.026	-0.006	577.3	
09	608.6	16.3	-144	659.9	-0.114	0.039	-0.021	575.3	
.9	592.6	14.6	-143.7	659.7	-0.112	0.047	-0.036	572.8	
.4	577.1	12.7	-143.4	660.2	-0.099	0.06	-0.053	569.9	
.1	581.8	10.2	-142.7	659.7	-0.095	0.076	-0.072	566.8	

Figure 3.5: A Snapshot of OpenFace 2.2 [7] Outputs

We then inserted OpenFace 2.2 [7] into the inference pipeline of one of our better performing model variants and attempted to emulate a video chat system, one half at a time. We subjected a “viewer” video sequence to head pose extraction by OpenFace 2.2 from all frames, as show in figure 3.3. We used one of the utility functions in the single-view MPI modules, `geometry.pose_from_6dof`, to extract the yaw, pitch, and roll angles of the “viewer” frames in a manner conducive to being accepted by the MPI inference. We then rendered the “viewee” video sequence at the head pose of the “viewer” frames with matching timestamps. Perhaps more precision could have been added by using not just head pose estimation but also gaze estimation with OpenFace. A snapshot of OpenFace 2.2 outputs for multiple frames in a sequence is shown in figure 3.5

Chapter 4

EXPERIMENTS AND RESULTS

In this chapter, we present some quantitative and qualitative evaluations of the variants of the recreated single-view MPI model retrained on different combinations of the MannequinChallenge and RealEstate10K datasets. We use the pretrained weights of the single-view MPI model as the benchmark and compare the abilities of all model variants at hand to generate novel views. We adopt some of the quantitative metrics from Tucker and Snavely's single-view MPI paper [35] — PSNR, SSMI [36], and LPIPS [37] — to give numeric values to the similarities between MPI-rendered video frames and the corresponding ground truth target frames the rendering process attempts to replicate.

The model variants used to compute the metrics stated above are characterized by the following hyperparameters/metadata:

- Depth loss weight, as explained in subsection 2.1.3.
- The number of disparity map channels specified in the `tf.function` input signature for the bilinear sampling function in our training script (Section A.1), `sample_disparities(disparity,points)`, involving the predicted disparity and the input visible points.
- The lower bound on the number of visible points required per frame. Videos with even one frame having the number of visible points below this threshold would be removed from training.

- The choice of datasets used to train on — MannequinChallenge, RealEstate10K, or both.
- Whether multiple GPU workers were engaged or not.

Even seemingly innocuous hyperparameter values such as those for the number of disparity map channels specified, we believe, could have easily held sway over training progress. Pitting these variants against each other in terms of the three computed metrics helped us select the best variant to simulate one half of a video chat with.

We manually sifted through the in-built test set of the MannequinChallenge dataset to handpick a set of 333 videos with 12,595 frames in total. These ORB-SLAM2-curated sequences had video-chat-relevant features. They mostly had the heads and torsos of people being focused on rather than there being wide shots of entire bodies. The number of people in the frames was mostly limited to one or two as opposed to there being multiple people featured. Moreover, although not a strict requirement, the head pose of people in these frames was roughly or even very loosely aligned with the camera. There was hardly anybody in any frame that seemed to look directly at the camera, such as would be expected in an actual video chat scenario. We put these cherry-picked frames in the `test-yes/` bin. We also curated `test-maybe/` (300 videos with 12,831 frames) and `test-no/` (24 videos with 728 frames) bins. They consisted of the rest of the MannequinChallenge test set with sequences either having no relevance to typical video chat settings (like there being hardly anyone in the frames) in the case of `test-no/` or falling heavily in the gray areas between `test-yes/` and `test-no/` in the case of `test-maybe/`. We even occasionally interspersed the `test-yes/` and `test-maybe/` bins with videos containing sequences that portrayed people facing diametrically opposite to the camera. This was just so we could really challenge the model variant being tested.

