3D Modeling of Cities for Virtual Environments

Calvin Davis[1], Jaired Collins[1], Emily Lattanzio[3], Joshua Fraser[1], Shizeng Yao[1], Haoxiang Zhang[1],
Bimal Balakrishnan[2], Ye Duan[1], Prasad Calyam[1], Kannappan Palaniappan[1]
[1]Dept. of Electrical Engineering and Computer Science, [2]Architectural Studies, University of Missouri, Columbia, MO
[3]High Point University, High Point, NC

***Abstract*—Modeling and simulation of large urban regions is beneficial for a range of applications including intelligent transportation, smart cities, infrastructure planning, and training artificial intelligence for autonomous navigation systems including ground vehicles and aerial drones. Immersive environments including virtual reality (VR), augmented reality (AR), mixed reality (MR or XR) can be used to explore city scale regions for planning, design, training and operations. Virtual environments are in the midst of rapid change as innovations in display technologies, graphics processors and game engine software present new opportunities for incorporating modeling and simulation into engineering workflows. Game engine software like Unity with photorealistic rendering and realistic physics have plug-in support for a variety of virtual environments and typically model the scene as meshes. In this paper, we develop an end-to-end workflow for creating urban scale real world accurate synthetic environments that can be visualized in virtual environments including the Microsoft HoloLens head mounted display or the CAVE VR for multi-user interaction. Four meshing algorithms are evaluated for representation accuracy and city-scale meshes imported into Unity for assessing the quality of the immersive experience.**

***Index Terms*—3D Reconstruction, Mixed Reality, Synthetic Environment, Pipeline, Meshes, Texture Mapping, Point Cloud**

I. Introduction

Mixed reality (MR) has significantly influenced technology by immersing users in synthetic virtual environments, providing capabilities such as communication [1] and autonomous vehicle testing [2]. Our work focuses on utilizing the immense capabilities of MR with 3D modeling to display cities as meshes in MR devices. This paper explores an effective pipeline for photo-realistic mesh reconstruction from a point cloud, its challenges, and its use cases on a city scale.

Synthetic virtual environments can improve communication, especially between professionals and team members, by showing 3D information that would otherwise take more effort and time to produce [1]. On a city scale, city planners and civil engineers can produce solutions to their problems and communicate it to others through a city's virtually reconstructed environment.

Autonomous vehicles, whether on land, water, or air, have had significant issues with testing. High costs in hardware and man-hours means the testing environment must be safe. To solve this issue, synthetic environments have been created to allow testing that is mostly consistent with the real world [2], [3]. Using virtual city-scale reconstructed environments with these testing platforms can enable rapid and safe development of autonomous drone algorithms.

The general workflow in Figure 1 was developed to view 3D point clouds in the Microsoft HoloLens 1, an augmented reality device and the Cave Automatic Virtual Environment (CAVE). The steps include converting dense point clouds generated with the multi-view stereo algorithm VB3D [4] into polygon meshes, applying photo-realistic texture mapping, and transferring the meshes into Unity to create synthetic environments for visualization and drone flight planning experiments. We experimentally compare four different free-to-use meshing platforms to determine efficacy for 3D modeling city-scale urban environments using a free and easy to recreate pipeline for MR and flight planning.

II. Mesh Generation From Point Clouds

A. Four Common Meshing Algorithms

The ball-pivoting algorithm (BPA) uses a ball of fixed radius to traverse a point cloud creating triangles when three distinct points are in contact with the ball, as the ball pivots around edges created during the previous step(s). Holes resulting from irregularly sampled surfaces are filled by running BPA multiple times with increasing radii [5].

Poisson surface reconstruction formulates mesh reconstruction as a Poisson problem resulting in a global solution producing smooth surfaces that are robust to noise [6]. The surface is extracted from oriented point samples by first calculating an inside-outside indicator function discretized to an octree then extracted using the marching cubes algorithm. Once the marching cube traverses to the desired octree depth, the triangulated 3d mesh is created by interpolating the points between the cube vertices. The Screened Poisson surface reconstruction method adds point constraints to reduce the original method's tendency to over smooth [7].

