Visualization of Point Clouds in Mobile Augmented Reality using Continuous Level of Detail Method

04-11-2020

Liyao Zhang

Mentor #1: Peter van Oosterom

Mentor #2: Haicheng Liu

Faculty of Architecture and The Built Environment, TU Delft, The Netherlands

TU Delft

Overview

1. Introduction
2. Methodology
3. Implementation
4. Results
5. Analysis
6. Comparison
7. Conclusion

Introduction – Augmented Reality

Augmented Reality (AR)

- An Augmented Reality system is a system that has the following properties (Azuma et al., 2001):
  - combines real and virtual objects in a real environment;
  - runs interactively, and in real-time;
  - aligns real and virtual objects with each other.
- AR applications can be used on mobile devices without specific equipment like helmets and handles.

(Pokémon GO)

Introduction - Motivation

Reasons of showing point clouds in mobile Augmented Reality using cLoD method:

- Point clouds have become important data resources of multiple fields, however, the use of point clouds in mobile AR is waited to be explored.
- Save a lot of time and resources if we can directly get use of point clouds in mobile AR: some pre-processing steps can be avoided.
- Large point clouds can't be visualized without LoD support, and will be visualized ugly with dLoD approaches.

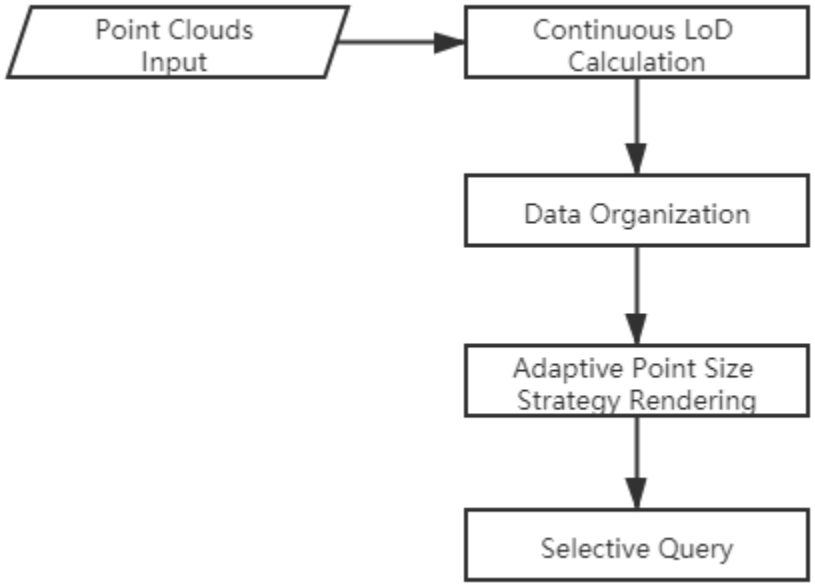
Challenges of showing point clouds in mobile AR using cLoD method:

- Dealing with huge-amount data of point cloud datasets with limited memory, CPU and GPU resources of mobile devices
- Reaching relatively high visual quality and performance requirements
- CLoD based visualization has not been used in mobile AR before