Of the various aspects of the code that we modelled from the textual descriptions and relevant code snippets obtained from both the single-view and stereo MPI papers such as `generator_wandb.py`, `data_loader.py`, `train_wandb.py`, and `test.py`, the scripts relevant to the experiments in this section are `test.py` and `generator_wandb.py` (Section A.1). For testing, the generator first aggregates all videos names from the directory input to it and for each of these, it picks various `reference_image` and `target_image` pairs which are internally 5 frames apart. `reference_image` is the frame that `test.py` uses to infer the MPI representation of the scene from and `target_image` is supposedly a view of the same scene from a different angle. The possibility that, when the camera moves from one scene to another in the same video, `reference_image` may depict a scene different from the one captured by `target_image` is expected to be extremely low as both datasets have been curated by similar ORB-SLAM2 and COLMAP processes. In such hypothetical cases, `target_image` will be erroneously rendered by `mpi.render` function as the corresponding `rendered_image`. But since we take the mean of the computed metrics over hundreds of `test.py` processed `reference_image`, `target_image` pairs, we believe the final accuracies of a variant's mean metrics will not be off the tracks much and that they shall still be used to determine a variant's performance satisfactorily. Each of the three metrics are calculated between `target_image` and `rendered_image`, which are situated 5 frames apart along the camera trajectory of the respective clip. We did not repeat the same test process for frames 10 apart, which would just have done been to show (as in the case of the single-view MPI paper) that the longer the baseline between reference/source and target views, the less the accuracy will be of the rendered image. On the same note, we have also not calculated the metrics internally for all processed `(reference_image, target_image)` pairs, which would just have been done to catch the hypothetical anomalies of the complete scene changes mentioned before.

We also took an interesting little detour in our project when we attempted to parallelize training across multiple GPUs, which we believed would allow us to increase the batch size[1] and thereby let larger and larger parts of our 60000+ training ready sequences with associated point clouds be used for learning by our recreated model. This would have assisted the model in better avoiding local minima and maxima. But, since TensorFlow's direct conversion procedure that would let standard single-GPU-utilizing scripts become multi-GPU-faring is as yet still an evolving process requiring careful attention to resource allocation issues among the various replicas of the parallelizable aspects of the model[2] spread across GPUs, our training got undercut after a good start by a resource exhaustion error at training step 178. Nevertheless, we computed all three metrics for this other model variant retrained on MannequinChallenge data using `tf.distribute.MirroredStrategy`, and capable of harnessing the power of multiple GPUs.

The rest of this chapter presents the results of the experiments done with the various model variants and the baseline pretrained model. We then cap it all off by presenting the results of incorporating OpenFace 2.2 into the inference pipeline. As of this writing, our generator is only able to pick random pairs of reference and target frames from the 333 `test-yes/` videos. Sequential pair-picking would avoid possible repetition of selected pairs and allow for an exhaustive coverage of the test set. Given that even the smaller of the two datasets has 100,000+ frames and that we have not been able to resolve the issue of the synthesized disparity maps becoming smudgier and smudgier until they turn completely gray/monochromatic even before any of the variants hit 14,000 training steps, it is not very likely that the model may see the same frame twice. So perhaps, computing evaluation metrics with training data can

[1] currently limited to 4 pairs of reference and target images and their respective camera poses and intrinsics, along with the 3D points of the reference image

