An Open Source Volumetric Capture System for Production Applications

Aidan Montag, Flip Phillips
Motion Picture Science
Rochester Institute of Technology
Rochester, New York, United States
ajm3312@rit.edu
flip.phillips@rit.edu

***Abstract*—We outline engineering and design decisions involved in developing a volumetric capture array system to complement the rise of "light field optimization" techniques, which leverage advances in machine learning to iteratively refine a 3D representation, based on captured 2D images of a scene. We explore approaches to camera calibration, including acquisition parameters such as intrinsic and extrinsic information, and creating simple 3D representations of the scene to act as initialization points for various volumetric capture methods. We also present a modular hardware design for an inexpensive, scalable, and upgradable camera array. Finally, we discuss the strengths and weaknesses of various approaches to volumetric capture, and how they can be applied to different production scenarios.**

I. Introduction

Volumetric Capture is one of the most exciting contemporary fields of research for production applications. Traditional cameras capture a two-dimensional projection of our three-dimensional world. In that projection, a large amount of information about the depth of a scene is lost. Volumetric capture seeks to restore that z axis information to the image. There are many approaches for capturing data necessary for a 3D reconstruction of a scene, many of which are decades old. The CT scanner was invented in the 1970s and captured volumetric information by taking layered X-rays of the subject, then stacking those layers on one another, resulting in a three-dimensional, volumetric image [1]. Meanwhile, other implementations resemble standard cameras more closely, such as the Lytro light-field camera, which uses a standard optical configuration with the addition of a microlens array to measure the incident angle of light hitting its sensor, which can then be converted to depth information [2].

Working with a 3D image allows for numerous additional creative possibilities in a production environment, compared to a 2D image. On set, it can enable a director to focus more deeply on the blocking and performances of their cast, knowing camera framing and motion can be offloaded to the post-production phase, or enable complex camera movements and positioning that would otherwise be impractical or impossible with a dolly or motion control system. In visual effects workflows, it allows the composite between the practical photography and synthetic imagery to be made in a 3D space instead of a 2D one, increasing flexibility for blending and relighting. Instead of a compositor needing to carefully align perspectives, matte subjects from a green screen, and color match the background, a volumetric subject can simply be 'dragged and dropped' into a 3D scene. Finally, with respect to displays, volumetric capture enables new opportunities for immersive, interactive viewing experiences. With the use of a virtual reality headset or CAVE, the viewer can step into the world of a film and explore it with a full six degrees of freedom, unlike more spherical immersive media cameras, which only allow three degrees.

Our work is mutually inspired by our research and production needs. Foundational work in volumetric imaging techniques are described in [3], [4]. These, in turn, evoke Gibson's Optic Array [5], [6]. Light fields are a crucial component of each of these approaches to understainding the both the synthetic creation *and* perception of our 3D visual world.

II. Light Field Optimization

For the present work, we investigate the construction of a system for a particular form of volumetric capture which we refer to as "light field optimization". This technique iteratively refines a digital 3D representation of a scene by minimizing the loss between a series of captured 2D images and corresponding 2D rasterizations of the synthesized representation. While the underlying approach to the optimization may vary, the most significant distinguishing factor between light field optimization methods is usually the 3D representation of choice. *Neural Radiance Fields* [7] represents one of the first light-field optimization techniques to gain prominence. While using optimization techniques to refine a 3D representation of a scene is not a new concept, [7] was one of the first to leverage advances in machine learning architecture, creating a more complex 3D representation that was perceptually inspired. In the case of a NeRF, the 3D representation is *itself* a neural network.

Since [7], much research has focused on finding a 3D representation that falls more in line with traditional 3D animation and rendering tools. The most popular production method is still traditional polygonal modeling, due to its computational efficiency, deformability, and the self reinforcing result of a wide range of toolsets being created for it due to that efficiency and deformability. Ideally, to integrate volumetric capture seamlessly into a traditional visual effects,

animation, or game design workflow, its representation should be as similar to the standard modeling workflow as possible. However, building a light-field optimizer to output a traditional 3D polygonal model has proven to be a complex task. High-quality topology requires delicate design and intentionality, attributes that optimization methods are typically unsuited for. Some implementations convert a pre-optimized representation, like a neural radiance field, into polygonal geometry; however, these conversions are never lossless and no longer represent the original light field.

