ADAPTING SINGLE-VIEW VIEW SYNTHESIS WITH MULTIPLANE IMAGES TO 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

October 2021

COMMITTEE MEMBERSHIP

TITLE: Adapting Single-View View Synthesis with Multiplane Images to 3D Video Chat

AUTHOR: Anurag Uppuluri

DATE SUBMITTED: October 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 to 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 [23] single-view view synthesis paper — by retraining it on *MannequinChallenge Dataset [14]* — 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 [26].* 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 was fortunate to interact with, faculty and staff at Fresno State and all the other schools I am an alumnus of, my family, and my friends for and all their guidance, love, support, and hard work in helping make my dreams a reality. It's as they say: you tend to a sapling regularly 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.
- 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 TABLES

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 like FaceTime, Zoom, and Google Meet, 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.

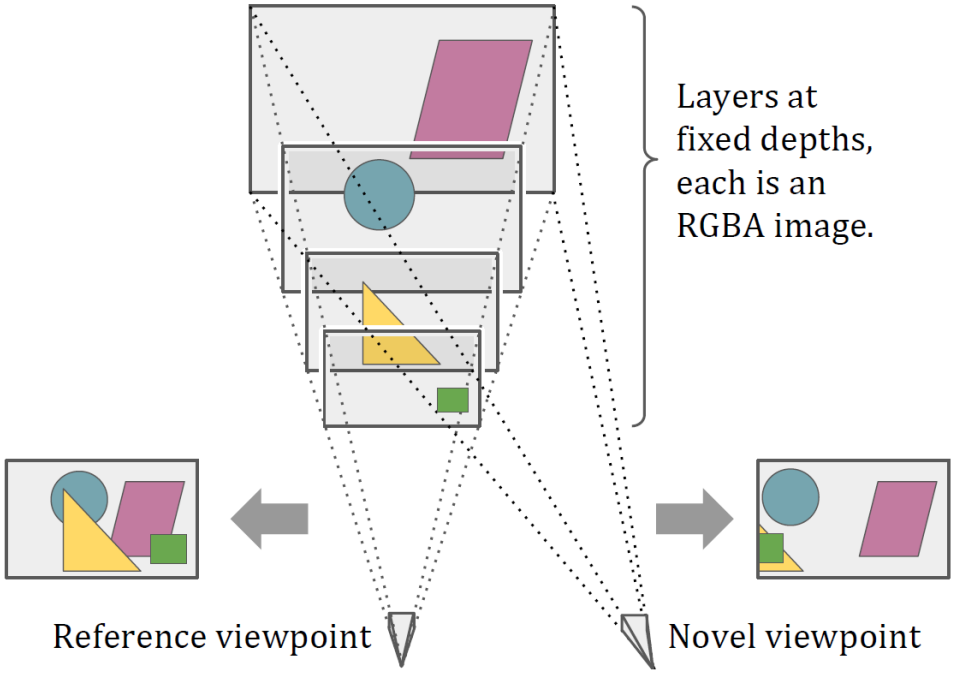
Project Starline is reported to use a groundbreaking light field rendering system that will improve glasses-free 3D / automultiscopic video chat experience by leaps and bounds when it releases later this year.

Figure 1.1: Google's Project Starline []

is reported to use 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. [26] (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. 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 in the MPI. To render from an MPI one sim-

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

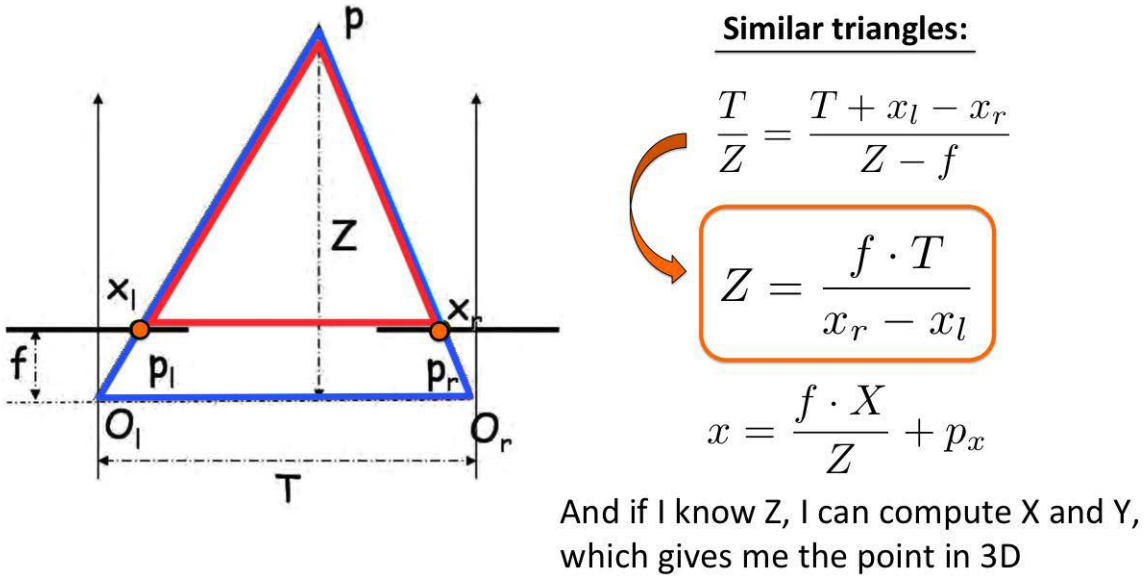
ply needs to alpha-blend all layers in back-to-front order, as explained in section 2.1. One popular instance of such depth planes used in an MPI is a set of 32 planes positioned at equidistant disparity, with the near and far planes being at 1m and 100m in 3D world space, respectively. Since disparity 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.