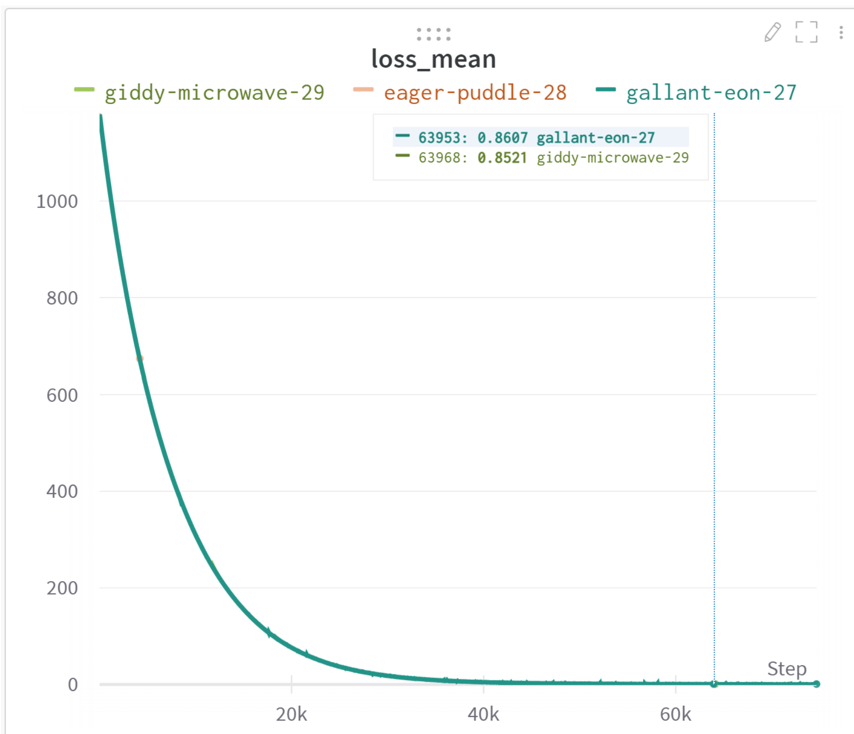
[2] such as the dataset generator, the loss functions aggregator, etc.

double in as doing the same with validation data itself, even though we haven't set aside validation data. As for the metrics, an LPIPS value of 0 indicates there is either a perfect match between the images being compared or the images being compared are one and the same. To the contrary, SSIM values of 1 indicate a perfect match. Both these metrics range from 0 to 1. PSNR values, measured in decibels (dB), don't generally have an upper limit but values 20 dB and higher are considered acceptable. In calibrating our implementations of these metrics, when we compared an image with itself, we found the mean LPIPS, SSIM and PSNR values over 300 images to be close to 0, 1, and 30, respectively.

Model Variant / wandb.ai Codename	**Depth Loss Weight**	**Number of Disparity Map Channels Specified w/ tf.function**	**Minimum Number of Visible Points per Frame**	**Steps Trained for**	**PSNR** ↑ Target vs Rendered	**SSIM** ↑ Target vs Rendered	**LPIPS** ↓ Target vs Rendered
RealEstate10K Pretrained Baseline / none	0.1	unknown	unknown	>100000	16.105	0.549	0.418
MannequinChallenge / northern-monkey-4	1	None	3	34918	16.018	0.534	0.296
MannequinChallenge / sunny-grass-5	1	1	100	34741	16.022	0.534	0.297
MannequinChallenge / fast-monkey-7	0.1	1	100	33546	16.005	0.533	0.298
MannequinChallenge + RealEstate10K / gallant-eon-27	1	None	3	37258	16.015	0.534	0.297
MannequinChallenge + RealEstate10K / giddy-microwave-29	0.1	1	100	31951	15.998	0.534	0.297
MannequinChallenge w/ Multi-GPU / none	1	None	2	178	16.105	0.549	0.418

Figure 4.1: Model Variants' Mean PSNR, SSIM, and LPIPS Evaluation Values over 300 Testing Instances