Overview

1. Introduction
2. Methodology
3. Implementation
4. Results
5. Analysis
6. Comparison
7. Conclusion

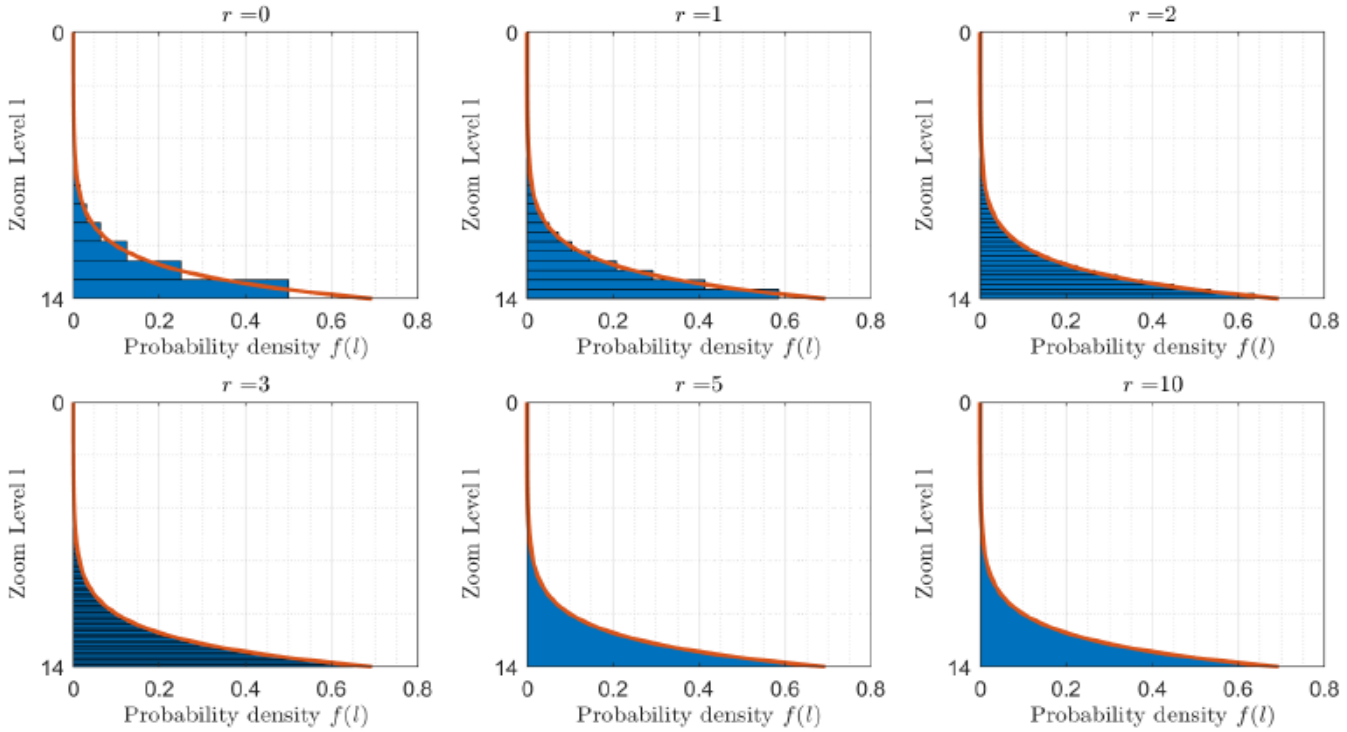
Methodology - Overview



Methodology – cLoD Calculation

This cLoD model is developed based on the idea of refining ideal discrete LoDs and making them be a continuous function.

Refined discrete levels (max level L = 13), r: refinement, r = 2^(1/2^r)



blue bars: refined discrete level-of-detail, red curve: continuous function (van Oosterom, 2019)

Methodology – cLoD Calculation

For ideal continuous function over levels (nD):

$$f(l,n) = \frac{2^{(n-1)l}(n-1)ln2}{2^{(n-1)(L+1)} - 1}$$

This function has Cumulative Distribution Function (CDF):

$$F(l,n) = \frac{2^{(n-1)l} - 1}{2^{(n-1)(L+1)} - 1}$$

When the number of sublevels approaches infinity, the CDF can be seen as continuous function. So, the inverse function of F(l,n) together with random generator U (uniform between 0 and 1) is used to generate continuous level l for points in nD space:

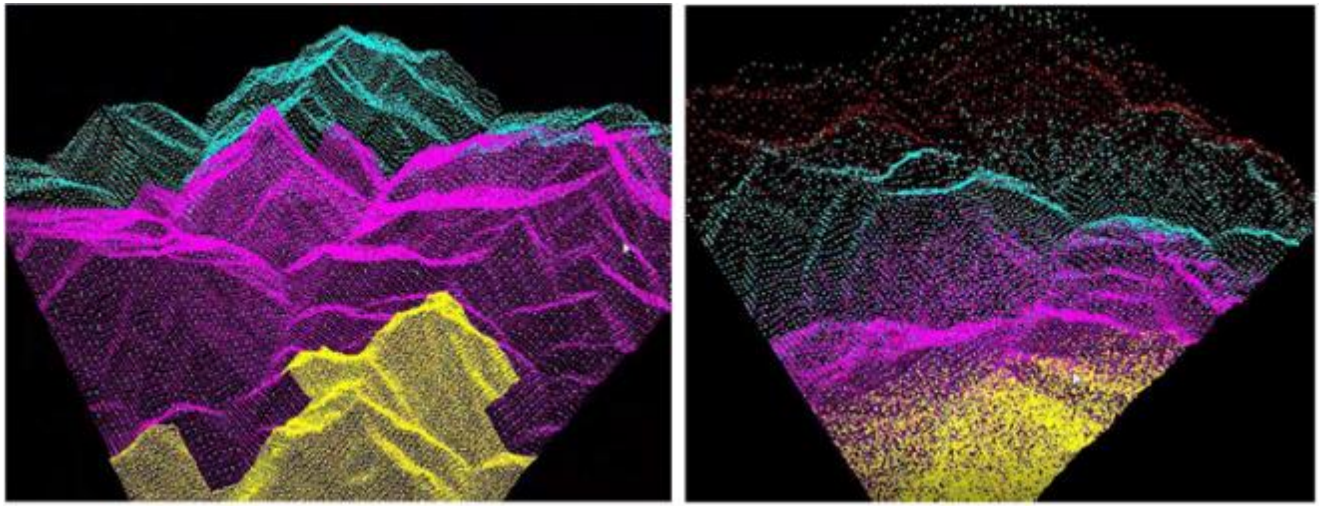
$$l = F^{-1}(U) = \frac{\ln\left(\left(2^{(n-1)(L+1)} - 1\right)U + 1\right)}{(n-1)ln2}$$ (van Oosterom, 2019)

- L – the max level of detail
- l – levels between 0 and L+1
- n – number of dimensions

Methodology – cLoD Calculation

Properties of cLoD model:

- This cLoD model has ideal distribution over LoDs
- Can realize smooth transition in density, avoid density shocks as present in discrete LoD approaches
- Keeps the desired relative point density as much as possible



dLoD (left) and cLoD(right) (Guan, 2019)

Methodology – Adaptive Point Size Strategy Rendering

Issues of showing points as the same size:

- If the size is too small, then there will be obvious holes between points;
- If the size is too big, the neighboring points will overlap a lot and cause a loss of information.

Points with different sizes (left) and with same size (right)

Methodology – Adaptive Point Size Strategy Rendering

Therefore, in order to get better visual quality, we'll use the Adaptive point size strategy, which sets the point size of each point as different values. Based on the perspective projection matrix, we derive a formula to calculate ideal point sizes at different depth in the viewing z-axis direction.

$$size = \frac{s * n * r * screenHeight}{z_{eye} * \tan(0.5 * fov)}$$

r = right coordinate of near clipping plane
s = coefficient to scale the points
n = near clipping plane distance

fov = field of view
screenHeight = height of screen
z_{eye} = point depth in the viewing z-axis direction

Methodology – Selective Query

- The computation in point-wise is too expensive for mobile devices and will cause an extremely low frame rate and even software crashes. Thus a uniform threshold over cLoD is used to filter the points.
- B+ tree is applied to speed up the query on the cLoD value.
- The main idea of filtering the points is to reach an ideal point cloud density for display at certain distance.
- The Cumulative Density (CD) at a certain level can be obtained from the Cumulative Distribution Function.

$$CD(l,n) = \frac{F(l,n)N}{E^n} = \frac{\left(2^{(n-1)l} - 1\right)N}{(2^{(n-1)(L+1)} - 1)E^n}$$

l = continuous level
N = the total number of points in the dataset
n = number of dimension
E^n = size of spatial domain in nD case

Methodology – Selective Query

- The value of ideal density is chosen based on the ideal point sizes at each depth in the adaptive point size strategy rendering step.

- A logarithm of distance from the center of the point cloud model to the camera is set as the denominator. -> Higher density when the model is nearby, and lower density when the model is far away.