A given image is reprojected onto multiple fronto-parallel MPI planes within the view frustum a common reference viewpoint that may or may not match with the given image's viewpoint. A novel image is synthesized by alpha-blending these intermediate layers in back-to-front order.

Figure 1.2: The Volumetric/Layered MPI Representation

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 (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



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 [6]

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, the ganglia therein, 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.

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 [23] 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 no 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 instructions for 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 3.3. As for simulating the 3D video chat experience itself, we linked-up the API of OpenFace 2.2 — a preexisting head pose estimation model — to the MPI inference procedure so the MPI inference may generate novel views rendered in the head pose evaluated by OpenFace 2.2, as explained in section 3.4

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. [26] and 2020's Tucker and Snavely [23], 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 Flynn et al. [8] and Kalantari et al. [11] from Zhou et al. as it, in turn, is heavily inspired by them and surpasses them performance-wise. As for progress in the field of 3D video chat, we

have mentioned the yet to be released 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 [13] 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 [5], 1998's Shade et al. [20], and 1999's Szeliski and Golland [22]. 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 [19].

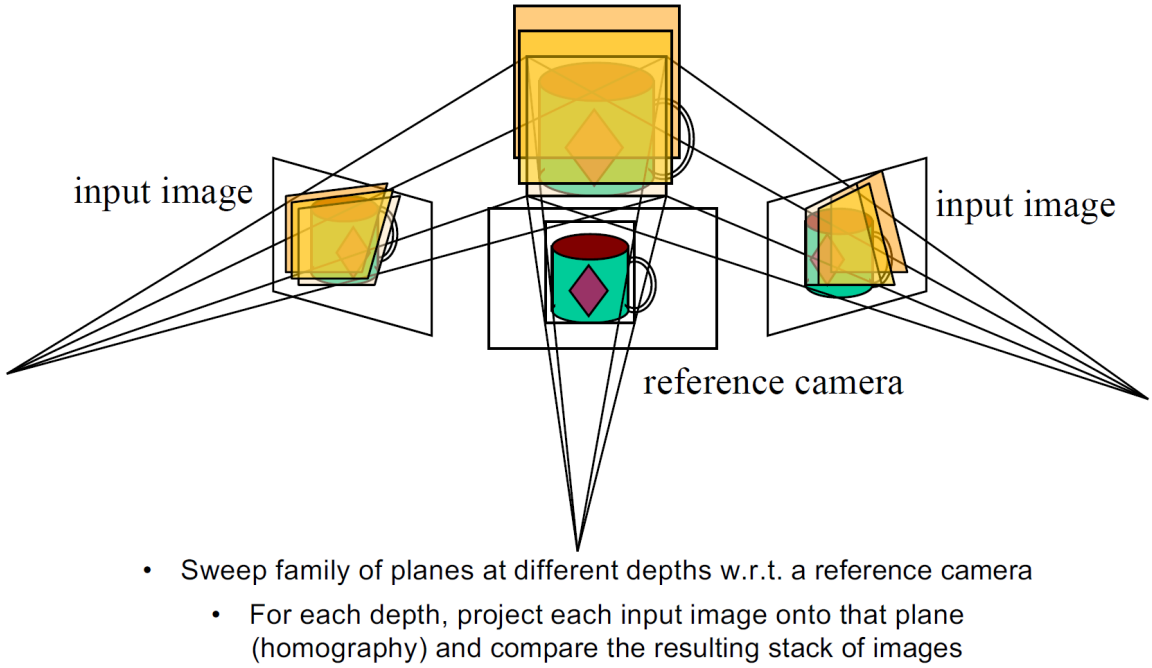


Figure 2.1: Plane Sweep Volume Representation [2]

Collins [5] 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 vote

[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

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 [20] 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, MPI has fixed depths for each layer as opposed to the variable layer depths of LDI (and PSV). 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 [22] 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 like SIFT[2] in pixel-and-sub-pixel-wise disparity estimation for 3D scene reconstruction from rec-

[2]Scale-Invariant Feature Transform [15]

tified 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 L1 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

Flynn et al. [8] 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 to the rescue.

[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

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 panes 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 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):

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

Kalantari et al.'s [11] 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 [26] *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 Flynn et al. 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 sharing 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 [17]. 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 [23] *single-view* MPIs are 2019's Srinivasan et al. [21], Mildenhall et al. [16], and Flynn et al. [7]. Srinivasan et al. improved the quality and increased the spatial and disparity 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-fields, and blends 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. Flynn et al. [7] 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 single-view approach for generating MPIs.

2.1.3 Base Papers

Zhou et al. [26] 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 a stereo pair of images to learn an MPI prediction network in the following manner:

- The camera parameters $c_1 = (p_1,\ k_1)$ and $c_1 = (p_2,\ k_2)$ 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_i and k_i are 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 like priors/biases

- The goal is to learn a network that generates an MPI representation (Figure **??**) 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 Flynn et al. [8], an effective way to encode pose information for training is via a PSV. Hence, the input to their customized encoder-decoder type network includes a PSV version of I_2 ($\hat{I}_2$) with the planes reprojected into the final 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, again, refers to the total number of MPI depth planes (Section 1.1). 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_d, are then easily computed by taking the per-pixel weighted average of I_1 and the predicted background image $\hat{I}_b$:

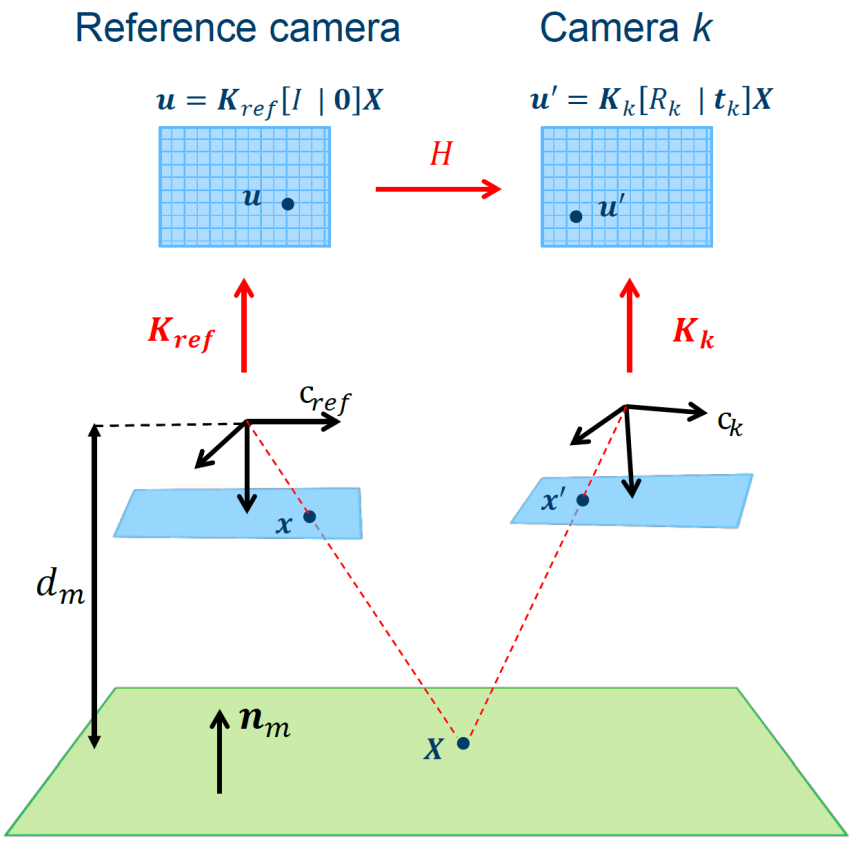
$$C_d = w_d \odot I_1 + (1 - w_d) \odot \hat{I}_b$$

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

Figure 2.2: Inferred MPI [26]

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 [25]) 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 like L_1 and L_2 norms. Adam gradient descent algorithm is used (similarly to Tucker and Snavely [23]) to optimize this loss. Adam [12] is better than regular stochastic gradient descent but is still not superior to Flynn et al.'s [7] 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 [10], 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.

Zhou et al.'s methods are ingenious in a number of ways. Some examples include them training to predict novel views at varying distances from input views so as not



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} [27].

Figure 2.3: Standard Inverse Homography or Reprojection [9]

to overfit to predicting only up to a limited number of baselines, their use of assorted but apt convolutional layers such as dilated convolutions to bring back larger scene contexts at lower computational costs and fractionally-strided convolutions [18] with skip connections [3] from preceding layers to capture even the finer texture details,

and their use of VGG perceptual loss to retain these intricate micro textures besides macro object geometries in synthesized views. Also commendable is their meticulous RealEstate10K dataset creation process which was continued by Tucker and Snavely in bringing the dataset to it's current state [26]. Knowing that state-of-the-art Structure from Motion (SfM) and bundle adjustment[6] algorithms like COLMAP [] are not yet 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 [] 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 way with COLMAP to obtain 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 just the point cloud up or down so the nearest set of points is at a fixed distance from the cameras. 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 ad-

[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 hitherto estimated camera parameters, using each additional view of the scene for feature matching as well

justment but rather proceeds to map out the entire terrain encountered by a robot by making connections between camera trajectories, viewed scenes, etc.

Some major observations made by Zhou et al. in their ablation studies include maybe mention the benefits of color blending wirghts as opposd to the other ablation studis of no FG + BG + blending weights?

larger output space and less utiization of refence image makes learning 3D scene structue harder? or just the RGB structue?

performance of model improves as more number of depth planes are used

greter the offset netween the refernce and renederd views the more planes are needed to render accurateky

2018 paper successfully applied it to soners qute difernt frm realeste 10K

but can we say the same about 2020 paper? time to find out!

extrapolating beyond the limits of the reresentaion gives a " stack of cards " effect

Tucker and Snavely [23] was the first to implement single-view view synthesis on videos in the wild. its fascinating to see how they were able to achive the coveted efficient single view view sytheis coeted by the vision and graphcs community

A special issue occurs when overseeing such a system via view synthesis, due to the uncertainty in the input data's global scale. The authors address this issue with a method of scale invariant view synthesis that makes use of sparse point sets generated during the training data generation process. The authors present an edge-aware smoothness loss that prevents depth maps created from predicted MPIs from becoming excessively fuzzy, even in the absence of depth supervision.

While the authors' purpose is view synthesis rather than depth prediction, they can easily generate disparity maps from the MPIs and use these to assess depth performance. The authors demonstrate the ability to predict MPIs for view synthesis from single image inputs without the need for ground truth 3D or depth information, and they introduce a scale-invariant approach to view synthesis that enables them to train on data with scale ambiguity, such as that derived from online video. A possible next path would be to combine MPI prediction with adversarial losses to determine whether it is possible to produce better, and more realistic, inpainting.

In the abstract they say they use scale-invariant view synthesis as supervision by which they mean they use additional views of the scene as supervision (possibly unscaled?)

Single-view view synthesis only during inference. During training 2020 model learns to predict an MPI in one stage without requiring any post-processing steps but nevertheless requires multiple images of the same scene as supervision in the one-stage training pipeline.