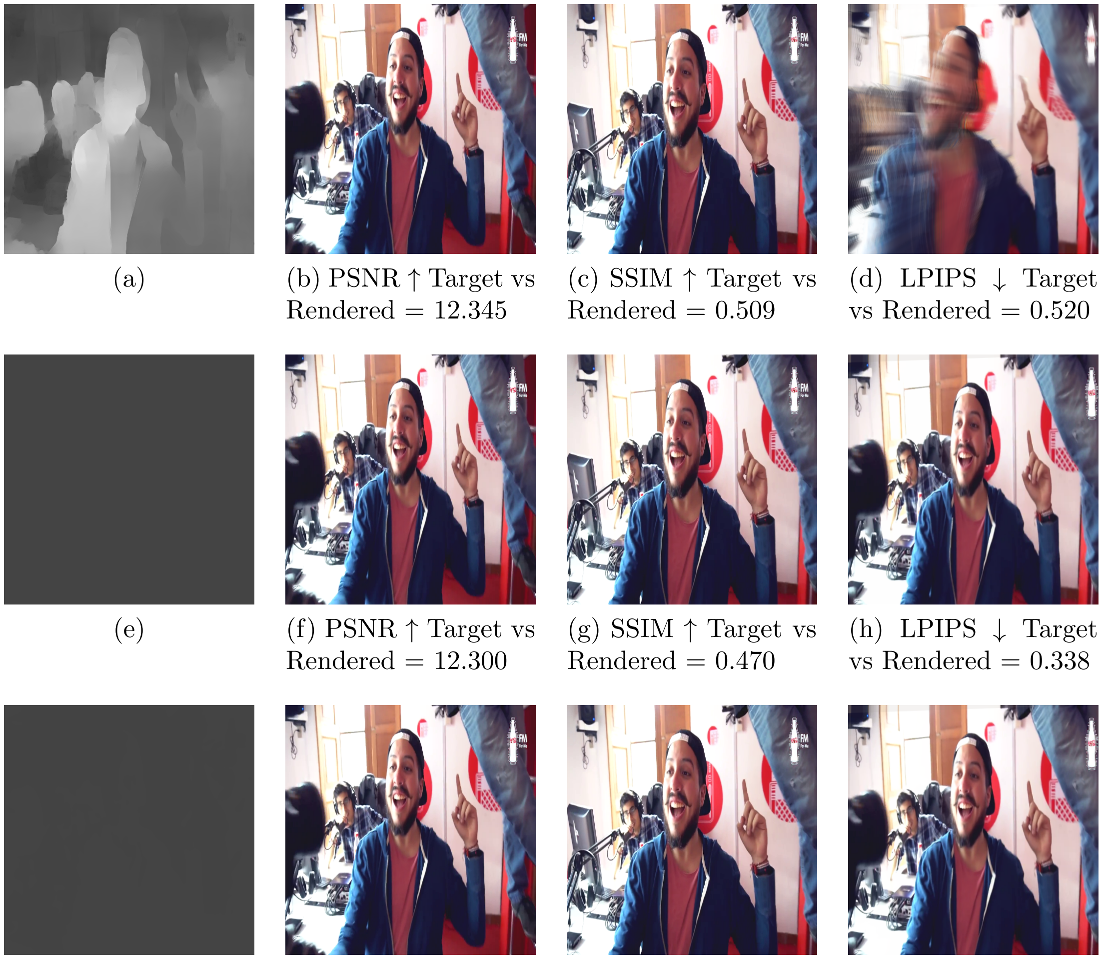
We can get a sense of how the variants stack up against one another from figure 4.1. Perceptual similarity comes the closest to the way humans judge the picture quality of an image. Hence, we chose the variant northern-monkey-4 for the final step of simulating a video chat. These interesting names are automatically allotted by wandb.ai at the start of any training run. If the run is relatively successful, we use the final model produced by it as one of our variants and evaluate its performance. All our variants have been trained to the limit and to the point where the loss becomes less than 1, after having come down all the way from 1188, and stagnates. This has always occurred sooner than 25,000 training steps for all our variants (Figure 4.2). It goes to show that had we been entirely successful in our implementation of the model, we would also have been able to train for way more than 100,000 steps, similarly to Tucker and Snavely [35].



wandb.ai somehow always shows twice the number of actual training steps completed on our server. Hence all our variants' training stagnates at 30,000+ steps and not at the 60,000+ steps shown in this wandb.ai-logged loss chart.

Figure 4.2: Typical Mean Loss Chart for Any of Our Training Runs

What further validates our choice of northern-monkey-4 is the set of output visualizations for all relatively successful model variants shown in figures 4.3 and 4.4. These outputs further reveal that even prior to all our fine-tuning the pretrained



(a) (b) PSNR ↑ Target vs Rendered = 12.345 (c) SSIM ↑ Target vs Rendered = 0.509 (d) LPIPS ↓ Target vs Rendered = 0.520

(e) (f) PSNR ↑ Target vs Rendered = 12.300 (g) SSIM ↑ Target vs Rendered = 0.470 (h) LPIPS ↓ Target vs Rendered = 0.338

(i) (j) PSNR ↑ Target vs Rendered = 12.282 (k) SSIM ↑ Target vs Rendered = 0.470 (l) LPIPS ↓ Target vs Rendered = 0.338