Point Cloud Library (PCL) implements mesh reconstruction with a greedy triangulation method [8] which incrementally adds edges that never cross another, so edges are never deleted. Each point p is assigned k neighbors in a sphere, determined by local density. A plane computed by a weighted least squares from point p's neighborhood is used to estimate the true surface normal. Afterwards, points are pruned based on visibility, connected to p, and consecutively connected to each other to form triangles. A maximum and minimum angle criteria are used to reduce the smoothing of corners. This method focuses on fast triangulation so that it may be used in real-time applications where speed and a robustness to noise are important.

B. Similar Pipelines

Maiti and Chakravarty [9] study surface reconstruction using point clouds resulting from feature-based photogrammetry. They evaluate both Poisson surface reconstruction [6] and the Ball-Pivoting Algorithm (BPA) [5] with a discussion on tuning algorithm parameters and demonstrating the impact these parameters have on the resulting water-tight mesh.

Bosch et al. created an open source ground truth and metric evaluation pipeline for urban areas sourced from commercial satellite imagery along with benchmark datasets [10]. They establish evaluation metrics for photogrammetric point cloud accuracy to LiDAR, horizontal accuracy to public mapping vector products, semantic labeling, volumetric accuracy, curvature and roughness, and triangle mesh model simplicity.

Poulis and You directly use LiDAR for city reconstruction [11]. To simplify the computation, a nadir view of a LiDAR point cloud is converted into a 2D XYZ map, where $X = \mathrm{R}$, $Y = \mathrm{G}$, and $Z = \mathrm{B}$. Gaussian distributions are then used to segment similar regions that correspond to roofs. Then, boundaries are extracted from the segmentation results. Planes are fitted to the boundaries and then extruded downward, producing a watertight mesh. For texture mapping, available images are matched to scenes of the extruded 3D mesh and bundle adjusted. Texture coordinates are found by projecting the mesh's triangles into all images and using the image with the highest projected area.

Kuschk shows that city mesh reconstructions can be produced by connecting neighboring points of a created digital surface model [12]. First, bundle adjusted camera poses and corresponding images are used for dense stereo reconstruction to obtain height information. Then, in the digital surface model, pixels $(x, y), (x + 1, y), (x + 1, y + 1), (x, y + 1)$ are naïvely connected into two triangles. Since the resulting mesh has too many polygons, two simplifications are performed: plane fitting to remove redundant vertices and removal of nearly collinear triangles. After simplification, the texturing method projects all triangles to all the images, choosing the image with the highest projected area.

III. Methods

This section details our pipeline for modeling and visualization of urban environments from 3D point clouds as shown in Figure 1. First a triangle mesh is generated from a city-scale point cloud using freely available software packages, then our custom high-resolution texturing algorithm is applied, and finally we import our results into Unity to create synthetic environments for HoloLens and CAVE.

A. Creating a Mesh

We begin with dense 3D point clouds of two urban areas Columbia, MO and Albuquerque, NM created using the VB3D aerial multiview stereo algorithm [4]. The aerial imagery are from a high resolution metric camera [13] with the onboard exterior camera orientations refined using the fast BA4S bundle adjustment algorithm [14] Additionally, LiDAR is available for Columbia, MO as an additional source of point data for algorithm comparison and evaluation.

We use the surface reconstruction algorithms from the following freely available software packages: the Ball-Pivoting Algorithm (BPA) [5] and Screened Poisson [7] available in MeshLab [15], Poisson Surface Reconstruction [6] in CloudCompare [16], Greedy Surface Triangulation [17] in Point Cloud Library (PCL) [17], and Poisson Reconstruction with Delaunay Triangulation [18] in the Computational Geometry Algorithms Library (CGAL) [19].

Each surface reconstruction algorithm provides parameters that tune its robustness to noise, holes, and non-uniform sampling. Figures **??** and **??** show the effects of these parameters on the resulting surface. Tuning these parameters not only affects the quality of the resulting surface, but also significantly affects the compute time and memory usage.

The Ball-Pivoting Algorithm (BPA) [5] is tuned by the ball radius, clustering radius, and angle threshold parameters. The ball radius parameter controls the size of the ball that traverses the point cloud, where a small radius will be sensitive to noise due to including most points in the mesh, and a large radius pivoting over noisy points will result in a smoother surface at the cost of finer details. Clustering radius filters out points that lie too close together in a neighbor as a percentage of the ball radius. Because clustering is defined as a percentage of the ball radius, it is important to note that these are not independent parameters. Angle threshold stops the ball traversal at an edge that would require a pivot angle greater than the threshold.

