Networked and Multimodal 3D Modeling of Cities for Collaborative Virtual Environments

Benjamin Hall[1], Joseph Kessler[2], Osayamen Edo-Ohanba[3], Jaired Collins[4], Haoxiang Zhang[4], Nick Allegreti[4], Ye Duan[4], Songjie Wang[4], Kannappan Palaniappan[4], Prasad Calyam[4]
[1]Dept. of Engineering Physics, Murray State University, Murray, KY
[2]Computer Science Dept., Truman State University, Kirksville, MO
[3]Dept. of Computer Science Electrical Engineering, University of Missouri-Kansas City, Kansas City, MO
[4]Dept. of Electrical Engineering and Computer Science, University of Missouri, Columbia, MO

Abstract—3D city-scale models are useful in a number of applications, including education, city planning, navigation systems, artificial intelligence training, and simulations. However, final models need to be immersive and interactive, which requires a mixed reality (XR) environment design that combines e.g., a Cave Automatic Virtual Environment (CAVE) VR system with the Microsoft Hololens2 in a networked and multimodal setting. In this paper, we propose a pipeline to convert a city-scale point cloud into a finalized city-scale textured mesh in which, a number of XR devices can share the same environment and co-exist in a shared space for model interactions. Specifically, we use input point clouds obtained from wide area motion imagery systems or off-the-shelf drones pertaining to cities of Albuquerque, New Mexico; Columbia, Missouri; and Berkeley, California. Using four different traditional algorithms and an additional deep learning method, we create meshes for the model interactions. For each mesh produced, we map high-resolution textures onto them, producing a more accurate city, which is then passed into the shared/networked Unity environment. Ten participants provided their assessment of mesh quality and interactivity of the networked environment during exploration of different city reconstructions with the CAVE and laptop device modalities. Results on the perceptual immersive quality of the Point2Mesh deep learning meshes highlights the need for improvements to handle large city scale point clouds.

Index Terms—3D Reconstruction, Mixed Reality, Pipeline, 3D Meshes, Texture Mapping, Depth/Z buffering, Point Cloud, Multimodal

I. Introduction

Because of the rise in popularity and utilization of mixed reality (XR) devices in everyday life, there is a need to explore all of the possible applications. While there has been some research into the application of city visualization, such as Davis et al. [1], there is room for improvement in the meshing algorithms and the texturing process. The meshing algorithms are what turn a 3D point cloud, or a large set of points in 3D space, into a mesh that is more recognizable and usable. The texturing process applies the real life texture to this mesh, adding realism to the mesh system.

However, there is dearth of works on applying interactive city scale meshes in a multiplayer environment with multiple XR devices with different modalities, such as the Cave Automatic Virtual Environment (CAVE) system [2] and Microsoft Hololens2. In addition, there is need to study methods to create an interactive network that could support multiple people on

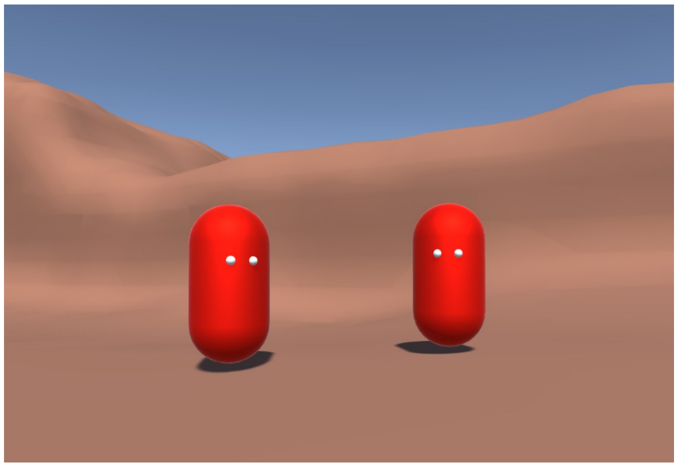


Fig. 1: Exemplar collaborative environment scene involving multimodal devices and two users that are represented by avatars during their 3D shared/networked space exploration.

different XR devices in which the entire meshes could be explored collaboratively. To address these needs, many challenges exist in transferring city meshes into such a networked and multimodal XR environment. One major issue relates to ensuring low communication latency between devices to support the interactivity. Additionally, for certain systems such as a HoloLens2, the rendering performance is tied to triangle count. For total immersion and realism, collision should be added to the environment in order to ensure the users walk on solid ground, and do not fall through the mesh. Lastly, a networking configuration needs to be designed e.g., using the Unity Mirror [3] capability in order to concurrently support a large amount of devices.

In this paper, we address above challenges by proposing a networked and multimodal XR environment design that features a pipeline to convert a city scale point cloud into a finalized textured mesh that is both immersive and interactive. Specifically, we use meshing algorithms for visualization efforts, such as Point2Mesh [4] that utilizes deep learning to improve their meshes over time, considering input point clouds of 3 different cities i.e., Albuquerque, NM; Columbia, MO; and Berkeley, CA. Our pipeline transformed these textured meshes