Single view view synthesis is an diffenert fromo the data collection aspedct of the 2018 paper becasue they retain the sparse point clouds and a record of whicvh points were tracked in each frame (refereed to as vidsible points)

how do they even come up with depth pixels for both LDI/MPI? do they already know depth from ground truth ???? and how do they calculate depth ???

we can give the number of layers to the network - does it make a difference? when you have more layers you get more depth info - more clarity in the rendered image -

somehow quantizing the depths tyhe 3d objecxt wiil fall in different plab=nes

why did they include colmap point could vs

mention about the net transmittance of the color that is beautifuly explained in the the over composition in the deep view paper

scale-invariant view sythesis for supervision PSNR AND SSIM metrics are compued on all pixels during evaluation

2020 doesn't require ground-truth 3D or depth predicted BG image to inpaint what lies behind foreground objects

premultiplied alpha color images needs to mentioned from deev view page 2369

Figure 2.4: Fronto-parallel planes in MPI layered representation

Fig 3. Example of fronto-parallel planes

However, in fact, the coordinates of picture dots cannot be determined with arbitrarily high precision. Rather than that, other sources of noise, such as geometric noise caused by lens distortion or interest point recognition mistake, cause inconsistencies in the picture coordinates that are measured. As a result, the lines created by the associated picture points do not always meet in three-dimensional space. The objective is then to identify a three-dimensional point that fits the observed picture points optimally. Numerous approaches exist in the literature for how to define optimality and how to determine the optimal 3D location. Due to the fact that the various approaches are based on distinct optimality criteria, they generate significantly varied estimations of the 3D point x when noise is present.

Depth maps are derived from the predicted MPIs and not predicted in parallel with the MPIs

2020 paper uses the same data curation process as the 2018 paper

Chapter 3

METHODS

3.1 Approach

3.2 Hypotheses

hypotheses of this thesis are as follows: 1) Te MPI 2020 paper, which produces good disparity maps for ¡the opposite of close-up shots¿ ¡long shots of objects/landscapes¿ [], is unable to do so for close-up shot of people/objects. When we re-recreated the 202 Hence, our first hypothesis is that re-retraining the re-created 2020 MPI model on

we appreciate the authors for precaution ing us in extending our dataset with the realestaTE 10K DATRAUSET

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Stereo Magnification (2018) paper introduces the MPI representation and also explains how the data was processed. Some parts of the code are refactored and reused in the 2020 MPI paper like how the 2018 paper loads data in: in particular the data loader (loader.py, datasets.py) does subsequence selection and some slight random cropping.

richard tucker email very very important:

In terms of relation to the previous MPI papers: Stereo Magnification (2018) introduces the representation and also explains how the data was processed. The code

itself is separate but you could look at how it loads data in: in particular the data loader (loader.py, datasets.py) does subsequence selection and some slight random cropping.

From the code we have released (https://github.com/google-research/google-research/tree/master/s
$ourthenetworkdefinition(convolutionallayers, kernelsizes, etc)(nets.mpi_from_image)howtorende$

What you don't have is (a) the implementation of the losses, and (b) data including point clouds and a way to load it in. One of the key points of Single-View View Synthesis was to use sparse point cloud data to make the view synthesis loss scale-invariant. To obtain such data you would have to process the mannequin dataset to obtain point cloud and/or depth, for example with COLMAP (maybe you have already done that), and then you could write a data loader or extend the one from Stereo Magnification.

Is your Mannequin Dataset this one: https://google.github.io/mannequinchallenge/www/index.htm
If so, a note of caution: it's quite a bit smaller than RealEstate10K, and so there is a risk of overfitting. (I have tried training on something like it myself, and I found that it was better to use a combination of both RealEstate and MannequinChallenge than just MannequinChallenge alone). But anyway, that's something to worry about after you are able to train.

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Whereas the 2018 model claioms to gebreraLIZER TO OTHER DATASEYT WIER-HOUT RETREAINGU, no suvk ckaimsd gave been madfe by the 2020 model

When we ran the inference part of the 2020 Single-View MPI model [1] on a video chat frame and found that the generated disparity map (a by-product of MPI inference in this network architecture) was visually inaccurate, it became the starting off point for our thesis. Comparatively, the inferred disparity map would be much more visually accurate when a real estate video frame was processed. The latter outcome is to be expected because the 2020 Single-View MPI model was exclusively trained on the RealEstate10K video dataset.

3.2.1 Problem Statement 1

How to increase the 2020 Single-View MPI model's depth prediction accuracy for video chat frames or any other video-chat-relevant image with close up shots of people.

As inference was one of the only parts of the network made publicly available by the authors on GitHub due to the proprietary nature of some other aspects of their code, the first step in addressing problem statement 1 was to recreate the 2020 Single-View MPI training procedure.

Recreating and retraining the network involved the following:

- Taking the readily available network information from the paper — like details about the various types of fully-convolutional encoder-decoder layers involved, etc. — and putting it in place with other aspects of the network that called for a more involved recreation process like the data loader part and the loss function.

[1] Deep Learning terms such as model, network, and CNN have been used interchangeably throughout this thesis.

- Processing all the videos from the required datasets with an ORB-SLAM2 package called COLMAP to extract the 3D point clouds of the scenes in each video frame which makes up one-half of the input to the training pipeline, the other half being the actual video frames themselves.

- Attempting to train the network on only the video-chat-relevant Mannequin Challenge video dataset [14] which is ∼90% smaller than the RealEstate10K dataset.

- Fixing the programming errors encountered during training.

- Retraining the recreated model on not just the Mannequin Challenge dataset but also the RealEstate10K dataset by clubbing both datasets together to be used as a singular source of input to the recreated 2020 MPI model.