Greedy Surface Triangulation [8] uses maximum number of neighbors, farthest neighbor, maximum edge length, minimum and maximum angles, and maximum surface angle. Defining a maximum number of neighbors and the farthest possible neighbor filters local influence since a plane is estimated from the neighborhood. The maximum edge length, minimum angle, and maximum angle limit resulting triangles, but the minimum angle might not be honored. The maximum surface angle determines the maximum angle between two triangle normals, and an optional consistency check is available to ensure adjacent normals lie on the same side of the surface.

The Poisson algorithm [6] controls the voxel resolution by setting the maximum depth of the octree for surface reconstruction. Increasing this depth results in higher-resolution triangle meshes. Samples per node is the minimum number of points that should lie in a node during octree construction as it adapts to sampling density. Five or fewer samples per node may be suitable in the absence of noise, while twenty or more may be required to smooth noisy data. The Screened Poisson algorithm [7] reduces the tendency to over-smooth with an additional interpolation weight parameter. A lower interpolation weight puts more importance on fitting the gradients, while a higher value constrains the mesh more to the points.

B. Selecting the Best Mesh

Meshes used for drone flight planning should not have any holes, otherwise a flight path could be created to erroneously fly into a surface that is thought to be open. The best meshing

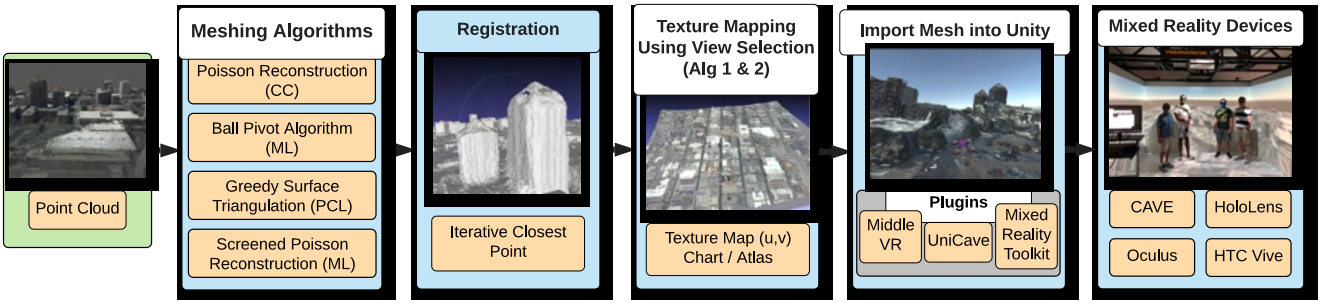
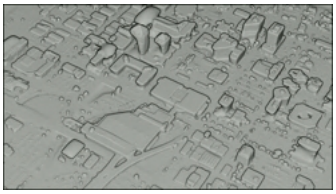
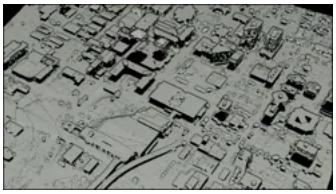


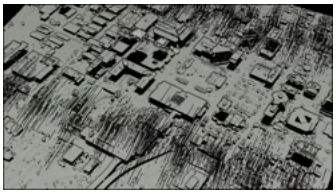
Fig. 1: Our workflow for generating texture mapped meshes from 3D point clouds to be viewed through a mixed reality device.



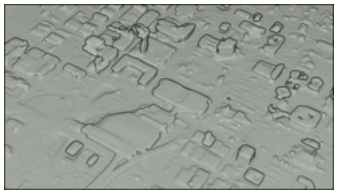
(a) Poisson reconstruction algorithm in CloudCompare. $\mu = 0.22$, $\sigma = 0.53$, #Faces = 2.9M



(b) Ball-pivoting algorithm in MeshLab. $\mu = 0.04$, $\sigma = 0.36$, #Faces = 1.7M



(c) Greedy surface triangulation algorithm in PCL. $\mu = 0.05$, $\sigma = 0.38$, #Faces = 2.0M



(d) Screened Poisson algorithm in MeshLab. $\mu = 1.15$, $\sigma = 2.6$, #Faces = 2.2M

Fig. 2: Comparison of LiDAR meshes to LiDAR_{pc} point cloud. Albuquerque, NM meshes were reconstructed using four different meshing algorithms. Mean and standard deviations are statistical measures of the distance between each point and the respective mesh in meters.