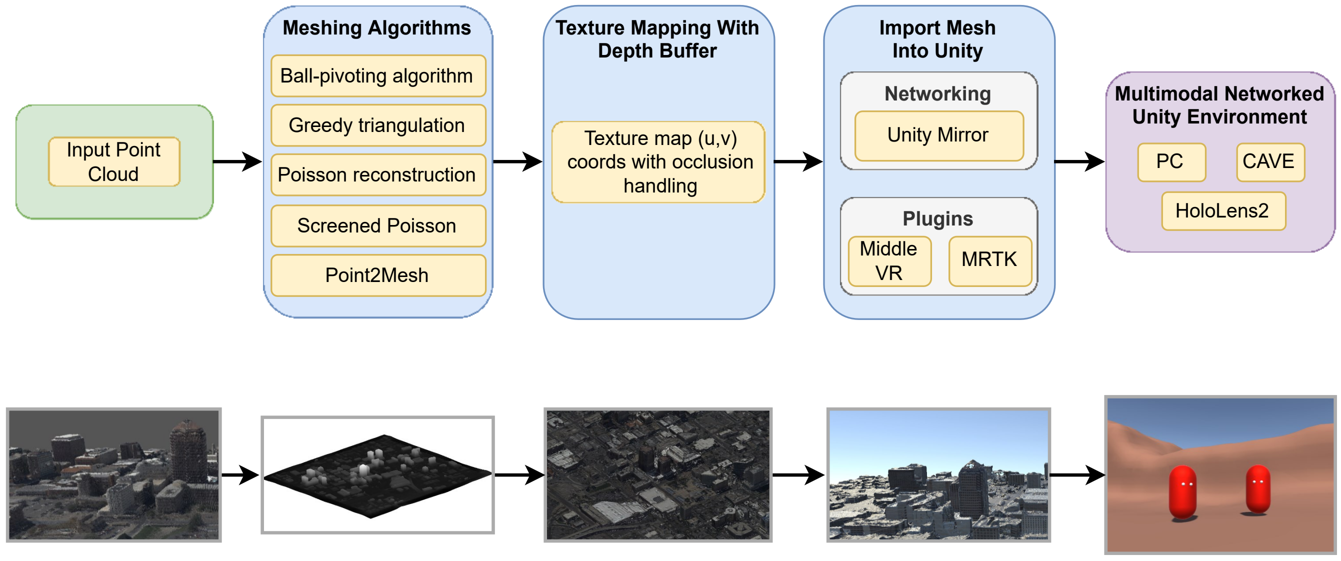


Fig. 2: Our proposed pipeline. A point cloud generated from Wide Area Motion Imagery (WAMI) or drone data is used as input into chosen meshing algorithms. Those meshes are then textured mapped with camera matrices that were bundle adjusted as part of the point cloud generation. A depth buffer is used to handle occlusion and reduce spurious textures, mostly observed on the ground near buildings. The textured mesh is imported into Unity, a game engine, with Unity Mirror for networking and Middle VR and MRTK plugins for multimodal devices. The final product is an environment where multiple devices can connect and interact within.

with an interactive shared/networked Unity environment that could be accessed and explored concurrently in a multi-user setting with both CAVE [2] and multiple laptop interfaces. Figure 1 shows one of the meshes and two users collaborating in our XR environment using multimodal devices; users can see and interact with each other, pointing out observations and working together.

To measure the effectiveness of the meshing algorithms that we utilized and the XR environment that we created, we organized a qualitative survey using our proposed pipeline shown in Figure 2. In the survey experiment, we had 2 groups of 5 participants using different devices including laptops and the CAVE system [2] to explore the meshes that we created. To measure the quality, the participants filled out a survey that inquired about the quality of their experience during the 3D model explorations. The results from this survey were then analyzed and normalized to create the results that we collected to measure how the different algorithms compared to each other as well as how much they would recommend the technology for wider adoption.

The remainder of the paper is as follows: Section II presents related works, Section III presents our methods for reconstruction and display of city scale point cloud meshes. Section IV presents the performance evaluation results. Section V concludes the paper.

II. Related Works

A. Meshing Algorithms

There are two different classes of meshing algorithms that we explored in this paper, including traditional and deep learning meshing algorithms. Traditional algorithms are algorithms that do not incorporate deep learning into the evaluation of point clouds. Bernardini et al. [5] is one of the earliest examples of this, using a virtual ball that rotates around the point selected and connects to points it touches, called ball-pivoting algorithm (BPA). Kazhdan and Hoppe [6] and Kazhdan et al. [7] are Poisson and screened Poisson, respectively. Screened Poisson regularizes over a group of points before executing the local and global fitting in the original Poisson reconstruction. In Csaba et al. [8], triangles are formed from edges created initially within the point cloud and it takes the first generation of these faces. Wongwaen et al. [9], mesheder et al. [10], Niu et al. [11], Jamin et al. [12] are other examples of this traditional approach, incorporating different ideas trying to overcome the difficulty of creating a fully accurate mesh from a set of points.

The deep learning method approaches to meshing incorporate trained networks in an effort to improve the accuracy of the generated meshes through multiple generations. Hanocka et al. [4] uses a convex hull of the initial point cloud, and over multiple generations shrinks the hull to the points and attempts to recognize patterns within the generated mesh and continue them. Some other examples of applying deep learning methods trained on previous point clouds and ground truths include Badki et al. [13], Lin et al. [14], and Lv et al. [15].

There are also different ways to evaluate the effectiveness of a generated mesh, as Berger et al. [16] attempts to implement such a method. It suggests a three phase pipeline for evaluating the accuracy of a reconstruction algorithm. In the first phase, it uses implicit surfaces for several differing surfaces (bumpy, smooth, sharp, etc.). From each surface, point clouds are synthetically created. Using the surfaces in combination with the synthetic point cloud, Berger uses Hausdorff distances to measure the accuracy of the reconstruction.

Our novel contributions to this area of mesh generation and viewing include applying a deep learning algorithm to city visualization, which, to the best of our knowledge, has not been explored yet.

B. XR Networking

The Visbox CAVE system [2] used in our experiments is a walk-in virtual reality environment with CAVE glasses and a handheld device used to communicate pose information and moving requests to the processing system for the 3D visuals. Our CAVE system consists of four 3 meter-by-3 meter sides and a projector for each side. In our exploration, we use it as an XR device as a part of our multiplayer environment used to view our generated meshes. We explored other XR devices such as the Microsoft Hololens2 [17], which runs on Mixed Reality Toolkit (MRTK). The Hololens2 is a mountable headset that can display interactive objects and programs in 3D on the glasses part of the device through the use of sensors on the top of the device. It is responsive to hand movements and incorporates hand gestures to make it interactive. The connect our XR devices together, we explored two different environments, including Unity Netcode and Unity Mirror. Unity Netcode is a networking library created by Unity that was built with the ease of use for developers in mind, using the Unity game engine via abstraction of the raw networking. Unity Mirror is what we ended up using to set up our multiplayer environment. Mirror is a high level networking environment that makes creating networked systems very quick. We found this one to be more effective in our area of work because it allowed for us to create a quick connection between many multimodal devices.