Methods such as Marching Cubes [8] provide a bridge between volumetric representations $X^{(3)} \subset \mathbb{R}^3$ and traditional surface meshes $X^{(2)} \subset \mathbb{R}^3$. In the context of volumetric capture, this process reconstructs the 3D volume $X^{(3)}$ from a set of 2D image observations $I_i \subset \mathbb{R}^2$. Once the volume is represented by a spatial density or signed distance function $f : \mathbb{R}^3 \to \mathbb{R}$, the boundary surface mesh $X^{(2)} \approx \partial X^{(3)}$ is extracted as an isosurface at threshold c:

$$X^{(2)} = \{x \in \mathbb{R}^3 \mid f(x) = c\} \tag{1}$$

However, these approaches are fraught with their own challenges. For example, the architecture of the original Reyes rendering system [9] supported complex geometry via the "micropolygon" representation, but it wasn't until later incarnations of the RenderMan architecture [10] that it fully integrated comprehensive volume support into the rendering pipeline [11].

The source of this volumetric $X^{(3)}$ data is typically generative, but more recently we have been able to capture these volumes from real-world environments by resolving multiple 2D camera projections, providing a rich source of content for 3D production.

Where the most public interest has been placed recently is in 3D representations that can meet in the middle. A "Gaussian Splat" [12] takes its roots from the point cloud created, i.e., the vertices of a traditional mesh. Each point in a Splat carries additional parameters to its 3D transform, such as rotation, scaling, opacity, color, and some degree of spherical harmonics, a function to approximate radiance variation by change in angle. While this method introduced new artifacts compared to Neural Radiance Fields, particularly perceptual artifacts in low-frequency areas, it can take advantage of various 3D rendering and GPU acceleration techniques, which makes it quicker to iterate and therefore quicker to run the output model. While it is not "plug and play" with traditional 3D renderers, a Gaussian renderer can also be more easily implemented in preexisting software. Additionally, implementations built upon the Gaussian, like triangle splatting [13], simplify the representation further, placing a three-sided polygon at each point, which can also vary in shape and color. Unlike a Gaussian, this implementation is plug-and-play with traditional renderers, but comes at the cost of even stronger artifacts.

While there is vast general excitement for these optimization techniques, there is little general consensus on a standardized method to be adopted in production workflows. Rightly so, as each carries strengths and weaknesses dependent on its placement on the spectrum between traditional toolset friendliness and optimization friendliness. This uncertainty becomes more extreme as we introduce motion to the equation. Most implementations are designed with static scenes in mind, and expanding them to play sequentially takes considerable forethought into maintaining temporal consistency and computational efficiency. The purpose of this project is to build a test-bed, one that can serve to experiment with the current spectrum of implementations and be adapted to support new ones, to help determine their various strengths and weaknesses for the specific application of performance capture in production. Since the variation in the light-field optimization method is primarily determined by the 3D representation output, the data collection requirements for input are generally agnostic. Those requirements are high-quality imagery from multiple views of the scene, precise camera intrinsic values, precise camera extrinsic values, and a simple geometric representation of the scene to act as an initialization point for the optimization function.

Fig. 1. Example Images from Synthetic Dataset

III. System Requirements

In general, the more viewpoints a volumetric system can capture, the higher the quality of the reconstruction. The number of cameras necessary to achieve a high-fidelity capture depends on the scale of the space, the placement of cameras, the quality of the initialization point cloud, data processing, and model training techniques. In our case, our capture suite will be an 18x20-foot volume located in our motion capture lab. To investigate the necessary number of cameras for a satisfactory capture, we reconstructed the laboratory synthetically and rendered datasets with virtual 8, 16, and 28 camera rigs, assembled in a hemisphere along the walls and ceiling of the room. Since our system will prioritize capturing human performance, we placed a synthetic human in the center of the room to act as our subject. These synthetic datasets also served to help develop our data processing algorithms, which meant much of our software could be completed before making a hardware investment.

To reconstruct our synthetic data, we used Postshot's Splat3 Gaussian splatting implementation. Our capture system is designed to be agnostic to the reconstruction method chosen; we chose Splat3 as it well represents popular contemporary reconstruction techniques. To best represent the capabilities of our real system, the ground truth camera extrinsics and intrinsics from the rendering pipeline were discarded and

Fig. 2. Comparison of Reconstruction Quality with Increased Capture Density

replaced with calculated parameters from our own software. Additionally, we performed a simple noise characterization of the time-of-flight cameras we used at the time to degrade the rendered depth maps and better represent realizable data. Nevertheless, there are many additional noise and error factors that we have not accounted for in our synthetic tests, and the output results should be considered a best-case scenario for a real system of our configuration.