The main reason for extending the Mannequin Challenge dataset with the originally used RealEstate10K dataset was so Hypothesis B of this thesis could be proved: —

Hypothesis A: The recreated model trained only on the video-chat-relevant Mannequin Challenge Dataset performs as well as the 2020 MPI model does on real estate data, when it comes to accurately predicting MPIs and disparity maps of video-chat-relevant frames involving close up shots of one or more persons.

Hypothesis B: If hypothesis A is disproved[2], the recreated MPI model should perform as well as the 2020 MPI model if the Mannequin Challenge training data were extended by the RealEstate10K dataset.

[2]Please refer to the Experiments and Results Chapter.

3.3 Data

for the 2018 paper During training, the images and MPI have a spatial resolution of 1024×576, but the model can be applied to arbitrary resolution at test time in a fully-convolutional manner. but maintaining resolution for 2020 paper is not required

4.3 Refining poses with bundle adjustment We next process each sequence at higher resolution, using a standard structure-from-motion pipeline to extract features from each frame, match these features across frames, and perform a global bundle adjustment using the Ceres non-linear least squares optimizer [Agarwal et al. 2016].

The steps taken to get both Mannequin Challenge and RealEstate10K datasets ready for training are as follows:

errors in our retraining attempts despite the authors of the preexisting model indicating in their paper that the model could be trained on a hodgepohdge of multiple resolutions

- Each dataset from the URLs listed in [26] and [14] was downloaded into the current working directory. Both these datasets consist of .txt files pertaining to each video that can be downloaded. Each .txt file begins with the downloadable video's YouTube link on the first line. And, from the second line onwards, it continues with listing the details of ORB-SLAM2 processed frames from the video [3] — like the timestamp (in microseconds), camera intrinsics, and camera extrinsics.
- The GitHub repository associated with this thesis was downloaded from `https://github.com/au001/view-synthesis.git` into the working directory.

[3] with one line for each frame

- The project's comprehensive Dockerfile was built from within the cloned repository by running *build-run-docker/build-docker.sh*. Considerable time was spent to resolve dependency-version compatibilities by consulting the build log file whenever Dockerfile failed to build. After a successful build, the final docker container was started with *build-run-docker/build-docker.sh*.

- Back inside either downloaded dataset directory, we ran the script *../view-synthesis/scripts/get_videos.py* on *train/* folder to download all videos with youtube-dl at 720p resolution. Alternatively run *../view-synthesis/scripts/get_videos_aria2c.py* to bolster youtube-dl's download speed with Aria2 download manager. Standard output was saved to a log file to address possible download errors.

- Inevitably youtube-dl would throw download errors on the first run as there would be some 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 with the aim of maintaining consistency. Although scaled videos should not pose any problem to the training or the 3D point generation with COLMAP, we proceeded to attempt something different from the 2020 MPI authors, assuming the permitting scaled videos in their pipeline.

- As of this writing 2663 of ∼3000 Mannequin Challenge videos and 67582 of ∼70000 RealEstate10K videos were downloadable at 720p resolution and with no download errors on the first attempt. Hence, it became imperative to also save all the outputs generated by running the previous script to a log file and work upon it fix all download issues.

Descriptions of Mannequin Dataset, Real Estate dataset, COLMAP processing pipeline, data loader —

COLMAP There is a correspondence between the camera location C in world coordinates the distsnce lambda from C to the objerved 3D world point in the scene along the viewing ray of a projected pixel x on the imaging

COLMAP is a 3D scene reconstruction pipeline. It attempts to recover the 3D scene structure from unstructured 2D images of the scene that come with no prior knowledge of the camera intrinsics, extrinsics, and nature of objects captured in the image. 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.

feature detection —¿ pairwise feature matchng —¿ correspondense estimation —¿ incremental structure from motion

We had to make sure that we subejct videos to colmap procesing only after ensuring that ther 720p version og them were donloaded and for videos that were missed during handling of these large datasets we hasd to make sure that after properluy redownlioafuinhg the video again we colmap processed ti again

3.4 Implementation

please complete richards conversation to glean more details about implementation

Richard Tucker emails about implementation details:

prof ventura: I think that must be it – I am running it in eager mode. We will work on switching over. Thanks again, Richard.

richard:

Our training setup was rather different: we used the old tf Estimator system as we had a configuration to allow us to do distributed training on ten workers.

But even on a single worker, our gradient calculation took less than a second. I wonder if maybe you are doing everything in eager mode so there is a lot of overhead? Using keras's model.fit or the old estimator system, or just wrapping things in tf.function, should allow the critical parts to run in graph mode ought to be faster.

Another possibility is things are too big to fit on your GPU. We used a batch size of 4, if I remember correctly.

prof ventura:

Okay, thanks, Richard. At least I had the right intuition in thinking that might be an issue!

We are able to train the model now and the loss does go down which is great. However, the gradient calculation

grads = tape.gradient(loss, model.trainable$_w eights$)

takes about one minute! Is that what you experienced as well? I am using a batch size of 8 and this is on an Nvidia V100.

richard:

Training details. We implement our system in TensorFlow [Abadi et al. 2016]. We train the network using the ADAM solver [Kingma and Ba 2014] for 600K iterations with learning rate 0.0002, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and batch size 1. During training, the images and MPI have a spatial resolution of 1024×576, but the model can be applied to arbitrary resolution at test time in a fully-convolutional manner. Training takes about one week on a Tesla P100 GPU.

They didn't even use multiGPU

issue happened is the resolution had to be fixed and colmap had to be run again

difference between train.py throws NaN gradient errors np.loadtext thingie and np.generatefromtext thingie such that one possibly inclused empty and the other discards copletely?

training just takes +1 frame to be the target only testing takes +5 or +10 frames according to the 2020 paper

Expanding upon pipeline description —

video ————¿ input 1 ————¿ training — — ————¿ COLMAP ——¿ 3D points per video frame ————-¿ input 2 ———¿ same training

how does data loader work?