Our work demonstrates a novel pipeline to explore viewing these generated meshes in an interactive and collaborative environment, making viewing these models accessible to a wider audience. Our work also extends prior work in Davis et al. [1] in terms of creating meshes that use an improved version of a texture mapping function that features a depth buffer, and a multiplayer shared/networked environment using XR devices to view the newly generated meshes.

III. Methods for Reconstruction of Point Clouds and Display of City Scale Point Cloud Meshes

This section details our pipeline for creating multimodal 3D city models for collaborative virtual environments. First, a dense 3D point cloud is refined to be more conducive to good normal estimation and mesh generation. Then, the point cloud is used as input into meshing algorithms using freely available software packages. Afterward, our custom high-resolution texturing algorithm is applied. Finally, the textured 3D model is imported into a Unity environment with networking to facilitate collaboration. We demonstrate the portability of the software by running it within a CAVE VR system and multiple computers.

A. Mesh Creation

Aerial imagery collected by Transparent Sky [18] in an airplane is used, specifically Albuquerque, New Mexico (Figure 3a); Columbia, Missouri (Figure 4a); and Berkeley, California (Figure 5a). General aerial imagery collected by other vehicles such as drones can also be used. The images are used in the VB3D aerial multiview stereo algorithm [19] to create a dense point cloud. To improve the results from the meshing algorithms, a single layer of points is obtained by creating a voxel at every point and using a depth buffer at every camera location.

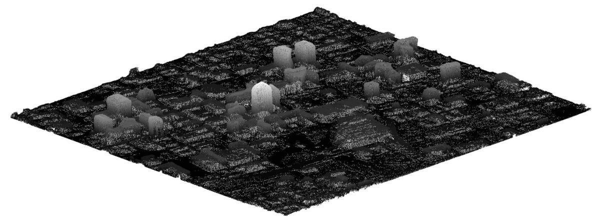
Each refined point cloud created from the cities are used in our chosen meshing algorithm, ball-pivoting algorithm [5], greedy triangle [8], Poisson reconstruction [6], screened Poisson reconstruction [7], and Point2Mesh [4]. A summary of each city and mesh is available in Figures 3, 4, and 5, where a grayscale height ramp is applied to provide a clearer view of the landscape in the reconstruction.

The ball-pivoting algorithm (BPA) uses a ball with a fixed radius to roll around a point cloud. Whenever three points are in contact with the ball, they are connected and form a triangle. Unless the input point cloud is uniform, it is highly likely that holes will be in the result. To close these, the authors recommend rerunning the algorithm with increasing radii [5]. Our chosen software that implements BPA, MeshLab [20], has a known bug that crashes the program on subsequent runs or if the input point cloud is too large. To that end, the results with BPA have been cropped and are much smaller than the full city.

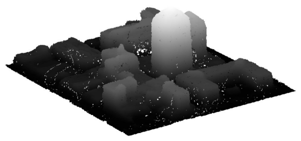
Greedy triangulation [8] was created for fast, online surface reconstruction. Each point is assigned a fixed number of neighbors, and then a weighted least squares plane is fitted to that neighborhood to estimate the surface normal. Points are pruned based on visibility, connected to the original point, and then connected to each other to form triangles. Several parameters for angles are used to reduce the smoothing of corners.

Poisson surface reconstruction casts mesh generation as a Poisson problem, which results in a global solution that is smooth and robust to noise [6]. The surface is extracted from oriented point samples by estimating an inside-outside indicator function and the marching cubes algorithm. The screened Poisson surface reconstruction [7] is similar to its predecessor, except a regularization term is added that weights the points in the global equation. The result is the surface retains more details and is less smooth, while still retaining its robustness to noise.

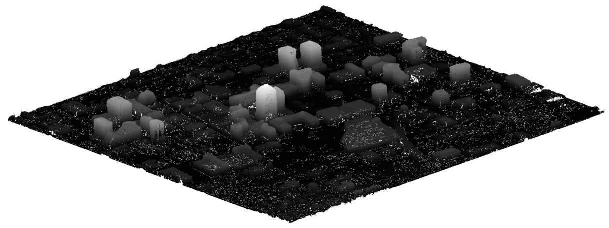
Finally, Point2Mesh [4] differs from the previous methods by using deep learning. It aims to recognize repeating patterns within the input point cloud. For sparse areas, Point2Mesh can interpolate the missing data to create a more complete mesh.



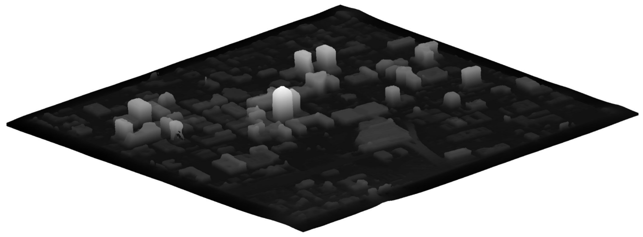
(a) Albuquerque, New Mexico point cloud



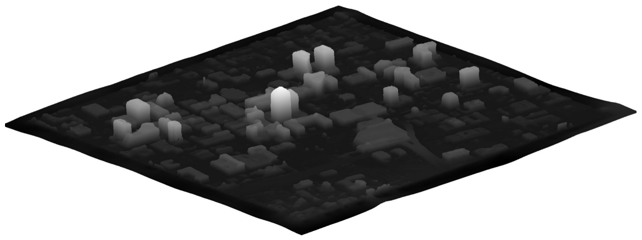
(b) Ball-pivoting algorithm on Albuquerque point cloud*



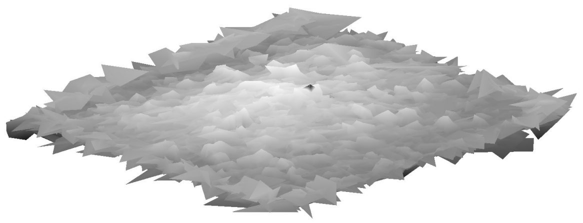
(c) Greedy Triangulation Algorithm on Albuquerque point cloud