We can see that as the number of cameras is increased, detail stays relatively constant; however, spatial cohesion improves dramatically. In many ways, the sampling characteristics of a volumetric capture array act similarly to those of an imaging sensor. Similar to how a sensor with a small imaging area per pixel is more prone to aliasing, the wider gaps between each camera create a higher sampling error between viewpoints.

When observing the reconstruction from a captured viewpoint, the quality becomes perceptually unaffected by the number of additional cameras in the array. This illustrates a fundamental concept of machine learning: the wider the range and the more diverse the training data is, the more flexible the output model will be. The reconstruction improvement when viewed from the camera's vantage point in the lower-numbered arrays implies that these models are "overfit" and are unable to account for new variation in the data. From these tests, we concluded that 16-26 would deliver satisfactory quality for an initial array.

IV. Camera Calibration

A. Bundle Adjustment

Once we had determined the number of viewpoints necessary for a high-quality capture, we began building a workflow to acquire the additional input parameters for optimization. For static captures, this process is typically handled using a method called "bundle adjustment" [14], usually with an implementation called COLMAP. The first step in this process is feature tracking, where 2D points within the scene are tracked from frame to frame sequentially; each 2D point is assumed to correlate with a static 3D point. In a single pass, a structure-from-motion bundle adjustment process will initialize the intrinsic parameters of each camera (focal length, lens center, distortion coefficients), extrinsic parameters of each camera (rotation and translation in 3D space), and the static

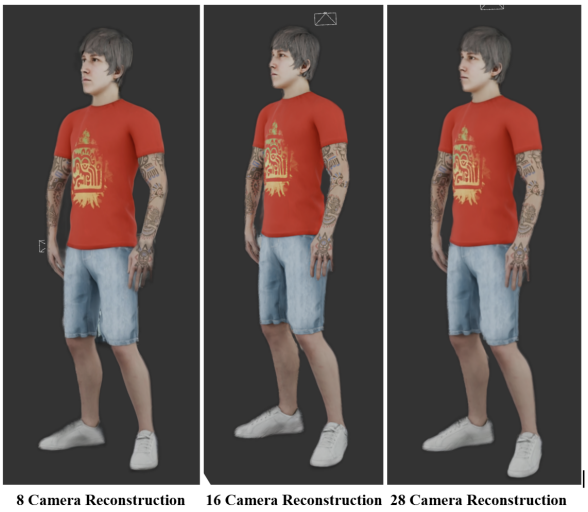

Fig. 3. Comparison of Reconstruction Quality from Sampled Viewpoint

3D positions for each tracked point. Each of these parameters will be refined iteratively using least-squares regression until the reprojection error between the sampled 2D coordinates of the imaged points and the projected 2D coordinates of the estimated 3D points is minimized.

This process is capable of achieving extremely high-accuracy results, with sub-millimeter precision for camera positions; however, there are important limitations that make it ineffective for a sparse camera array. First, the offset of each sequential camera must be relatively small. Point tracking algorithms function by searching for a similar block of pixels within a specific radius of the original point. Larger camera motion requires larger search areas, which increases processing time and the likelihood of false positives. The density of input points can also be scene-dependent, as images dominated by low frequencies will have fewer high-contrast points to track. Second, bundle adjustment relies on having dense input data to prevent overfit scene representations. Most photogrammetry and volumetric capture systems that rely on bundle adjustment assume a single camera is moved around a space for 100–300 viewpoints. This means that only one intrinsic parameter set must be estimated, with plentiful views and points to help validate the model.

Since our system will have a tenth of the number of views necessary for reliable structure-from-motion techniques, we instead needed to develop our own process. Fortunately, the nature of the system gives additional resources and flexibility in how we approach this. First, it is not a necessity to acquire all three input attributes in a single pass. Our cameras have fixed lenses, which means intrinsic parameters will stay constant from capture to capture. While the cameras will likely move from capture to capture (assuming the thermal conditions of the room and camera are constant), they will likely remain static from frame to frame, with the only necessary per-frame datapoint being the point cloud. This meant that we could