Variants from top to bottom: baseline, gallant-eon-27, giddy-microwave-29
Outputs from left to right: disparity map, reference frame, target frame, rerendered target

Figure 4.3: Baseline and MannequinChallenge+RealEstate10K-based Model Variants' Output Visualizations with a MannequinChallenge Target Frame

model found it hard to synthesize the disparity for video-chat-relevant frames. In the testing example used, the person has clearly moved closer to the camera but the frame synthesized by the baseline model shows "stack of cards" effects. This could potentially also be the reason that while the picture quality for the renderings seems to have been greatly improved by our fine-tuning (evident from the improved LPIPS values), the already nebulous disparity synthesis (when it comes to video chat frames) has been rendered asunder. It also stands to reason that perhaps depth/disparity is taken more into account by the SSIM metric than the other two metrics, owing to the stark decrease in SSIM values for the fine-tuned variants. It is structural similarity after all, and we have already established that depth is part of the 3D structure of the scene. Hence, we have also been further validated in our efforts to even stick to the course of retraining the baseline model in the first place.

(a) (b) PSNR ↑ Target vs Rendered = 12.315 (c) SSIM ↑ Target vs Rendered = 0.470 (d) LPIPS ↓ Target vs Rendered = 0.337

(e) (f) PSNR ↑ Target vs Rendered = 12.315 (g) SSIM ↑ Target vs Rendered = 0.470 (h) LPIPS ↓ Target vs Rendered = 0.337

(i) (j) PSNR ↑ Target vs Rendered = 12.284 (k) SSIM ↑ Target vs Rendered = 0.471 (l) LPIPS ↓ Target vs Rendered = 0.339

Variants from top to bottom: northern-monkey-4, sunny-grass-5, fast-monkey-7
Outputs from left to right: disparity map, reference frame, target frame, rerendered target

Figure 4.4: MannequinChallenge-based Model Variants' Output Visualizations with a MannequinChallenge Target Frame

In reference to the qualitative results presented throughout this work, we invoke the reader to adopt Tucker and Snavely's [35] use of pointers such as the handling of occluded content, the production of undesirable artifacts at the edges of foreground objects, and so on, to qualitatively compare the discrepancies in the results generated by each model variant. Similarly, visually checking for the accuracy of the synthesized disparity maps, as was illustrated at the beginning of chapter 3, is also useful in verifying the quality of the MPIs produced. We encourage the reader to zoom into the electronic version of this thesis or take to the GitHub repository accompanying this work (Section A.1) for easier visual verification.

Chapter 5

DISCUSSION

Through this thesis, we have had the opportunity to simulate both halves of a 2-way pipeline that is able to render novel views from the perspectives of both participants in a video chat. Going by the synthesized monochromatic disparity maps and even by the SSIM values, we found that our model variants struggled to synthesize disparity well. Going by the LPIPS values, we found that they excelled at synthesizing the actual target view itself. This is indeed unexpected owing to the fact that only one of the given 32 MPI layers is able to essentially duplicate the reference image in its entirely.

Although the sharpness of the rerendered images is almost twice as good with our chosen model variant as with the pretrained baseline, the predicted MPIs layers have all but collapsed to a single depth layer. This is also evident from the way the training would start to produce completely gray disparity maps from around step 14,000 onward, as noted in chapter 4. We believe that the reason for this is more likely to be found in the weights we assigned to our various loss functions that aggregate into a mean loss. Ablation experiments involving taking out the pixel loss and/or bringing the smoothness loss way down would help to isolate the issue even more. Although we made our best efforts to reconstruct the loss functions and the rest of training setup as close to the textual descriptions in the paper as possible, it would definitely shine a lot more light on the root cause of the problem if we are able to access the training script of the authors — something that they've had to keep from the public. Also, since we were also meticulous with our data curation, we don't believe it is likely that the input data has any part to play in the generation of NaN loss errors.