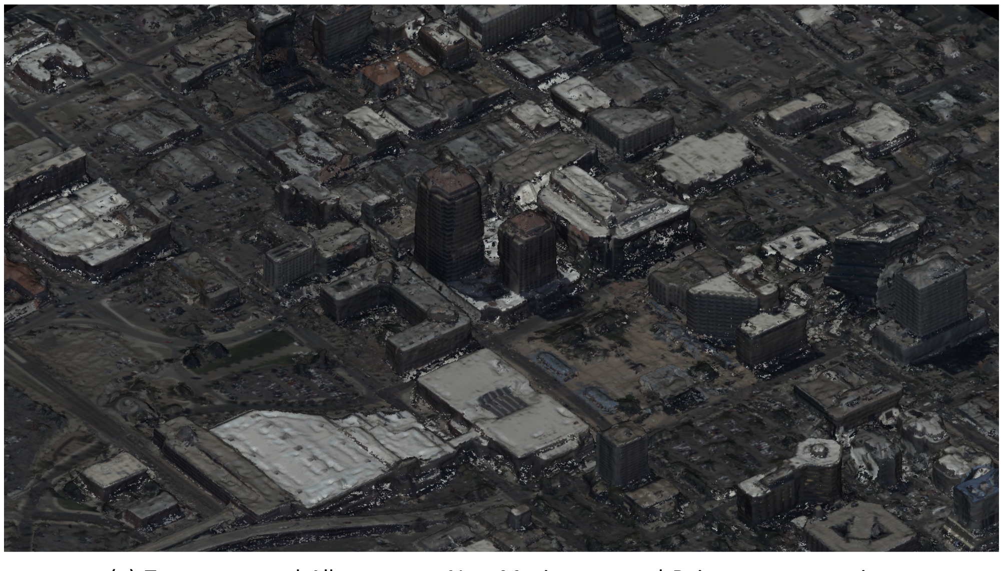
(d) Poisson Reconstruction Algorithm on Albuquerque point cloud



(e) Screened Poisson Reconstruction Algorithm on Albuquerque point cloud

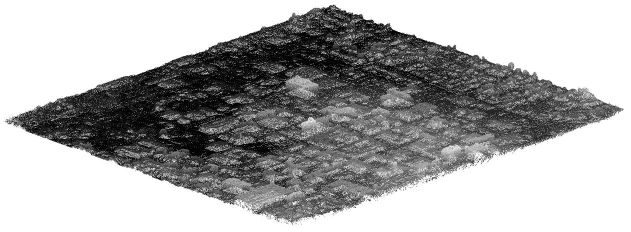


(f) Point2Mesh Algorithm on the Albuquerque point cloud

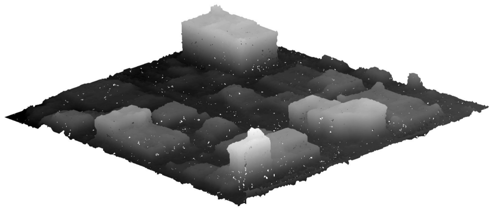


(g) Texture mapped Albuquerque, New Mexico screened Poisson reconstruction

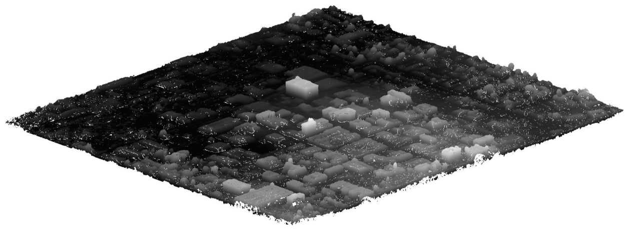
Fig. 3: Visual Comparison between point clouds and their respective meshes for Albuquerque, New Mexico. Note: The BPA input point cloud was cropped to a much smaller set due to a known bug that crashes MeshLab.



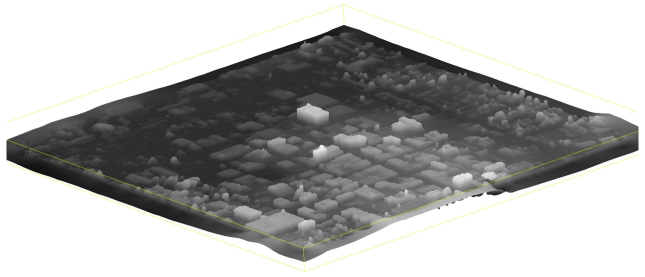
(a) Columbia, Missouri point cloud



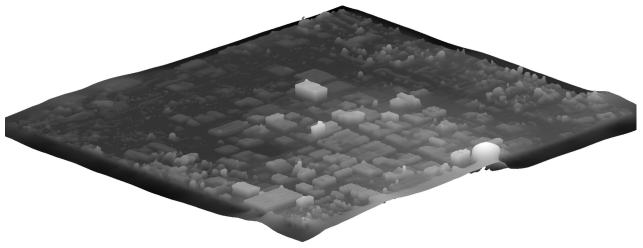
(b) Ball-pivoting algorithm on Columbia point cloud*