Fig. 4. Example Bundle Adjustment Output from Colmap

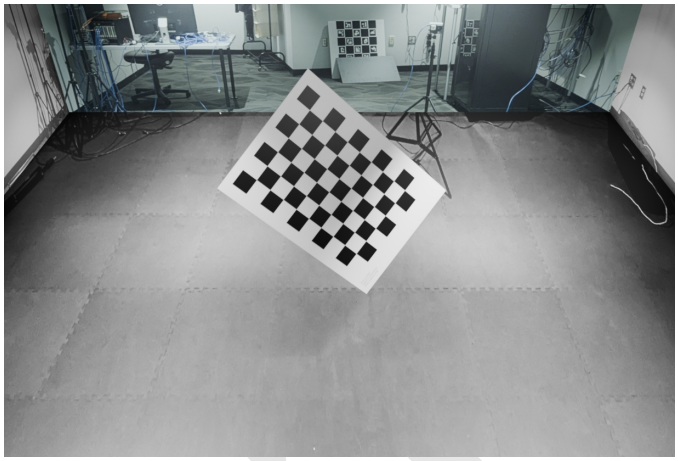

Fig. 5. Checkerboard used for calibration in Synthetic Datasets

break our software up into a linear intrinsic calibration stage, an extrinsic calibration stage, and a point cloud creation stage, with each process utilizing the data collected earlier.

B. Intrinsic Calibration

Our system relies on OpenCV checkerboard calibration to estimate the intrinsic parameters of the camera. Checkerboards work as excellent calibration tools because they provide known geometry. An edge detection algorithm can extract each of the corners on the board, and its structural uniformity makes its points easy to threshold from others within the frame. Its symmetry often makes the axis of orientation ambiguous; this can be rectified by placing a fiducial marker in each square to assign each point a unique ID.

The checkerboard calibration itself is a bundle adjustment process, where the basic camera model—fx, fy (focal length), cx, cy (camera center), and some degree of distortion coefficients—is initialized, along with the rotation and translation vectors of a checkerboard in various images taken by the same camera. These values are refined to reduce the reprojection error between the extracted points from the image and the reprojected points based on the estimated camera model and checkerboard transform.

Before the calibration step, processes like subpixel refinement can be applied to increase the precision of the 2D points extracted beyond the integer precision of the camera resolution.

C. Extrinsic Calibration

Our initial implementation relied on using the checkerboard for calibrating camera extrinsics as well. Once the cameras are placed into the volume, a new dataset can be collected of the checkerboard being waved around the room. A perspective-n-point algorithm [15], such as RANSAC, can be used to find the checkerboard transform from the camera based on the fixed intrinsics acquired during the previous calibration phase. Given that frames in the array are synchronized, the relative positions of each camera can be determined by inverting their transform from the checkerboard. These relative positions can be refined by averaging together multiple frames as the checkerboard is moved around the volume.

As development on the system progressed, manually calibrating the system before every usage became extremely inconvenient, particularly when testing new features and updates outside the calibration process. For the second stage of the project, we switched to using the checkerboard for intrinsics alone and instead acquired extrinsic data with the use of AprilTags.

AprilTags [16] are fiducial markers from which the four corners and a tag ID can be extracted in an image. Given the scale of the tag is known, the tag transformation from the camera can be estimated similarly to how the checkerboard is. A single tag alone does not necessarily provide robust or precise extrinsic data; numerous tags scattered throughout a space, however, can lower error and allow for greater flexibility in camera positioning, given that an equally precise map of those tags' locations can be provided. An effective AprilTag calibration would mean that the camera position could be estimated from the first frame of a volumetric capture, greatly speeding up the process.

We rebuilt our synthetic lab in Blender to contain AprilTags along its walls and created two datasets: a marker calibration set, where a single camera is moved around the room incrementally with about 100 separate viewpoints; and a camera calibration dataset containing only 16 views, simulating the static positions of a camera array before beginning a volumetric capture. The first dataset would be used to build the "marker map," filled with the translation and rotation values of each marker within the room, and the second would be used to validate that map by estimating the camera positions based on it, and comparing those positions with the ground truth from Blender.

Our first approach to marker map creation involved stitching together multiple frames by using the shared marker with the lowest reprojection error. The additional shared markers in the sequential frames would have their transforms averaged with a weight determined by the sum of inverse reprojection

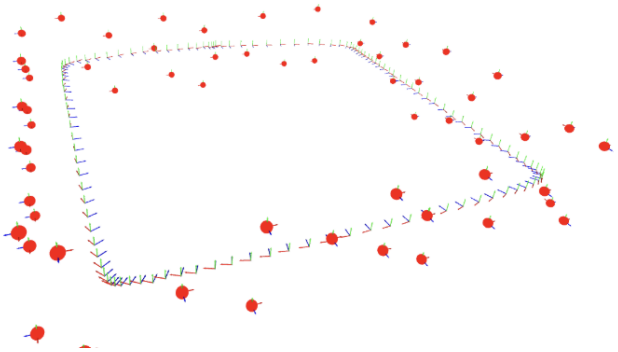

Fig. 6. Marker Map Created From Apriltag Output

error. Once the map was complete, camera transforms were estimated by means of the pose given by every marker in the captured frame. This approach neither delivered the precision nor the robustness necessary for quality real-world capture. The primary issue is that it was far too dependent on the quality of the tag itself. Where the synthetic tags used in the simulation were perfectly flat, providing excellent rotation accuracy, practical tags are prone to bending and distortion. Even the smallest imperfection in rotation caused the marker map to drift while corresponding tags were stitched together.