Original authors choice with colmap my reason for pursuing colmap what does colmap do¿ ——¿ bundle adjustment, triangulation,

COLMAP highlevel

Triangulation Bundle Adjustment

Chapter 4

EXPERIMENTS AND RESULTS

4.1 Experiments

Here are some of the quantitative and qualitative evaluations of the variants of the recreated 2020 Single-View MPI model trained on different combinations of the Mannequin Challenge and RealEstate10K datasets. We use the pretrained weights of the 2020 MPI model as the benchmark and compare the abilities of all models at hand to generate novel views (more extensive training underway). We adopt some of the quantitative metrics from the 2020 MPI paper [23] — PSNR, SSMI [24], and LPIPS [25] — 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 following are the model variants used to compute the metrics stated above to help provide arguments for or against each of the hypotheses stated in section 3.1: —

- The pretrained weights of the 2020 single-view MPI model trained exclusively on RealEstate10K data.
- The recreated 2020 model retrained exclusively on the Mannequin Challenge dataset with transfer learning using the pretrained weights of the original 2020 MPI model. In this transfer learning process, none of the layers of the pretrained weights were frozen and so could learn and evolve based on the Mannequin Challenge data they were newly exposed to.

- The recreated 2020 model retrained not just on the Mannequin Challenge dataset but also on the RealEstate10K dataset with similar transfer learning as in the previous variant. This variant was encouraged to be tried out by one of the authors of the 2020 MPI paper in our correspondences with him. [23]

We sifted through the test set of the Mannequin Challenge dataset to hand pick a set of 333 videos that contained ORB-SLAM2 recognized sequences[1] which had video-chat-relevant features like the head and torso of people being focused on rather than having wide shots of entire bodies, the number of people in the frame being mostly limited to one or two as opposed to multiple people being featured, and the head pose of people being roughly or even very loosely aligned with the camera (there was hardly anybody that looked directly at the camera). We put them in the test-yes/ bin. We also curated test-maybe/ (311 videos) and test-no/ (25 videos) bins that consisted of the rest of the Mannequin Challenge test set with sequences either having no relevance to video chat (like there being hardly anyone in the frames) for test-no/ to those falling heavily in the gray areas between test-yes/ and test-no/ for test-maybe/. We even occasionally interspersed the test-yes/ and test-maybe/ bins with videos containing sequences that portrayed people facing diametrically opposite the camera just to 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 procured from both the 2020 Single-View and 2018 Stereo MPI papers[2] like `generator_test.py`, `generator_train.py`, `data_loader.py`, `train.py`, and `test.py`, the scripts relevant to the experiments in this section are test.py and generator_test.py. For testing, the generator first aggregates all videos names from the

[1] the timestamp, camera intrinsics and extrinsics of all frames of each of which are listed in the corresponding .txt files in the dataset

[2] Please find our code repository at `https://github.com/au001/view-synthesis.git`

directory input to it and for each of these, it picks reference_image and target_image to be either 5 or 10 frames apart. reference_image is the frame that test.py uses to infer the MPI 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 rare as both datasets have been curated by a similar ORB-SLAM2 process like COLMAP. In such hypothetical cases, target_image will be erroneously rendered by mpi.render() in 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. We first test and compute metrics of frames 5 part and then we repeat the same test process for frames 10 apart just to show (as in the case of the 2020 MPI paper) that the longer the baseline between reference/source and target views, the less the accuracy will be of the rendered image. Moreover, we also calculate the metrics for all processed reference_image, target_image pairs, to catch the hypothetical anomalies of complete scene changes mentioned above.

Another jewel in our project was our attempt to parallelize training across multiple GPUs, which we believed would allow us to increase the batch size — currently limited to 4 pairs of reference and target images along with their respective camera poses, intrinsics, and 3D points of the reference image — and thereby let larger parts of our 70000+ training ready sequences with associated point clouds be used for learning by our recreated model. 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 — like the dataset generator,

the loss functions aggregator, etc. — spread across GPUs, our training got undercut after a good start by the following error at training step 178:

```
tensorflow.python.framework.errors_impl.
    ResourceExhaustedError: OOM when allocating tensor
    with shape[1,32,512,512,3] and type float on /job:
    localhost/replica:0/task:0/device:GPU:0 by allocator
    GPU_0_bfc [Op:StridedSlice] name: render/
    compose_back_to_front/strided_slice/
```

Nevertheless, we computed all three metrics for the second model variant retrained on Mannequin Challenge data but this time with `tf.distribute.MirroredStrategy`, harnessing the power of multiple GPUs.

4.2 Results

The figures throughout the paper are best seen by zooming in to the electronic version of the paper.

The results for the experiments done on the baseline pretrained model and the variant retrained on only the video-chat-relevant Mannequin Challenge dataset are presented in this section. As of this writing generator_test.py is only able to pick random pairs of reference and target frames from the 333 test-yes/ videos. The results for sequential pair picking, which would avoid possible repetition of picked pairs and allow for an exhaustive coverage of the test set, will replace the results below in a subsequent manuscript of this thesis shortly alongside the results for the variant trained on both the Mannequin Challenge and RealEstate10K datasets and more visually applealing/insightful graphs. Moreover, the number of training iterations/steps will

Model Variant	Dataset(s) (re)trained on / No. of Videos	**LPIPS ↓** target_image vs rendered_image		**LPIPS ↓** reference_image vs target_image	
		n = 5	n = 10	n = 5	n = 10
Pretrained	RealEstate10K / ∼70k	0.418	0.525	0.446	0.555
Recreated	Mannequin Challenge / 1841	0.319	0.433	0.446	0.555
Recreated	Mannequin Challenge + RealEstate10K	–	–	–	–
Recreated multi-GPU	Mannequin Challenge	0.418	0.525	0.446	0.555

n refers to the distance between the reference and target frames picked by the generator. Retraining promises marked improvement over original pretrained model.