(c) Greedy Triangulation Algorithm on Columbia point cloud

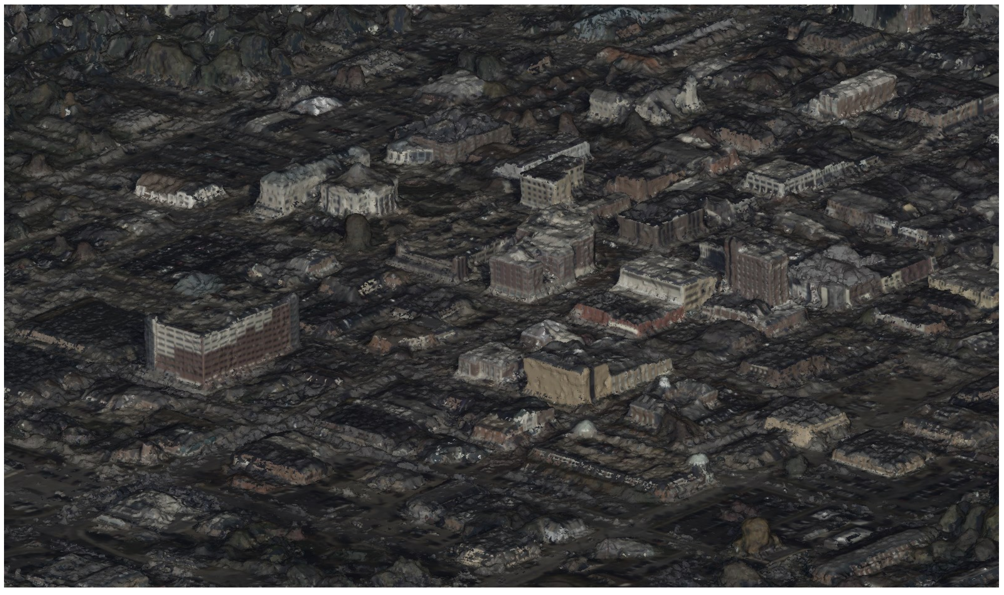


(d) Poisson Reconstruction Algorithm on Columbia point cloud



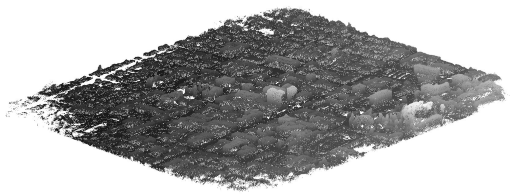
(e) Screened Poisson Reconstruction Algorithm on Columbia point cloud

(f) Point2Mesh Algorithm on the Columbia point cloud

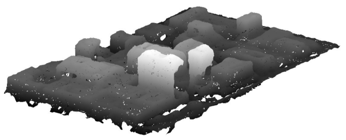


(g) Texture mapped Columbia, Missouri screened Poisson reconstruction

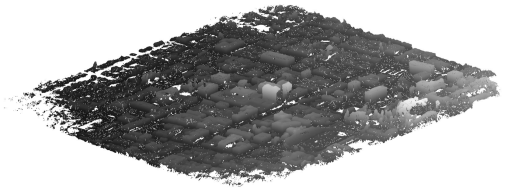
Fig. 4: Visual Comparison between point clouds and their respective meshes for Columbia, Missouri. Note: The BPA input point cloud was cropped to a much smaller set due to a known bug that crashes MeshLab.



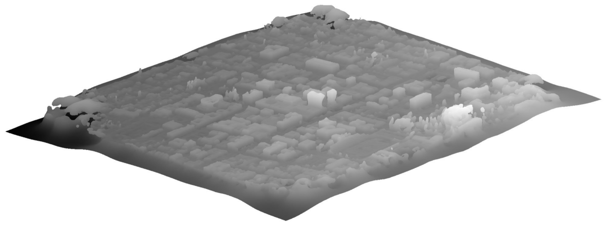
(a) Berkeley, California point cloud



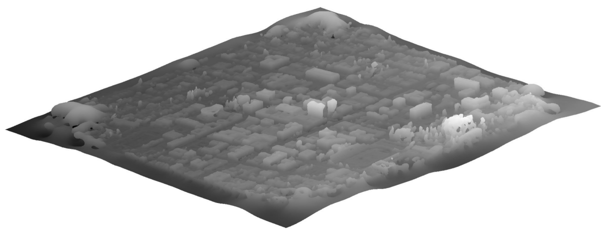
(b) Ball-pivoting algorithm on Berkeley point cloud*



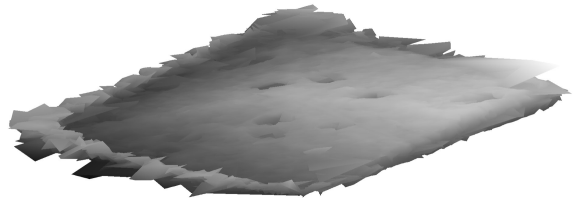
(c) Greedy Triangulation Algorithm on Berkeley point cloud



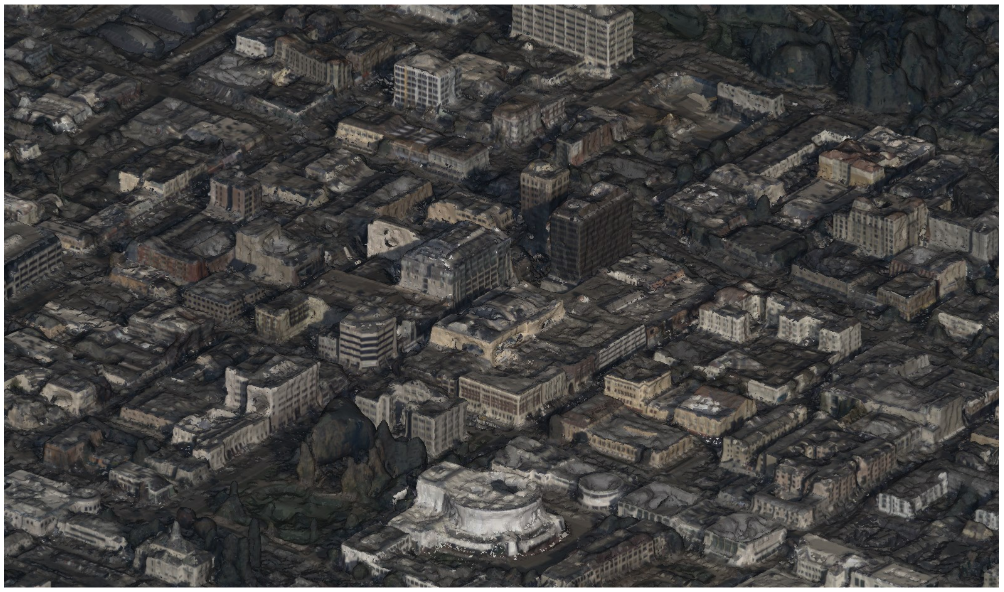
(d) Poisson Reconstruction Algorithm on Berkeley point cloud



(e) Screened Poisson Reconstruction Algorithm on Berkeley point cloud



(f) Point2Mesh Algorithm on the Berkeley point cloud



(g) Texture mapped Berkeley, California screened Poisson reconstruction

Fig. 5: Visual Comparison between point clouds and their respective meshes for Berkeley, California. Note: The BPA input point cloud was cropped to a much smaller set due to a known bug that crashes MeshLab.