Our second approach forewent the rotation data and only relied on the translation data estimated from each tag. Each frame in marker creation was converted into a point cloud, and point-to-point registration was used to determine the transform from the first frame to the next. Once the point clouds were aligned, shared points could be averaged together, and new points could be appended. This process benefited from being iterable, meaning projections could continually be averaged together until the error stops decreasing. After the marker map was created, we could then estimate camera positions by performing the same point registration function between the captured point cloud and the newly created marker map. This approach greatly improved robustness; however, it failed to achieve the sub-millimeter precision necessary for high-quality volumetric capture.

Our final implementation took primary inspiration from the very system we attempted to replace: COLMAP. We still applied the registration techniques from before, but this time the marker and camera transforms estimated from that process were used as initialization values. Then, we built a least-squares regression function to minimize the residual error between the extracted 2D marker centers and the corresponding reprojected marker points. With this implementation, camera positions are only estimated using a PnP RANSAC [17] method based on the marker centers and output marker map. This optimization process helped us bypass bottlenecks created by inaccuracies in marker depth estimation, and guarantee the most precise result possible, delivering the necessary precision with relatively quick computation time.

V. Point Cloud Creation

Once camera extrinsics and intrinsics are estimated, the last step is point cloud creation. This point cloud will only serve as an initialization point for our light field optimizer, so it is not necessary to have absolute precision; however, the higher the quality of the point cloud, the quicker our optimizer will be able to converge on a coherent reconstruction. Since we wanted our system to be able to run with a sparse set of viewpoints, having each camera be able to create its own reconstruction became a priority for our system. This meant that, corresponding with the RGB capture from our cameras, there would need to be a depth output.

A. Time-of-Flight Depth

Our initial design utilized a time-of-flight camera. This imaging system emits narrow-frequency light into the room and is filtered to be sensitive only to light around its emitter's wavelength. Instead of measuring the intensity of light that returns to the sensor, it measures the amount of time between the emitter strobing and the signal being detected by each pixel. This time is multiplied by 1/2 the speed of light to determine the distance the ray traveled before reflecting back to the sensor, resulting in a depth map. This depth map can be rectified onto the RGB image and projected into the scene using the intrinsic and extrinsic parameters of the camera, creating a colored point cloud.

While this hardware configuration worked well during early tests with a minimal number of cameras in the volume, issues began to arise as we increased the number. Since each camera is made with the same wavelength emitter, each was sensitive to light being cast by its neighbors, which caused the cameras to interfere with each other. Very quickly, the output point clouds became unusable. One possible solution to this problem would have been desynchronizing the time-of-flight cameras from one another. As long as no two cameras are strobing at the same time, they would be able to operate in the space with no interference. Unfortunately, our camera API did not provide the low-level access needed to change the internal timing of the cameras.

With the interference issue aside, there were many other issues with the cameras that made them unsuitable for our application. While they were capable of detecting objects at a distance of up to 4 meters, the power of the emitter meant the data became increasingly noisy as our volume expanded. Additionally, rectifying the depth camera with RGB was an unrefined manual process, which interrupted an otherwise automatic calibration workflow.

B. Stereo Depth

For our stage 2 implementation, we switched to stereo-based depth acquisition. Stereo systems work by measuring the disparity between matching blocks of pixels in image pairs. If the cameras are properly rectified with one another, disparity will decrease as the object is moved further away, and given that the focal length and interpupillary distance are provided, that disparity measurement can be directly converted

Fig. 7. Depth map output from time of flight camera

Fig. 8. Depth map output from stereo camera

to depth (z). Stereo has certain drawbacks compared to a time-of-flight method. It is far more scene-dependent; if an image is noisy or lacks high-frequency information, the disparity solver will struggle to find matching blocks. Additionally, it can be more easily fooled by things like repeating patterns, leading to inaccurate estimates.

Stereo systems are far better at longer distances, with IPD being easily tunable based on the target range. Additionally, parameters necessary for rectification, like the translation and rotation offsets between each camera, can be acquired during the same checkerboard process as the intrinsic values, allowing it to easily integrate into the prebuilt workflow. It also allowed us to experiment with "Foundation Stereo" [18], which is a machine learning-based implementation built on a model trained using synthetic computer-rendered depth maps to get higher quality outputs.