TABLE I: Quantitative evaluation of VB3D meshes to VB3D_{pc} or LiDAR_{pc} point clouds. The four meshing algorithms are evaluated across the full city scene by comparing the triangular meshes to the reconstructed VB3D point cloud of Albuquerque, New Mexico and to the LiDAR point cloud for the same region. Units are in meters.

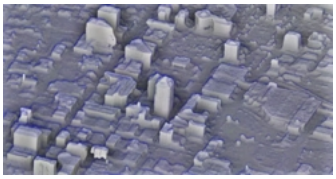
	Poisson			Ball-Pivoting			Greedy			Screened Poisson		
Alg Parameters	Octree=11, Samples=10, 1.5, 1.5			Ball Radius=2, Clustering=20			Mu=2.5, Max Neighbors=100			Octree=11, Samples=1		
Mesh → PointCloud	μ	σ	# Faces	μ	σ	# Faces	μ	σ	# Faces	μ	σ	# Faces
VB3D → VB3D_{pc}	0.85	0.89	1.19M	0.45	0.70	2.38M	**0.01**	**0.12**	8.94M	1.73	1.98	0.19M
VB3D → LiDAR_{pc}	1.19	1.49	1.19M	0.81	1.01	2.38M	0.50	0.93	8.94M	1.70	1.93	0.19M
LiDAR → LiDAR_{pc}	0.22	0.53	2.92M	0.04	0.36	2.23M	0.05	0.38	2.06M	1.15	2.60	1.74M

Fig. 3: Side-view of the point cloud rendered model of Albuquerque, New Mexico; created with multi-view stereo approach with 215 aerial images.

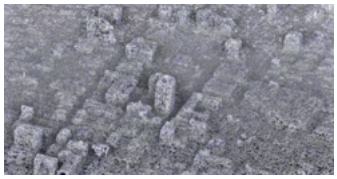
Fig. 4: Albuquerque, New Mexico. Matches Figure 5.

algorithm would also need to be robust to noise in the underlying point cloud. Figures 5 and 6 show a few results of our meshing. To find the best mesh with BPA and greedy triangulation, a grid search was performed with their many parameters. Figure 7 shows a few BPA results. Care is taken to produce the best mesh that fits the characteristics of the data.

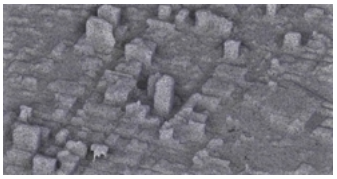
BPA and greedy surface triangulation produce meshes that are too noisy. No smoothing is used due to the nature of the algorithms. Additionally, holes are present in the generated



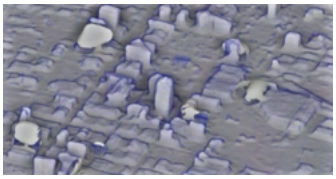
(a) Poisson (in CloudCompare). $\mu = 0.85$, $\sigma = 0.89$



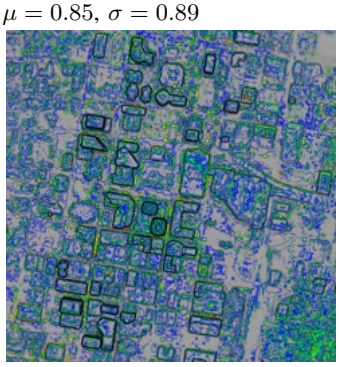
(b) Ball-pivoting algorithm (in MeshLab). $\mu = 0.45$, $\sigma = 0.70$