- By visualizing all the points with level less than l, we can reach the wanted density.

$$CD(l,n) = \frac{D}{ln\sqrt{((x-u)^2+(y-v)^2+(z-w)^2)+1}}$$

l = continuous level
n = number of dimension
D = ideal density
x, y, z = coordinates of the point cloud center in world space
u, v, w = camera coordinates in world space

Overview

1. Introduction
2. Methodology
3. Implementation
4. Results
5. Analysis
6. Comparison
7. Conclusion

Implementation – Tools

Software

- ARCore (version 1.17.0)
- Unity game engine (version 2018.4.21)

Language

- C#
- LAZ file system in C# - LASzip
- High-Level Shading Language (HLSL)

Hardware

- The tests and benchmarks are carried on a *Redmi K20 Pro* model.
- Qualcomm Snapdragon 855 processor at 2.84 GHz, 8 GB of Random Access Memory (RAM), 2340 x 1080 pixels resolution, and a triple-camera setup.

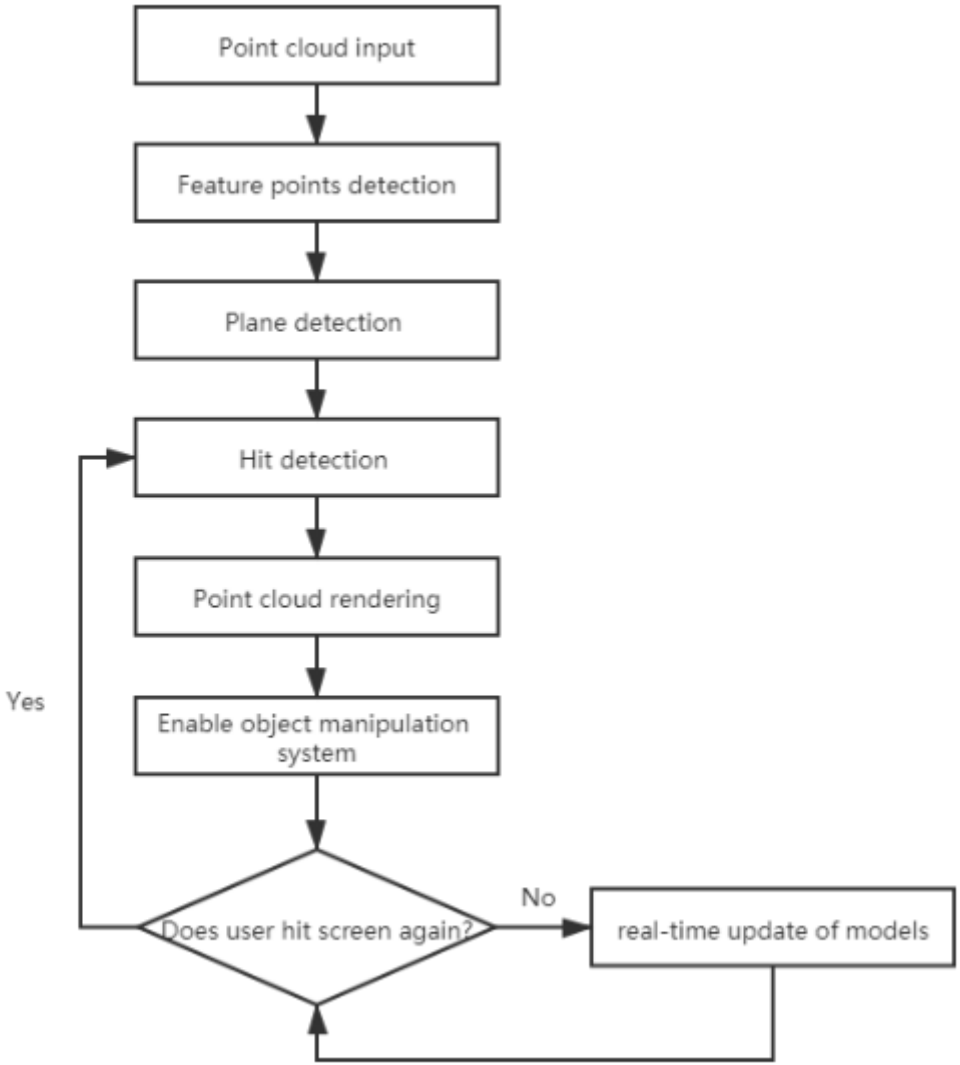
Implementation – Datasets

- **Furniture Point Clouds:** point clouds of furniture, such as chair, table, and sofa.