C. Voxel Carving

The final point cloud creation system implementation we added to our system is called voxel carving. With a voxel carving method, a discrete grid of uniform voxels is initialized and is "chipped away" using binary masks of the foreground object, projected from the extrinsic parameters [19]. Since the primary objective of our system is to capture human performance, the human can be masked using a segmentation model such as Segment Anything by Meta [20], or Nvidia RVM (Robust Video Matting) [21].

The voxel carving method has the primary benefit of not requiring specialized hardware; a standard monocular camera is all that is necessary. Since less data must be recorded, this lightens the load of capture and speeds up processing later on, making it the most efficient solution. The compromise is that this system is only able to estimate geometry for the masked object, meaning if the entire space must be represented volumetrically, this approach becomes insufficient. Additionally, the quality of the point cloud is far more dependent on the number and placement of cameras. Additionally, since the method can only measure voxel presence by the silhouette of the object, concave surfaces cannot be accurately represented. While the general shape of human subjects is not particularly affected by this, it makes voxel carvings' output quality heavily dependent on the nature of the object it is reconstructing.

VI. Hardware Design

After creating and proving our original concept with the synthetic datasets, we began outlining hardware requirements for a physical camera to satisfy our needs. While searching for hardware solutions, we found that none of the off-the-shelf or turnkey systems satisfied the requirements; however, we determined that we would be able to construct a camera system that would not only meet those needs but also be modular and upgradable as the process went forward. Our camera design, which we call the "Chumper," is based on a Raspberry Pi 5. The enclosure is 3D-printed and features a swappable lid that serves as a mounting point for any sensor package we may need. To maintain cable management, we utilize Power over Ethernet, which also enables remote control and file transfer.

Our original design utilized an Arducam 2-megapixel IMX415 RGB sensor, which paired nicely with the Arducam time-of-flight camera due to them both using the same Pivariety driver. When we switched to the stereo configuration, we upgraded to an 8MP IMX477. While the limitations of the CSI connector made taking advantage of higher resolution impossible for motion capture, we are able to use it during the calibration phase, which helps increase the precision of our corner detection. An alternative to this camera, which we did not get the opportunity to investigate, was a global shutter camera.

A. Global Shutter vs. Rolling Shutter

Since rolling shutter cameras scan down the sensor as an exposure is taken, a skewing distortion can result in the final image if an object in the scene, or the camera itself, is moving. This means that during our calibration process, we must ensure that the camera and checkerboard are totally still for every image used for refinement. A global shutter camera exposes each pixel at the same time, meaning the only artifact from motion in the scene is blur from the shutter duration.

Overall, the benefits of a global shutter are mostly convenience. Currently, our practice for calibration is to place the

Fig. 9. Version 1 Of the Volume Capture Camera

Fig. 10. Version 2 of the Volume Capture Camera

checkerboard on a wall, the camera on a tripod, and move the tripod, letting it rest at the various views needed for calibration. A global shutter camera could make this process smoother by simply holding the checkerboard and waving it around the scene. The compromise of this convenience would be the overall image quality. An equivalently priced global shutter camera, such as the IMX296, sacrifices resolution and dynamic range compared to the IMX477.

To synchronize our cameras with one another, we rely on Precision Time Protocol (PTP) [22]. Our host PC serves as the "master" clock, which the system clocks on each Pi become "slaves" to. In Picamera2, we utilize the built-in synchronization protocol to align the shutter with the internal clock and save timestamps for each exposure, which can be converted to frame number based on the selected framerate. An LED on each unit signifies that the cameras are in the process of synchronizing with a blink, or ready to capture synchronized frames with a solid color. Images are written as uncompressed 3-channel, 8-bit TIFF files onto the internal SSD of the Raspberry Pi. This format minimizes CPU load while recording, at the cost of file efficiency. After the recording is complete, each Pi serves as a network drive, meaning that its files can be accessed by the host machine for processing almost immediately.

VII. Data Processing Pipeline

Before 3D light-field optimization can begin, we must prepare our dataset in 2D. The first step in this pipeline is extracting the extrinsic parameters for each camera, using the same calibration function described earlier. During capture, after each camera is initialized and before recording begins, a "calibration" frame is taken, which, unlike the pixel-binned capture frames, uses the full resolution of the sensor. Camera intrinsics are loaded from a parameters file stored on the Pi itself and used in combination with the calibration frames to calculate each extrinsic. The extrinsics are written to a text file in COLMAP format and placed in a unique folder for each frame captured during the take. Our calibration script prepares each image sequentially, camera to camera, so each network drive can be accessed one at a time rather than cyclically per frame.