(c) Greedy surface triangulation (in PCL). $\mu = 0.01$, $\sigma = 0.12$

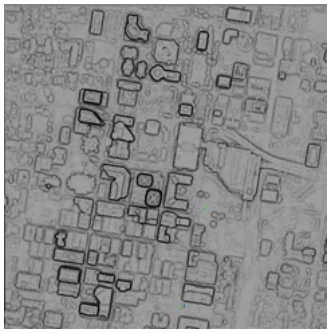


(d) Screened Poisson (in MeshLab). $\mu = 1.73$, $\sigma = 1.98$

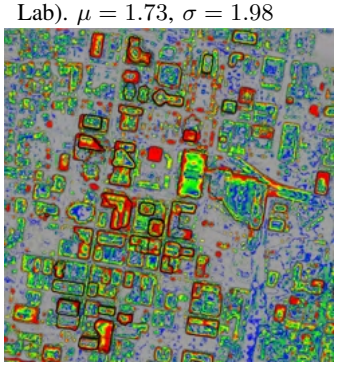


(e) Poisson reconstruction

(f) Ball-pivoting algorithm



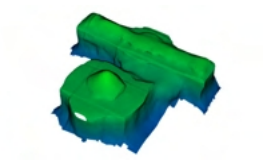
(g) Greedy surface triangulation



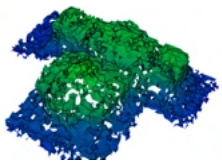
(h) Screened Poisson

Fig. 5: Visual comparison of VB3D meshes with $VB3D_{pc}$ point cloud for ABQ dataset (see Figure 4 for an aerial image from the same viewpoint). Top row shows rendered meshes with mean accuracy and standard deviation of the distance errors between each point and the mesh surface given in meters. Second row shows meshing results for the full scene with errors between 0 to 1 meter shown in gray, errors more than 1 meter colored using a jetmap: blue colors are lower errors, while red colors are higher errors.

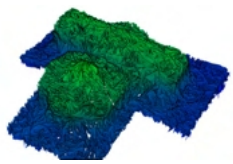
(a) Building from original image



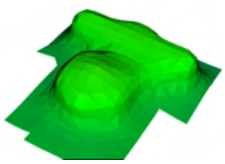
(b) Poisson reconstruction. $\mu = 1.74$, $\sigma = 1.79$, # Faces = 15.6K



(c) Ball-pivoting algorithm. $\mu = 0.76$, $\sigma = 0.90$, # Faces = 11.5K



(d) Greedy. $\mu = 0.34$, $\sigma = 0.57$, # Faces = 91.0K



(e) Screened Poisson. $\mu = 2.45$, $\sigma = 2.29$, # Faces = 1.1K

Fig. 6: Cropped building meshes of the Boone County courthouse in Columbia, MO. The LiDAR ground truth point was first infilled using a point density of 4 pt/m^2, then accurately aligned with the VB3D point cloud using the iterative closest point algorithm with 50,000 3D points. The final RMSE between the LiDAR ground truth and the VB3D point cloud was 1.03. Mean and standard deviation of the distance errors (in meters) between each LiDAR ground truth point and the mesh surface.

meshes. Post-processing could be done to remove holes in the mesh, but not all could be easily filled. Both Poisson and screened Poisson methods generate meshes that are watertight, robust to noise, and smooth. Screened Poisson includes point constraints to prevent losing detail and oversmoothing, which we also evaluated.

CloudCompare's default mesh distance function is used to compute the distances between the point cloud and mesh. Figure 6 visualizes a cropped building using the four meshing algorithms. Table I and Figure 5 shows the mesh algorithms compared to an aerial image.

C. Texture Mapping Using View Selection

The resulting polygon meshes contain only per-vertex color retained from the source point clouds. Achieving photo-realism requires applying high-resolution textures to our model. Rather than create a single texture atlas and UV parameterization, we divide the mesh into triangle sets texture mapped to one of a small set of the original high-resolution aerial images evenly spaced about a circular orbit of our scene. We use the associated bundle adjusted camera poses for each image and face normals to choose the view to use as a texture for each triangle. One triangle set is created for each texture image and triangles are assigned to a set based on the smallest angle between the camera view vector and the face normal as shown in Algorithm 1.