Algorithm 1 Face Assignment

```
1: Input
2:     $M$    mesh with faces in winding order
3:     $n$    number of cameras for texturing
4:     $P_n$  array of cameras, size $n$
5:     $Z_n$  array of camera look-at directions, size $n$
6:     $w$    image width
7:     $h$    image height
8: Output
9:     $M_f$  mesh with assigned faces
10:
11: AssignFaces($M, n, Z_n$)
12: InitializeDepthBuffers($M, P_n, Z_n, w, h$)
13:
14: procedure AssignFaces($M, n, Z_n$)
15:     for all face $f$ 2 $M$ do
16:         $v_0, v_1, v_2$   vertex 0, 1, 2 2 $f$
17:         $u$   $v_1$   $v_0$
18:         $v$   $v_2$   $v_1$
19:         $p$   $u \rightarrow v / |u \rightarrow v|$        . Face $f$'s normal
20:         $i$   $\mathrm{argmin}\{p \cdot z, z$ 2 $Z_n\}$
21:         $M_{f,assigned}$   $i$   . Face $f$ 2 $M$ assigned camera $i$
22:     end for
23: end procedure
24:
25: procedure InitializeDepthBuffers($M, P_n, Z_n, w, h$)
26:     initialize $D_n$ with size $w, h$
27:     for all $f$ 2 $M_f$ do
28:         $i$   $M_{f,assigned}$
29:         for $v_0, v_1, v_2$ 2 $f$ do        . Vertices in the face
30:             convert $v_0, v_1, v_2$ to homogeneous
31:             $(x_0, y_0)$   $P_i v_0$     . Project $v_0$ to camera $i$
32:             $(x_1, y_1)$   $P_i v_1$     . Project $v_1$ to camera $i$
33:             $(x_2, y_2)$   $P_i v_2$     . Project $v_2$ to camera $i$
34:         end for
35:         $L, X, Y$    interpolate depths $L$ from implicit
    triangle rasterization
36:         for all $l$ 2 $L$ and $x, y$ 2 $X, Y$ do
37:             if $l > D_i(x, y)$ then
38:                 ReassignFace($i, f, M, Z_n$)
39:                 break      . Exit loop. Face is reassigned
40:             end if
41:         end for
42:     end for
43: end procedure
44:
45: procedure ReassignFace($i, f, M, Z_n$)
46:     $v_0, v_1, v_2$   vertex 0, 1, 2 2 $f$
47:     $u$   $v_1$   $v_0$
48:     $v$   $v_2$   $v_1$
49:     $p$   $u \rightarrow v / |u \rightarrow v|$          . Face $f$'s normal
50:     $j$   $\mathrm{argmin}\{p \cdot z | z$ 2 $Z_n, i$ 2̸ $Z_n\}$
51:     $M_{f,assigned}$   $j$   . Face $f$ 2 $M$ is assigned camera $j$
52: end procedure
```

Algorithm 2 Generate Texture Map Coordinates for Triangles

```
1: Input
2:     $M$    mesh with assigned faces
3:     $n$    number of cameras for texturing
4:     $P_n$  camera matrices associated with desired images
5:     $w$    image width
6:     $h$    image height
7: Output
8:     $T$    textured mesh coordinates
9: for all face $f$ 2 $M$ do
10:     $i$   $M_{f,assigned}$            . Index of camera for face
11:     for all vertex $v$ 2 $f$ do
12:         convert $v$ to homogeneous
13:         $(u, v)$   $P_i v$        . Project and dehomogenize
14:         $u$   $u/w$                       . Normalize
15:         $v$   $v/(1$   $(v/h))$          . Normalize and flip
16:         $T_f$.add$(u, v)$        . Append $(u, v)$ to $T$ for $f$
17: end for
18: end for
```

B. Texture Mapping/Depth Buffer

To create a realistic 3D model of a city, the meshes should be textured. The previous work by Davis et al. [1] did not handle occlusion, so the ground near buildings were textured incorrectly. To remedy this, we implemented a depth buffer to improve texture quality and handle occlusion. Algorithms 1 and 2 start with an untextured 3D mesh and corresponding images from different angles of the area of the mesh and outputs pixel assignments for each face of the mesh.

Algorithm 1 starts with each face f and calculates its normal vector. After calculating this normal, it computes projection of the normal with the camera view vector i. The most negative value represents the camera that is most able to view the face. Therefore, it assigns face f to camera i for pixel assignment. Afterwards, a depth buffer is initialized with the width and height of the image resolution. Each assigned face's vertices are projected onto the camera using the camera matrix for the assigned camera. It then interpolates the depth of the face using implicit triangle rasterization and compares it to the existing value in the depth buffer matrix. If the calculated depth is smaller than the depth buffer, then it reassigns the camera to the face with the distance. Afterwards, the face is assigned to the camera with the next most negative dot product.

Then, the texture map generation function shown in Algorithm 2 uses the assigned faces and meshes to map (u, v) coordinates. It iterates through each vertex v in each face f and its assigned camera i, and converts the coordinate to homogeneous coordinates. It then projects the homogenized v onto the camera with its matrix P_i. Subsequently, it normalizes the coordinates and appends the location of the pixel for the vertex to the textured mesh coordinates T.