After reading an image, the first step in the pipeline is stereo rectification and undistortion. Light-field optimizers tend to rely on pinhole camera models for rendering to help maintain speed and fast iterations. This means that providing the calibrated spherical model alone is insufficient for rendering. Additionally, to measure disparity across the x-axis for depth calculation, the two cameras must be perfectly aligned with convergence at infinity. Since our hardware configuration has the cameras fixed to the board, this rectification must be handled digitally. As an output of the intrinsic calibration process, a projection matrix is created, which can be applied to each image. It is important to note that this matrix will change the relative rotation vector of the camera, so the previous step, extrinsic calibration, should be performed with rectified points as well. Once the images are rectified, we can write an additional text file in COLMAP format to each frame folder using the new camera matrix as a result of the reprojection.

If the user prefers to utilize the monocular workflow with voxel carving for point cloud creation, the rectification and depth estimation phases may be skipped. In both processes, the process image must be unidistorted and reprojected to fit the new pinhole matrix before optimization.

The next step in the stereo pipeline is to estimate depth frames; our script supports both traditional block-matching and disparity filtering algorithms, as well as machine learning-based foundation methods. The foundation methods take longer per frame but provide higher-quality outputs. We save our depth maps as 16-bit TIFF files, with integer values scaled to a "depth range" as specified by the user. If additional precision is necessary, we recommend 32-bit EXRs. After depth estimation is complete, we discard the image from the right camera; although, as we will discuss later, there may be

Fig. 11. Volumetric Capture Output

benefits to using both cameras in light-field optimization.

Even though our cameras are the same model, we found there to be variation in tone between images. Using the rectified left frame, we apply a white balance and color correction matrix to the image based on a photographed 24-patch color checker. After color calibration is applied, the final step in the 2D phase is optional segmentation. Since most applications are only interested in capturing the subject within the volume, we can segment out the background to create cleaner and more efficient volumetric rendering. Our process relies on NVIDIA Robust Video Matting [21], which is a machine learning model that generates an opacity map to extract human subjects from the frame. We then write the image as a PNG file to its corresponding frame folder, with this opacity map as its alpha channel.

The final phase in the processing segment is point cloud creation, which utilizes the processed masked images, new camera intrinsics, and optional depth maps for the stereo pipeline. In the first step of the stereo pipeline, the alpha channel is converted to a binary map and multiplied onto the depth image, then each corresponding RGB value is assigned to each pixel in the depth map and projected into 3D space based on the new camera matrix. Depth values equal to 0 as a result of the alpha channel are not projected into the scene. This process is applied to every camera in each frame. Finally, voxel downsampling is applied to the final point cloud to filter redundant points and increase efficiency.

In the monocular pipeline, the voxel carving approach described earlier is applied to create a sculpted voxel grid of the subject. The voxel grid can be converted into a point cloud by placing a vertex at the center of each surface voxel. Additionally, it can be colored by back-projecting the 3D position into 2D image space for each image, and associating that voxel with the RGB color at that pixel. This point cloud is finally saved as a text file in COLMAP format, making the dataset complete and ready for optimization using the process of the user's choice.

VIII. Future Work and Conclusion

Any project of this nature will never be fully complete. As future approaches to light field optimization are developed, the system will need to change to adapt to the landscape. In the meantime, there are many avenues to refine and streamline the system, from workflow optimization to processing datasets more quickly. Our encoding system could be more sophisticated. Simple modifications, like chroma subsampling or interframe encoding, could greatly improve efficiency while maintaining image quality.

On the processing side, our current matting technique is limited to human subjects; adding a segmentation model such as Segment Anything by Meta [20] could allow for a wider array of props and accessories to be captured. The biggest inefficiency in the processing pipeline is the computational power of the Raspberry Pi after the capture phase. Currently, increasing the number of cameras linearly increases the processing time per frame in the data preparation stage. Utilizing a cluster compute system, where each Pi acts as a node within the system, would expand the computational abilities of the system as computational demand increases.

Where the most work remains to be done is in fulfilling the intended purpose: experimenting with various reconstruction techniques and adapting them for production use. Whether it be Gaussian splatting, triangle splatting, or neural radiance fields, adapting these 3D representations takes three major steps: motion encoding for temporal consistency, radiance deconstruction for relighting purposes, and streamlined compatibility with industry-standard software like Maya, Houdini, and Blender. The possibilities for volumetric capture are exciting, but if it is to be widely adopted, it must be proven as an accessible and intuitive tool for digital storytellers of all varieties.