After assigning each triangle to an associated image set, the face must be UV parameterized to the image for texture display. Generating UVs for each triangle only requires projecting vertex coordinates to the image plane, normalizing image plane coordinates, and vertically flipping to match

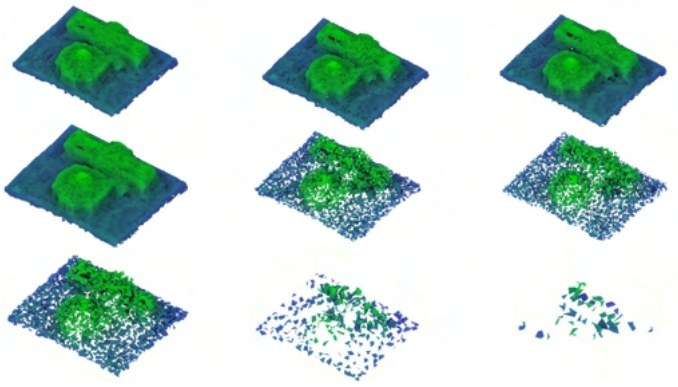


Fig. 7: Grid of ball-pivoting algorithm results. Angle threshold set to 90. Clustering radius increases downward. Ball radius increases to the right.

Algorithm 1 Match Faces to Views for Texture Mapping

Input
  M mesh with faces in winding order
  n number of image views to texture from
  Z_n array of camera look-at directions, size n
Output
  R_n list of meshes with removed faces
procedure REMOVEFACES(M, n, Z_n)
  initialize $R_n = M, \forall n$
  for all face $f \in M$ **do**
    $v_0, v_1, v_2 \leftarrow$ vertex 0, 1, 2 $\in f$
    $u \leftarrow v_1 - v_0$
    $v \leftarrow v_2 - v_1$
    $p \leftarrow u \times v / |u \times v|$
    $i \leftarrow \text{argmin}\{p \cdot z, z \in Z_n\}$
    delete f from R_n, $n \neq i$
  end for
end procedure

texture origin conventions as shown in Algorithm 2. The triangles with UV texture coordinates are written to a mesh file associated with the image used for texturing resulting in a separate mesh for each of the images used in Algorithms 1 and 2. We use MeshLab to merge the separate point clouds into a single Stanford polygon format (.ply) file with texture references. Figure 8 shows the result of texture mapping using eight images. Texture mapping could be improved with depth testing to remedy incorrectly textured triangles resulting from occlusions using a z-buffer algorithm [20].

D. Unity Game Engine and Mixed Reality Plugins

The next step in the pipeline is loading the meshes into the Unity game engine. Unity is chosen since it supports twenty eight platforms, including iOS, Oculus, and Windows Mixed Reality, Unity can port a 3D project to a wide range of different MR devices. In addition, the Unity Asset store has over thirty-one thousand 3D assets, which would be useful in applying this pipeline to different scenarios that require additional features.

Algorithm 2 Generate Texture Map Coordinates for Triangles

Input
  M mesh with faces removed
  P camera matrix associated with desired image
  w image width
  h image height
Output
  T textured mesh coordinates
for all face $f \in M$ **do**
  for all vertex $v \in f$ **do**
    convert v to homogeneous
    $(u, v) \leftarrow Pv$, project vertex to image
    $u \leftarrow u/w$, normalize
    $v \leftarrow v/(1 - (v/h))$, normalize and flip
  end for
end for

Fig. 8: Albuquerque mesh with texture from Algorithm 2 using eight views.

It is noteworthy to mention, however, that Unity is not an open source platform. This is not a limiting factor in this pipeline because it is not used for profit.

In order to load this project into Unity, each mesh and image needs to be imported into the game engine as an asset. After converting the images into Unity's equivalent of a texture, a material, they are each applied to the corresponding mesh. All meshes are overlapped by assigning them the same position in (x, y, z) coordinates. Figure 9 shows a generated mesh using our pipeline inside of Unity.

There are many Unity plugins for VR environments. In this study, we build projects for two platforms, Hololens 1 and CAVE. To use the Hololens 1, Microsoft's Mixed Reality toolkit is downloaded and '3D' is selected as the Unity project. This plugin can change the Unity settings to be ideal to use with HoloLens 1. Although the plugin sets up the Unity environment, there are many steps to deploy the application to HoloLens. In the build settings, the Universal Windows Platform is set to a D3D project with x86 architecture and default compression. The platform is then to Universal Windows. To interact with the mesh hologram we add interaction profiles to the project, which include TaptoPlace and BoundsControl, which allows clicking to move and scaling of the hologram,

respectively. A step-by-step guide on the rest of the process is provided by Microsoft [21].

MiddleVR 2.0 and UniCAVE 2.0 are the plugins ran for the CAVE experiments. These plugins make it convenient for users to set up the VR environment, connect VR headset and wands, and customize the scripts to control the wand event.