One of the obvious next steps would be to perform *hyperparameter sweeps* with wandb.ai to find optimal hyperparameters, including the weights of the loss functions, and potentially solve the vanishing/exploding gradients problem which could very well be related to the issue of the swiftly saturating disparity maps. If we are actually able to get the plan to work, it would reveal why the pretrained model found it hard to synthesize disparity for video chat frames in the first place: it was not exactly as generalizable as the authors hinted it might be. But, if after running all possible hyperparameter sweeps with something like wandb.ai, we still find that the model performs poorly, then the obvious next thing to look at would be the actual training scripts used by the authors to discover how way off the mark we could have been in replicating their network.

5.1 Conclusion

We assembled 2020's state-of-the-art single-view view synthesis pipeline. We applied Multiplane Images, which are essentially mini-local-light-field representations, to the field of 3D video chat because they are one of the first representations capable of real-time, high-quality, spatially-consistent view synthesis. We completed implementing both ways of a potentially real-time, rendering pipeline that takes in the head pose of each "viewer" video frame and rerenders the corresponding "viewee" video frame — the one that syncs with the timestamp of the "viewer".

5.2 Future Work

We consider exciting future opportunities with this project in this section. We may increase the training speed of the MPI model by making it a multi-GPU model with the constantly-evolving, cutting-edge `tf.distribute.Strategy` API for distributed training with TensorFlow/Keras. We could perhaps implement taking the average of the head poses of multiple people in the video frames of multiple-participant video conferences rather then just one-on-one video chats and make their average head pose change the rendering viewpoint of the scene to be rerendered. We could proceed to make the pipeline real time by involving a game engine or any other framework capable of further improving real time rendering. We may try training on variable resolution video frames and not all just 720p ones. Overfitting can be further reduced by using a CNN in place of the gradient descent algorithm, similar to Flynn et al.'s DeepView [12].

BIBLIOGRAPHY

[1] Cal Poly Github. `http://www.github.com/CalPoly`.

[2] What is difference between multi view stereo (MVS) and structure from motion (SFM) methods in 3D surface reconstruction? *Quora*.

[3] Custom training: walkthrough | TensorFlow Core.

[4] Svetlana Lazebnik, 2019.

[5] aria2 - The ultra fast download utility, Dec. 2021. original-date: 2010-11-27T09:41:48Z.

[6] N. Adaloglou. Intuitive Explanation of Skip Connections in Deep Learning. *AI Summer*, Mar. 2020.

[7] T. Baltrusaitis, A. Zadeh, Y. C. Lim, and L.-P. Morency. OpenFace 2.0: Facial Behavior Analysis Toolkit. In *2018 13th IEEE International Conference on Automatic Face Gesture Recognition (FG 2018)*, pages 59–66, May 2018.

[8] C. Bavor. Project Starline: Feel like you're there, together. *Google*, May 2021.

[9] L. Biewald. Experiment Tracking with Weights and Biases, 2020. Software available from wandb.com.

[10] R. Collins. A space-sweep approach to true multi-image matching. In *Proceedings CVPR IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 358–363, San Francisco, CA, USA, 1996. IEEE.

[11] S. Fidler. Depth from Two Views: Stereo. pages 1–43.

[12] J. Flynn, M. Broxton, P. Debevec, M. DuVall, G. Fyffe, R. Overbeck, N. Snavely, and R. Tucker. DeepView: View Synthesis With Learned Gradient Descent. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2362–2371, Long Beach, CA, USA, June 2019. IEEE.

[13] J. Flynn, I. Neulander, J. Philbin, and N. Snavely. Deep Stereo: Learning to Predict New Views from the World's Imagery. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5515–5524, 2016.

[14] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[15] T. V. Haavardsholm. Forelesninger - UNIK4690 - Vår 2016 - Universitetet i Oslo, 2016.

[16] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, Cambridge, 2 edition, 2004.

[17] N. K. Kalantari, T.-C. Wang, and R. Ramamoorthi. Learning-Based View Synthesis for Light Field Cameras. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia 2016)*, 35(6), 2016.

[18] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, Jan. 2017.

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[20] Z. Li, T. Dekel, F. Cole, R. Tucker, N. Snavely, C. Liu, and W. T. Freeman. Learning the Depths of Moving People by Watching Frozen People. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[21] D. G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, Nov. 2004.