C. Mesh to Networked Unity Environment

The textured meshes were then put into a Unity environment to produce a first-person, player-to-player, and networked 3D

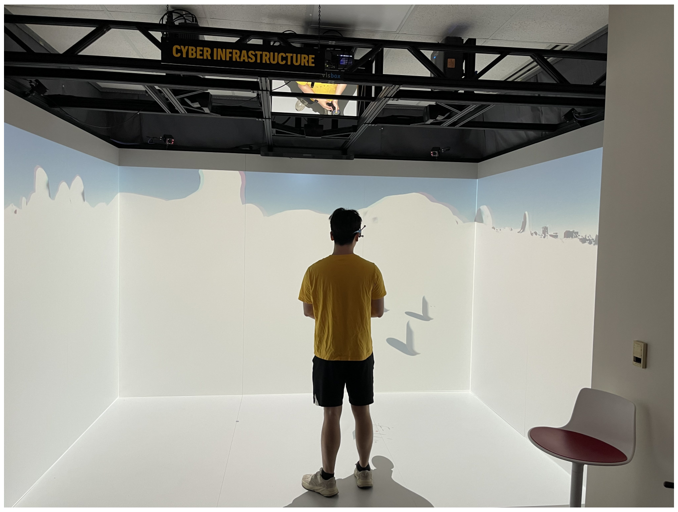


Fig. 6: User in the CAVE during an experiment in the multi-player environment setup with the Unity Mirror capability.

city experience as shown in Figure 6. In order to achieve networking between multimodal devices, we created platform-dependent build files to distribute among platforms. Unity supports networking with its own in-house libraries, but we found that documentation and maturity are lacking. Instead, we use Unity Mirror [3], which has more support and works well out of the box. Through Unity Mirror, with a properly defined player prefab, network manager, and network HUD, a networked environment was produced and distributed to our target platforms.

Pseudocode of Mirror is shown in Algorithm 3. When a server starts, a network manager is created on the server. This facilitates the communication and synchronization between clients. Once a client requests access to the server, the client registers itself with the network manager and then starts a loop until it disconnects. The client processes network messages, updates synchronization variables, calls Unity's standard update methods, and then sends its own networks to the network manager.

For the CAVE [21], Middle VR was used in addition to the networking libraries. There was much more time devoted to the networking of the HoloLens2 device, but the HoloLens2 did not run similarly to the Visbox CAVE because of their differing software, MRTK—specifically for head-mounted devices–while the Middle VR software could only be run on a computer. Due to this, there was a huge hindrance in the ability to integrate the HoloLens2 into the environment.

From the finished product, a number of different users—limited to 100—could access and see the same mesh and interact with other users in the environment. Assessment is hard, since we did not have ground truth 3D cities. Additionally, no reliable quantitative measure was found. To remedy this, we conducted a qualitative survey with ten different individuals as detailed in Section IV. They were asked to explore the environment that we created, and then asked to rate their experience in different aspects, such as the quality of the meshes, their virtual reality experience, and quality of the collaborative networking.

Algorithm 3 Unity Mirror Networking

```
Input
    S   Server
    C   Client
    N   Mirror network manager

initialize server S
OnServerStart(S, N)

procedure OnServerStart(S, N)
    Initialize network manager N on server S
    while S is listening for clients C do
        if C requests access to S then
            C initializes networked widgets with N
            C is placed in the hosted world of S
            ClientLoop(N, C)
        end if
    end while
end procedure

procedure ClientLoop(N, C)
    while C is connected to N do
        Process network messages from N
        Update synchronization variables
        Unity FixedUpdate, Update, LateUpdate
        Send network messages to N
    end while
end procedure
```

IV. Performance Evaluation

We applied five different meshing algorithms to three different point clouds that were collected previously from Transparent Sky [18]. In Figures 3, 4, and 5 for Albuquerque, New Mexico; Columbia, Missouri; and Berkeley, California, respectively, we can see the original point cloud followed by the five different meshing algorithms, and then a textured version of the Screened Poisson Reconstruction meshes. For the ball pivoting algorithm in each city, due to the constraints of the software, we were unable to run it for the whole city point cloud. Instead, we took a crop from a part of the city that contained a few buildings and meshed the smaller point cloud.

The depth buffer texturing algorithm shows improvement over Davis et al. [1] by removing most occluded artifacts. That is, a building's texture is mostly unseen on the ground around it. However, the illumination between the different images used to texture the models varies heavily, which reduces the accuracy and immersion of the overall model. Large faces without noise that mostly face any chosen image look the best. Otherwise, a global illumination model can be used to better smooth the textures in the future.

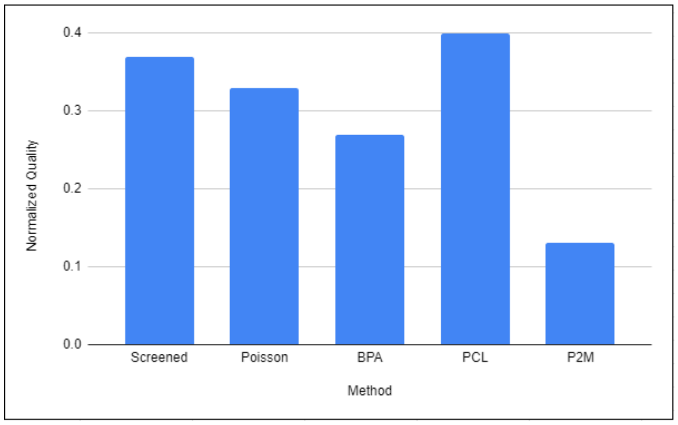


Fig. 7: The sum of the Qualitative rating for each of our methods on the Albuquerque point cloud. The higher the number, the better the quality.

A qualitative test was performed to measure the effectiveness of our pipeline. We asked ten different individuals to participate in our experiment to gauge how the different 3D models look compared to each other. The five different reconstructed models from Albuquerque, New Mexico were placed in the same Unity environment and asked the participants to explore for five minutes. They were then surveyed for their impressions and asked to rank the models. The results obtained are presented as a sum of the qualitative rating of each mesh as shown in Figure 7. With x_n representing the score from 1-10 for every algorithm from the n^{th} participant and p representing the number of participants, we can normalize the quality for each mesh, shown in Eq. 2.

$$\text{Quality} = \sum_{n=1}^{p} x_n \tag{1}$$

$$\text{NormalizedQuality} = \frac{\text{Quality}}{10p} \tag{2}$$

We observe that the highest quality mesh according to our metric was made with greedy triangulation [8]. Even though there are more holes in this mesh than the other ones, the reason that people thought more positively about this method is because of the way that it represented the city. It was more recognizable from far away, which is something that was an issue. The reason being that - while they were in the multiplayer environment, there was no collision where there were holes. This meant that the participants could fall through the map, leading to a less effective analysis of the mesh.