MiddleVR is a Windows-only plugin that supports an Unity project running under multi-display, stereoscopy, and VR system. It has already set up the connection between VR equipment and VR system such as CAVE environment and joystick. MiddleVR supports multi-PC and multi-GPUs with multiple projectors, so it ensures that many types of projects can be run with this plugin. MiddleVR has many built-in scripts that save a user's time to implement, such as 'object grabbable'. Users just add the components to the game object and customize parameters. Although MiddleVR has many advantages for interfacing Unity with the CAVE VR environment, its proprietary and expensive.

UniCAVE is an open source plugin for Unity made by University of Wisconsin-Madison. UniCAVE does not have many built-in functions, so users have to write their own code to implement. Compared to MiddleVR, UniCAVE does not automatically set up the connection between the VR equipment and environment.

Fig. 9: Mesh inside of Unity

IV. Results

Noisy data poses a significant problem. While trying to quantify our results by calculating distances between LiDAR points and mesh faces, we find the numerical accuracy of each meshing algorithm is not a great metric. Poisson methods in general are robust to noisy data [6], so aesthetically and qualitatively, its mesh reconstructions are cleaner. Poisson reconstruction also produces smooth results, and when compared to sharp LiDAR, its distance is high. BPA and greedy surface triangulation perform well quantitatively when compared to LiDAR, but do so by sacrificing visual coherency. If the input data were not noisy and uniformly distributed, BPA is expected to perform the best. Figure 6 shows the results on a close-up of a selected building.

The texture mapping algorithm can produce a photo-realistic model, but without depth testing, some ground textures are incorrect. Additionally, the illumination on textures could be different since they are pulled from different viewpoints and time, so a way to correct the differences is needed as the number of images used increases. Taller buildings that are not occluded show the best results.

With the Hololens 1 we are able to stream the mesh from Unity and grab, move, and rotate the hologram. This hologram can be put anywhere in the world, i.e. on a table, floor, or scaled 1:1 as if you were walking through the city. The results show that with Unity one can view the meshes in most popular VR environments. This is proven with the CAVE, shown in Figure 10. Various use cases for our model in the CAVE include a first-person view for planning drone flights, viewing already planned flights, and pure visualization of a city.

Fig. 10: Lab Members inside of CAVE at University of Missouri

V. Conclusion

To make more effective use of drone flight air time, we developed a pipeline to use aerial images of the area the drone will fly through to create a representative synthetic environment. This allows the optimal flight path to be found before setting foot in the specific location. Factors such as drone malfunctions, inclement weather, and high purchase costs will still be limiting factors of drones' usefulness, but making the most effective use of the drone's airtime minimizes flight risks. While the method proposed in this paper are ideal for drone flight planning experiments, it can have much wider applications: training simulations in specific environments, autonomous car navigation simulations, video games set in real cities, and 3D blueprints to help plan where to build roads and structures. The method is cost effective and simple. However, with every new implementation of this pipeline the final product will be limited by the quality of the original data, the size of the files that can be loaded into MeshLab and Unity, and the desired meshing algorithm to be used.

The pipeline for creating these representative virtual environments is comprised of four components: convert the point

cloud to a mesh using MeshLab and simplify the number of polygon faces, apply texture mapping using our method, import mesh layers into Unity game engine, and finally, configure the scene in Unity for the Hololens 1 and CAVE. Any mixed reality device that Unity supports can view the meshes. In addition, with a circular orbit aerial image data set, any specific location can be turned into a synthetic environment.

ACKNOWLEDGMENTS

Emily Lattanzio (High Point University) and Calvin Davis (Uiversity of Missouri) were supported by the NSF REU program at the University of Missouri EECS Department. The research was partially supported by the National Science Foundation under awards CNS-1950873 (REU) and CNS-2018850 (MRI) and U.S. Army Research Laboratory W911NF-1820285. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the U. S. Government or agency thereof.