[22] N. Mayer, E. Ilg, P. Häusser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox. A Large Dataset to Train Convolutional Networks for Disparity, Optical Flow, and Scene Flow Estimation. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4040–4048, 2016.

[23] B. Mildenhall, P. P. Srinivasan, R. Ortiz-Cayon, N. K. Kalantari, R. Ramamoorthi, R. Ng, and A. Kar. Local Light Field Fusion: Practical View Synthesis with Prescriptive Sampling Guidelines. *ACM Transactions on Graphics (TOG)*, May 2019.

[24] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós. ORB-SLAM: A Versatile and Accurate Monocular SLAM System. *IEEE Transactions on Robotics*, 31(5):1147–1163, Oct. 2015. Conference Name: IEEE Transactions on Robotics.

[25] E. Penner and L. Zhang. Soft 3D reconstruction for view synthesis. *ACM Transactions on Graphics*, 36(6):235:1–235:11, Nov. 2017.

[26] C. Pinard and A. Manzanera. Does it work outside this benchmark? Introducing the Rigid Depth Constructor tool, depth validation dataset construction in rigid scenes for the masses. *arXiv:2103.15970 [cs]*, Mar. 2021.

[27] P.-L. Pröve. An Introduction to different Types of Convolutions in Deep Learning. *Medium*, Feb. 2018.

[28] P. Radhakrishnan. What is Transfer Learning? *Medium*, Oct. 2019.

[29] D. Scharstein and R. Szeliski. A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms. *International Journal of Computer Vision*, 47(1):7–42, Apr. 2002.

[30] J. L. Schönberger and J.-M. Frahm. Structure-from-Motion Revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[31] J. L. Schönberger, E. Zheng, M. Pollefeys, and J.-M. Frahm. Pixelwise View Selection for Unstructured Multi-View Stereo. In *European Conference on Computer Vision (ECCV)*, 2016.

[32] J. Shade, S. Gortler, L.-w. He, and R. Szeliski. Layered depth images. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques - SIGGRAPH '98*, pages 231–242. ACM Press, 1998.

[33] P. P. Srinivasan, R. Tucker, J. T. Barron, R. Ramamoorthi, R. Ng, and N. Snavely. Pushing the Boundaries of View Extrapolation With Multiplane Images. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 175–184, Long Beach, CA, USA, June 2019. IEEE.

[34] R. Szeliski and P. Golland. Stereo Matching with Transparency and Matting. *International Journal of Computer Vision*, 32(1):45–61, July 1999.

[35] R. Tucker and N. Snavely. Single-view View Synthesis with Multiplane Images. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[36] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, Apr. 2004.

[37] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In *CVPR*, Apr. 2018.

[38] T. Zhou, R. Tucker, J. Flynn, G. Fyffe, and N. Snavely. Stereo Magnification: Learning View Synthesis using Multiplane Images. In *SIGGRAPH*, 2018.

[39] Y. Zhou and J. Gregson. WHENet: Real-time Fine-Grained Estimation for Wide Range Head Pose. *arXiv:2005.10353 [cs]*, Sept. 2020.

[40] zikuicai. Derivation of formula (2), inverse homography · Issue #19 · google/stereo-magnification. *GitHub*, 2019.

APPENDICES

Appendix A

CODE SOURCES AND SNIPPETS

A.1 Code Sources

- Tucker and Snavely's [35] network definition: `nets.mpi_from_image`
- Tucker and Snavely's rendering code: `mpi.render`
- Zhou et al.'s [38] data loader: `loader.py`; `datasets.py`
- Tucker and Snavely's comprehensive inference Google Colaboratory notebook: `single-view-mpi.ipynb`
- The GitHub repository for this thesis including sample renderings and demos: `https://github.com/anuraguppuluri/view-synthesis.git`

A.2 Code Snippets

- Gradient calculation:

```
grads = tf.GradientTape().gradient(loss,
    model.trainable_weights)
```

- Linux command to locate point cloud `.txt` files with less than 2 3D points:

```
find -type f -exec bash -c '[ $(grep -cm 2 ^ "${1}") != "2" ]
    && echo "${1}"' -- {} \;
```