Table 4.1: LPIPS Mean Values

be much more than the current 10k steps which is projected to increase the accuracy even more.

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 or 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

These metrics mean value tables produced via the random reference, target pair experiments so far validate hypothesis A of this thesis by showing that transfer learning with completely unfrozen layers seems to be helping the 2020 Single-View MPI pick up

Model Variant	Dataset(s) (re)trained on / No. of Videos	**SSIM ↑** target_image vs rendered_image		**SSIM ↑** reference_image vs target_image	
		n = 5	n = 10	n = 5	n = 10
Pretrained	RealEstate10K / ∼70k	0.549	0.492	0.418	0.370
Recreated	Mannequin Challenge / 1841	0.560	0.494	0.418	0.370
Recreated	Mannequin Challenge + RealEstate10K	–	–	–	–
Recreated multi-GPU	Mannequin Challenge	0.549	0.492	0.418	0.555

n refers to the distance between the reference and target frames picked by the generator.
Retraining promises marked improvement over original pretrained model.

Table 4.2: SSIM Mean Values

from where it left off by specializing and improve upon its performance in accurately predicting close up shots of people in video-chat-relevant frames.

A further testimony to this improvement can be obtained by inspecting the performance of even the prematurely halted multi-GPU variant. It performs at par with the original pretrained model which indicates that the pretained model has begun to continue where it left off and specialize in processing video-chat-like frames. It would have run properly if not for the resource errors mentioned in the previous section that could point to underlying issuess like possible unchecked growth of TensorFlow graphs per pipeline replica or something else. This seems to be the case even though the replicas seem to be getting properly allocated inputs and their respective outputs also seem to be getting well gelled together in the end.

Model Variant	Dataset(s) (re)trained on / No. of Videos	PSNR ↑ target_image vs rendered_image		PSNR ↑ reference_image vs target_image	
		n = 5	n = 10	n = 5	n = 10
Pretrained	RealEstate10K / ∼70k	16.105	14.088	13.185	11.589
Recreated	Mannequin Challenge / 1841	16.729	14.664	13.185	11.589
Recreated	Mannequin Challenge + RealEstate10K	–	–	–	–
Recreated multi-GPU	Mannequin Challenge	16.105	14.088	13.185	11.589

n refers to the distance between the reference and target frames picked by the generator.
Retraining promises marked improvement over original pretrained model.

Table 4.3: PSNR Mean Values

The authors of 2020 MPI paper used the following pointers like the handling of occluded content, the production unpleasant artifacts at the edges of foreground objects, and so on to qualitatively compare the discrepancies in the results generated by each variant model. In addition to visually checking for these, we, like the authors also found that visually checking the disparity maps is also useful in verifying the quality of the MPI produced.

(a) reference (b) target (c) disparity (d) rendered

(e) reference (f) target (g) disparity (h) rendered

(i) reference (j) target (k) disparity (l) rendered

(m) reference (n) target (o) disparity (p) rendered

First Row: Pretrained Model – tested 5 frames apart; Second Row: Retrained Model – trained on 1841 Mannequin Challenge videos – tested 5 frames apart; Third Row: Pretrained Model – tested 10 frames apart; Fourth Row: Retrained Model – trained on 1841 Mannequin Challenge videos – tested 10 frames apart

Figure 4.1: Model Variants' Output Visualizations

Chapter 5

CONCLUSION

As hypothesized, the PSNR and SSIM metrics of the baseline model compared to the retrained model show that there is a slight improvement in the performance of a model trained exclusive on the non-video-chat-relevant real estate data to the much-more-video-chat relevant Mannequin Challenge data

The qualities of the rendered image and the predicted MPI have been improved first by going from training exclusively on real estate data to going to exclusive training on mannequin data and then finally onto a mix of both. We complete the one-way part of a two-way potentially real-time rendering pipeline that takes in the head pose of each source video frame and renders the corresponding target video frame i.e., the one that syncs with the timestamp of the source in this head pose.

5.1 Future Work

Maybe we could implement taking the average of the head poses of multipe people in the video frames of video conferences instead of just video chat and make their average head pospe change the scene rendering viepoint of teh secne to be rerenered.

Through this thesis, I had the opportunity to form a 2-way pipeline that is able to render new views from the perspectives of both the participants in Video Chat conversation.

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 train-

ing with TensorFlow/Keras. This would allow for feeding a lot more images/video-frames to the model, which would further reduce the accuracy of the model.

Using Grad-CAM to locate the bottlenecks in the recreated MPI neural net to optimize hyperparameter tuning for producing more accurate results, esp. predicted depths / disparity. https://www.pyimagesearch.com/2020/03/09/grad-cam-visualize-class-activation-maps-with-keras-tensorflow-and-deep-learning/

Render in both directions, making the pipeline two-way and then proceed to make it realtime by involving a game engine or any other framework capable of realtime rendering.