Our tool does not accomplish this yet, but as we continue to test, iterate, and improve it, we hope it will transform from a testbed offering a glimpse into the future to a resource that helps reinvent the way stories are told.

Acknowledgment

This work was supported in part by the National Science Foundation (NSF) under Grant Award Number 2238180 to Flip Phillips.

References

[1] C. H. McCollough and P. S. Rajiah, "Milestones in ct: Past, present, and future," *Radiology*, vol. 309, no. 1, p. e230803, 2023, pMID: 37847140. [Online]. Available: https://doi.org/10.1148/radiol.230803

[2] R. Ng, M. Levoy, M. Brédif, G. Duval, M. Horowitz, and P. Hanrahan, "Light Field Photography with a Hand-held Plenoptic Camera," Stanford University, Technical Report CTSR 2005-02, 2005.

[3] R. A. Drebin, L. Carpenter, and P. Hanrahan, "Volume rendering," *ACM SIGGRAPH Computer Graphics*, vol. 22, no. 4, pp. 65–74, 1988.

[4] M. Levoy and P. Hanrahan, "Light field rendering," in *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. ACM, Aug. 1996, pp. 31–42.

[5] J. J. Gibson, *The Ecological Approach to Visual Perception*. Psychology Press, 1979.

[6] ——, *The Senses Considered as Perceptual Systems*. Houghton-Mifflin, Jan. 1966.

[7] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," *arXiv preprint arXiv:2003.08934*, 2020.
[8] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," *ACM SIGGRAPH Computer Graphics*, vol. 21, no. 4, pp. 163–169, Aug. 1987.
[9] R. L. Cook, L. Carpenter, and E. Catmull, "The Reyes image rendering architecture," *SIGGRAPH Comput. Graph.*, vol. 21, no. 4, pp. 95–102, Aug. 1987.
[10] P. Christensen, J. Fong, J. Shade, W. Wooten, B. Schubert, A. Kensler, S. Friedman, C. Kilpatrick, C. Ramshaw, M. Bannister, B. Rayner, J. Brouillat, and M. Liani, "Renderman: An advanced path-tracing architecture for movie rendering," *ACM Trans. Graph.*, vol. 37, no. 3, aug 2018.
[11] M. Wrenninge, *Production Volume Rendering: Design and Implementation*. A K Peters/CRC Press, 2012.
[12] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3d gaussian splatting for real-time radiance field rendering," *arXiv preprint arXiv:2308.04079*, 2023.
[13] J. H. et al., "Triangle splatting for real-time radiance field rendering," *arXiv preprint arXiv:2505.19175*, 2025.
[14] Y. Chen, Y. Chen, and G. Wang, "Bundle adjustment revisited," *arXiv preprint arXiv:1912.03858*, 2019.
[15] T. Collins and A. Bartoli, "Infinitesimal plane-based pose estimation," *International Journal of Computer Vision*, vol. 109, no. 3, pp. 252–286, 2014.
[16] E. Olson, "April laboratory april papers: Autonomy * perception * robotics * interfaces * learning," https://april.eecs.umich.edu/papers/details.php?name=olson2011tags, 2011.
[17] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.
[18] B. Wen, M. Trepte, J. Aribido, J. Kautz, O. Gallo, and S. Birchfield, "Foundationstereo: Zero-shot stereo matching," *arXiv preprint arXiv:2501.09898*, 2025.
[19] K. N. Kutulakos and S. M. Seitz, "A theory of shape by space carving," *International Journal of Computer Vision*, vol. 38, no. 3, pp. 199–218, 2000.
[20] N. Ravi, V. Gabeur, Y.-T. Hu, R. Hu, C. Ryali, T. Ma, H. Khedr, R. Rädle, C. Rolland, L. Gustafson, E. Mintun, J. Pan, K. V. Alwala, N. Carion, C.-Y. Wu, R. Girshick, P. Dollár, and C. Feichtenhofer, "Sam 2: Segment anything in images and videos," *arXiv preprint arXiv:2408.00714*, 2024. [Online]. Available: https://arxiv.org/abs/2408.00714
[21] S. Lin, L. Yang, I. Saleemi, and S. Sengupta, "Robust high-resolution video matting with temporal guidance," *arXiv preprint arXiv:2108.11515*, 2021.
[22] *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, IEEE Std. IEEE Std 1588-2019, 2020.