REFERENCES

[1] D. Marini, R. Folgieri, D. Gadia, and A. Rizzi, "Virtual reality as a communication process," *Virtual Reality*, vol. 16, no. 3, p. 233–241, 2011. 1

[2] P. J. Durst, C. Goodin, C. Cummins, B. Gates, B. Mckinley, T. George, M. M. Rohde, M. A. Toschlog, and J. Crawford, "A real-time, interactive simulation environment for unmanned ground vehicles: The autonomous navigation virtual environment laboratory (anvel)," *2012 Fifth International Conference on Information and Computing Science*, 2012. 1

[3] Y. Chen, S. Chen, T. Zhang, S. Zhang, and N. Zheng, "Autonomous vehicle testing and validation platform: Integrated simulation system with hardware in the loop*," *2018 IEEE Intelligent Vehicles Symposium (IV)*, 2018. 1

[4] S. Yao, H. AliAkbarpour, G. Seetharaman, and K. Palaniappan, "3D patch-based multi-view stereo for high-resolution imagery," in *Geospatial Informatics, Motion Imagery, and Network Analytics VIII*, vol. 10645. International Society for Optics and Photonics, Apr. 2018, p. 106450K. 1, 2

[5] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin, "The ball-pivoting algorithm for surface reconstruction," *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, no. 4, pp. 349–359, Oct. 1999. 1, 2

[6] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," in *Proceedings of the fourth Eurographics symposium on Geometry processing*, ser. SGP '06. Cagliari, Sardinia, Italy: Eurographics Association, Jun. 2006, pp. 61–70. 1, 2, 6

[7] M. Kazhdan and H. Hoppe, "Screened poisson surface reconstruction," *ACM Transactions on Graphics*, vol. 32, no. 3, pp. 29:1–29:13, Jul. 2013. 1, 2

[8] Z. C. Marton, R. B. Rusu, and M. Beetz, "On fast surface reconstruction methods for large and noisy point clouds," in *2009 IEEE International Conference on Robotics and Automation*. Kobe: IEEE, May 2009, pp. 3218–3223. 1, 2

[9] A. Maiti and D. Chakravarty, "Performance analysis of different surface reconstruction algorithms for 3d reconstruction of outdoor objects from their digital images," *SpringerPlus*, vol. 5, no. 1, p. 932, 2016. [Online]. Available: https://doi.org/10.1186/s40064-016-2425-9 2

[10] M. Bosch, A. Leichtman, D. Chilcott, H. Goldberg, and M. Brown, "Metric Evaluation Pipeline for 3d Modeling of Urban Scenes," *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 42W1, pp. 239–246, May 2017. 2

[11] C. Poullis and S. You, "3d reconstruction of urban areas," *2011 International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission*, 2011. 2

[12] G. Kuschk, "Large scale urban reconstruction from remote sensing imagery," *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XL-5/W1, p. 139–146, 2013. 2

[13] T. Sky, "https://transparentsky.net/," *[Online; Accessed on July 22, 2021]*. 2

[14] H. Aliakbarpour, K. Palaniappan, and G. Seetharaman, "Robust camera pose refinement and rapid SfM for multiview aerial imagery – without RANSAC," *IEEE Geoscience and Remote Sensing Letters*, vol. 12, no. 11, 2015. 2

[15] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia, "MeshLab: an Open-Source Mesh Processing Tool," *Eurographics Italian Chapter Conference*, pp. 129–136, 2008. 2

[16] CloudCompare. (2021) GPL software. [Online]. Available: http://www.cloudcompare.org/ 2

[17] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *2011 IEEE International Conference on Robotics and Automation*, May 2011, pp. 1–4. 2

[18] M. Corsini, P. Cignoni, and R. Scopigno, "Efficient and Flexible Sampling with Blue Noise Properties of Triangular Meshes," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 6, pp. 914–924, Jun. 2012. 2

[19] P. Alliez, L. Saboret, and G. Guennebaud, "Poisson surface reconstruction," in *CGAL User and Reference Manual*, 5.3 ed. CGAL Editorial Board, 2021. [Online]. Available: https://doc.cgal.org/5.3/Manual/packages.html#PkgPoissonSurfaceReconstruction3 2

[20] H. X. Han and M. Zeiger, "The Local Z-Buffering Rendering," *IAENG International Journal of Computer Science*, vol. 32, no. 4, pp. 424–429, 2006. 5

[21] (2021, Feb.) Initializing your project and deploying your first application. [Online]. Available: https://docs.microsoft.com/en-us/windows/mixed-reality/develop/unity/tutorials/mr-learning-base-02 6