Try training on variable resolution video frames and not all just 1280x720 ones

Ideal for a headless server?

make use of docker multistage builds to have all components in a single dockerfile with multiple froms like tf/tf-gpu-2.2 as well as nvidia-cuda10.2-devel-ubuntu18.04

possible hypothesis: did cuda support improve disparity map

possible hypothesis: cuda gpu support is possibly not required for OpenFace inference

features of OPenFace like head pose estimation may still work without CUDA recognition by the server either COlab or El Capitan

need gpu support for maybe mpi training alone and not any other components of the pipeline as the rest of the pipeline is just inference

COLMAP will be quicker with GPU https://colmap.github.io/faq.htmlavailable-functionality-without-gpu-cuda

major hypothesis: one major reason with disparity map to be less because the batch size was only 4 frames at once if multi-gpu access were available then disparity map would have been better

Mainly mpi and maybe even colmap (for inference speed) seem to require GPU/CUDA support. I've been trying to get GPUs to be used by all my packages on Docker like MPI, COLMAP and their dependencies OpenCV, Dlib etc. I doesn't seem to work yet. So I'll resort to using these packages on Docker without GPU support for now.

in videos we have recording of my insights today - 8/15/21 been falling behind and didn't report until resukts

and update to flagship versions

the main thing Dr ventura is that the CUDA install was broken and I needed it for multiple programs like dlib, openface, notwithstanding colmap

ask prof ventura to update the nvidia drivers

why did i go off on a tangent? https://stackoverflow.com/questions/43022843/nvidia-nvml-driver-library-version-mismatch Available functionality without GPU/CUDA

https://colmap.github.io/faq.htmlavailable-functionality-without-gpu-cuda If you do not have a CUDA-enabled GPU but some other GPU, you can use all COLMAP functionality except the dense reconstruction part. However, you can use external dense reconstruction software as an alternative, as described in the Tutorial. If you have a GPU with low compute power or you want to execute COLMAP on a machine without an attached display and without CUDA support, you can run all steps on the CPU by specifying the appropriate options (e.g., –SiftExtraction.use$_gpu$ = $false for the feature extraction step). But note that this might result in a significant slow-down of the reconstruction pipeline. Please, also note that feature extraction on the CPU can consume ee

$-SiftExtraction.max_image_sizeand/orsetting--SiftExtraction.first_octave0orbymanuallylimit$
$-SiftExtraction.num_threads.$

nvidia-smi Failed to initialize NVML: Driver/library version mismatch

what is cuda and nvcc all about? https://varhowto.com/check-cuda-version/

dockerfile colmap run needs to be explained with video clip in MAnnequinChallenge

proof that gpu is being used by colmap in images in MannequinChallenge

https://linuxize.com/post/linux-time-command/ time all functions

make sure I'm able to restart the model at any poit and continue traning whre I left off and add datasets

I need info on inference code

Ask ventura why did yoyu say the disparity was bad

Hopefully successfully able to use the latest versions of all components of the mpi pipeline for both training and inference

Another application would be if we have a VR headset with a camera on it we can track the rotation of the camera and by doing that you're tracking the rotation of the person's head so that you can render the VR content at the right angle

Ask prof ventura is colmap autmatically redoes all error videos Ask pro ventura about copyright for his own epipolar geometry lectures

stereo = 2 images pretty close to each other paired in a special way so that you can get really dense estimations of the depth so basically for every pixel you could get a depth estimate rather than some sparse sampling of keypoints canonical stereo case

only have pure horizontal translation and no rotation and no other translations in Y or Z

blueer values are closer and redder values are farther away in disparity maps

check 7000 train set and 1400 test set of 2018 paper

why doesn't 2020 and 2018 papers employ SLAM algorithms directly from COLMAP and not indorectly by themsellves or are they refereing to the same slam algorithms

overfitting can be further reduced by using a cnn in the place of a gradient descent algorithm like flynn et all deep stereo 2019 i.e., essentially combining 2020 paper with this predecessor chaekc the first chaPpter of interduction of 2019 deep stereo for more info abou this 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.

Actually, the DeepView paper has a beautiful software to customized, visualize, and render any type of MPI! It's kind of like the state of the art MPI manipulator. `https://augmentedperception.github.io/deepview/`. So maybe improve the 2020 MPI html visualizer upto the standarsd of the deep view one or atleast use it to tweak and experiment the various MPI paraeters lik number of layers etc before deploying and training and testing.

Adam is better than regular stochastic gradient descent but still not superior to Flynn et al.'s [7] implementation of learned gradient descent.

from deep stereo paper the causes for overfitting

Deep networks have enjoyed huge success in recent years, particularly for image understanding tasks [20, 29]. Despite these successes, relatively little work exists on ap-

plying deep learning to computer graphics problems and especially to generating new views from real imagery. One possible reason is the perceived inability of deep networks to generate pixels directly, but recent work on denoising [35], super-resolution [6], and rendering [21] suggest that this is a misconception. Another common objection is that deep networks have a huge number of parameters and hence are prone to overfitting in the absence of enormous quantities of data, but recent work [29] has demonstrated state-of-the-art deep networks whose parameters number in the low millions, greatly reducing the potential for overfitting.

from deep stereo on the success of neural nets

In this work we present a new approach to new view synthesis that uses deep networks to regress directly to output pixel colors given the posed input images. Our system is able to interpolate between views separated by a wide baseline and exhibits resilience to traditional failure modes, including graceful degradation in the presence of scene motion and specularities. We posit this is due to the end-to-end nature of the training, and the ability of deep networks to learn extremely complex non-linear functions of their inputs [25]. Additionally, although we focus on its application to new view problems here, we believe that the deep architecture presented can be readily applied to other stereo and graphics problems given suitable training data. Because of the variety of the scenes seen in training our system is robust and generalizes to indoor and outdoor imagery, as well as to image collections used in prior work.

BIBLIOGRAPHY

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

[2] Svetlana Lazebnik, 2019.

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

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

[5] 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.

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

[7] 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.

[8] 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.

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

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

[11] 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.

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

[13] 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.

[14] 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.

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

[16] 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.

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

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

[19] 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.

[20] 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.

[21] 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.

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

[23] 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.

[24] 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.

[25] 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.

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

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