V. Conclusion

Creating meshed and textured city-scale 3D models in a multimodal and networked environment provides unique challenges that are hard to overcome with any single algorithm. Multiple points in the process can be scrutinized and improved, especially meshing algorithms and texturing. For meshing, we test four popular algorithms and a deep learning method—Point2Mesh—showing that usable models can be obtained with free and open source software. While deep learning is very promising, our own tests with city data and Point2Mesh severely underperformed compared to traditional methods.

The 3D meshes were created in a coordinate space that allowed easy texture mapping. Each point was projected to each selected camera and a depth buffer was added to remove spurious textures. Without such a buffer, areas around the ground of buildings were incorrectly mapped since occlusions were not being tested. To measure mesh quality, we asked ten participants to explore a networked Unity environment on several computers and a CAVE. They were surveyed for their experience, and obtained results show that greedy triangulation and screened Poisson are the best, perhaps due to the surfaces being closely tied to the points. For regular Poisson reconstruction, it is common to have the surface stray away from the underlying points. BPA only connects points, but is unpopular due to the noise caused by our input data. Our results also show that Point2Mesh is the least popular, which is not surprising given that the output was mostly incoherent to the users.

Future work can build on our work that demonstrates how a pipeline can be designed to transform point clouds into a networked and multimodal environment using freely available software. Extended results can focus on how a custom texture mapping with a depth buffer can provide highly detailed city-scale reconstructions. Further, networking code and platform-specific plugins such as Middle VR or MRTK can be added to better integrate Hololens2 in visualization experiments.

Acknowledgements

Benjamin Hall (Murray State University), Joseph Kessler (Truman State University), and Osay Edo-Ohanba (University of Missouri-Kansas City) 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 NSF CNS-2018850 (MRI). 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] C. Davis, J. Collins, J. Fraser, H. Zhang, S. Yao, E. Lattanzio, B. Balakrishnan, Y. Duan, P. Calyam, K. Palaniappan, and et al., “3d modeling of cities for virtual environments,” 2021 IEEE International Conference on Big Data (Big Data), 2021.

[2] R. Tredinnick, B. Boettcher, S. Smith, S. Solovy, and K. Ponto, “Uni-cave: A unity3d plugin for non-head mounted vr display systems,” in 2017 IEEE Virtual Reality (VR), 2017, pp. 393–394.

[3] “Open source networking for unity.” [Online]. Available: https://mirror-networking.com/

[4] R. Hanocka, G. Metzer, R. Giryes, and D. Cohen-Or, “Point2mesh: A self-prior for deformable meshes,” ACM Trans. Graph., vol. 39, no. 4, 2020. [Online]. Available: https://doi.org/10.1145/3386569.3392415

[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, 1999.

[6] M. Kazhdan, M. Bolitho, and H. Hoppe, “Poisson surface reconstruction,” in Proceedings of the fourth Eurographics symposium on Geometry processing, vol. 7, 2006.

[7] M. Kazhdan and H. Hoppe, “Screened poisson surface reconstruction,” ACM Transactions on Graphics (ToG), vol. 32, no. 3, pp. 1–13, 2013.
[8] Z. Csaba, R. B. Marton, M. Beetz et al., “On fast surface reconstruction methods for large and noisy point clouds,” in Robotics and Automation, 2009. ICRA’09. IEEE International Conference on, 2009, pp. 3218–3223.
[9] N. Wongwaen, S. Tiendee, and C. Sinthanayothin, “Method of 3d mesh reconstruction from point cloud using elementary vector and geometry analysis,” in 2012 8th International Conference on Information Science and Digital Content Technology (ICIDT2012), vol. 1. IEEE, 2012, pp. 156–159.
[10] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger, “Occupancy networks: Learning 3d reconstruction in function space,” in Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2019, pp. 4460–4470.
[11] Y. Niu, J. Chen, X. Ke, and J. Chen, “Stereoscopic image saliency detection optimization: A multi-cue-driven approach,” IEEE Access, vol. 7, pp. 19 835–19 847, 2019.
[12] C. Jamin, P. Alliez, M. Yvinec, and J.-D. Boissonnat, “Cgalmesh: a generic framework for delaunay mesh generation,” ACM Transactions on Mathematical Software (TOMS), vol. 41, no. 4, pp. 1–24, 2015.
[13] A. Badki, O. Gallo, J. Kautz, and P. Sen, “Meshlet priors for 3d mesh reconstruction,” in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 2849–2858.
[14] C.-H. Lin, O. Wang, B. C. Russell, E. Shechtman, V. G. Kim, M. Fisher, and S. Lucey, “Photometric mesh optimization for video-aligned 3d object reconstruction,” in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 969–978.
[15] C. Lv, W. Lin, and B. Zhao, “Voxel structure-based mesh reconstruction from a 3d point cloud,” IEEE Transactions on Multimedia, vol. 24, pp. 1815–1829, 2021.
[16] M. Berger, J. A. Levine, L. G. Nonato, G. Taubin, and C. T. Silva, “A benchmark for surface reconstruction,” ACM Transactions on Graphics (TOG), vol. 32, no. 2, pp. 1–17, 2013.
[17] D. Ungureanu, F. Bogo, S. Galliani, P. Sama, X. Duan, C. Meekhof, J. Stühmer, T. J. Cashman, B. Tekin, J. L. Schönberger, P. Olszta, and M. Pollefeys, “Hololens 2 research mode as a tool for computer vision research,” ArXiv, vol. abs/2008.11239, 2020.
[18] Transparent Sky, “https://transparentsky.net/,” [Online; Accessed on July 22, 2021].
[19] 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.
[20] 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.
[21] Visbox. [Online]. Available: http://www.visbox.com/