- **Architecture Point Clouds:** Point clouds of an underground garage, obtained by the NAVVIS M6 indoor mobile mapping system. Available at: https://www.navvis.com/m6-pointclouds

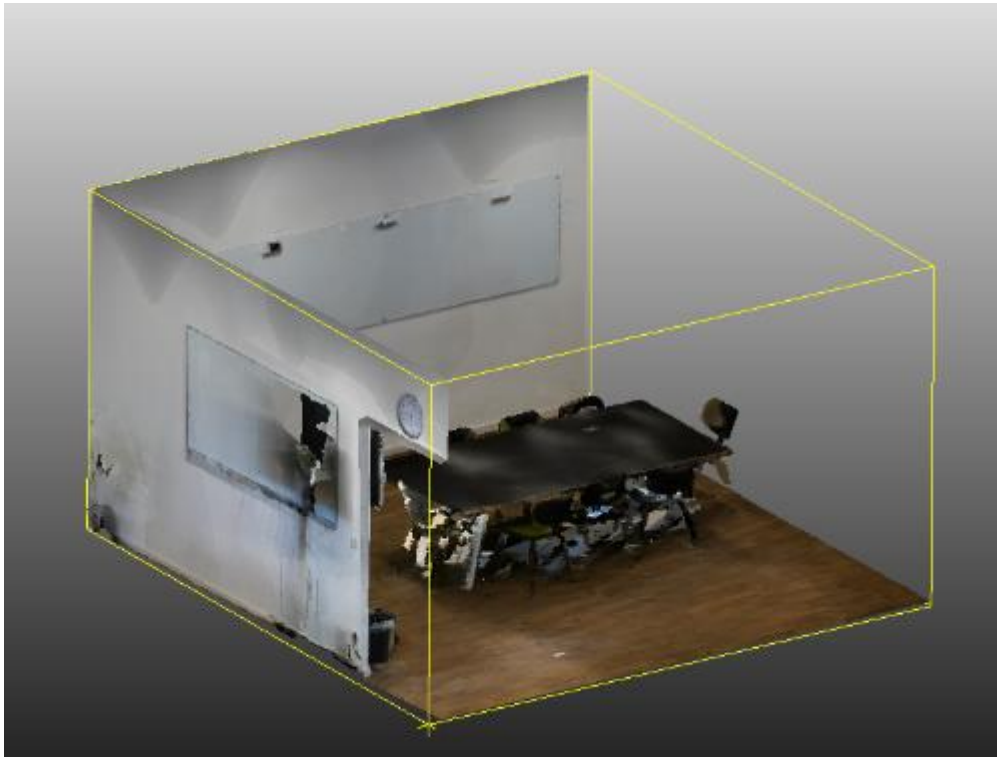
- **Terrain Point Clouds:**
  - *NEON AOP Discrete Return Light Detection and Ranging (LiDAR) Point Cloud*, which is an American Society for Photogrammetry and Remote Sensing (ASPRS) LASer format data product in UTM map projection. Available at: https://data.neonscience.org/data-products/DP1.30003.001;
  - *AHN2 (Actueel Hoogtebestand Nederland) dataset*, which is the digital elevation map for the whole Netherlands. Available for download via the PDOK (Publieke Dienstverlening Op de Kaart).

Implementation - Overview



Implementation – Point Cloud Input and Storage

- The point clouds are stored as LAZ files in order to reduce the file size and speed up loading. A C# library called LASzip is used to read the LAZ files.
- The minimum and maximum x, y, z coordinates of the point cloud are first read from the header of the LAZ file and waiting to be used in the later transformation.
- The coordinates, colour and calculated cLoD of each point are stored as separated arrays, and are sorted based on the cLoD value in ascending order. The cLoD array is then organized using a B+ tree to speed up the selective query on the continuous levels.



Implementation – Hit Detection

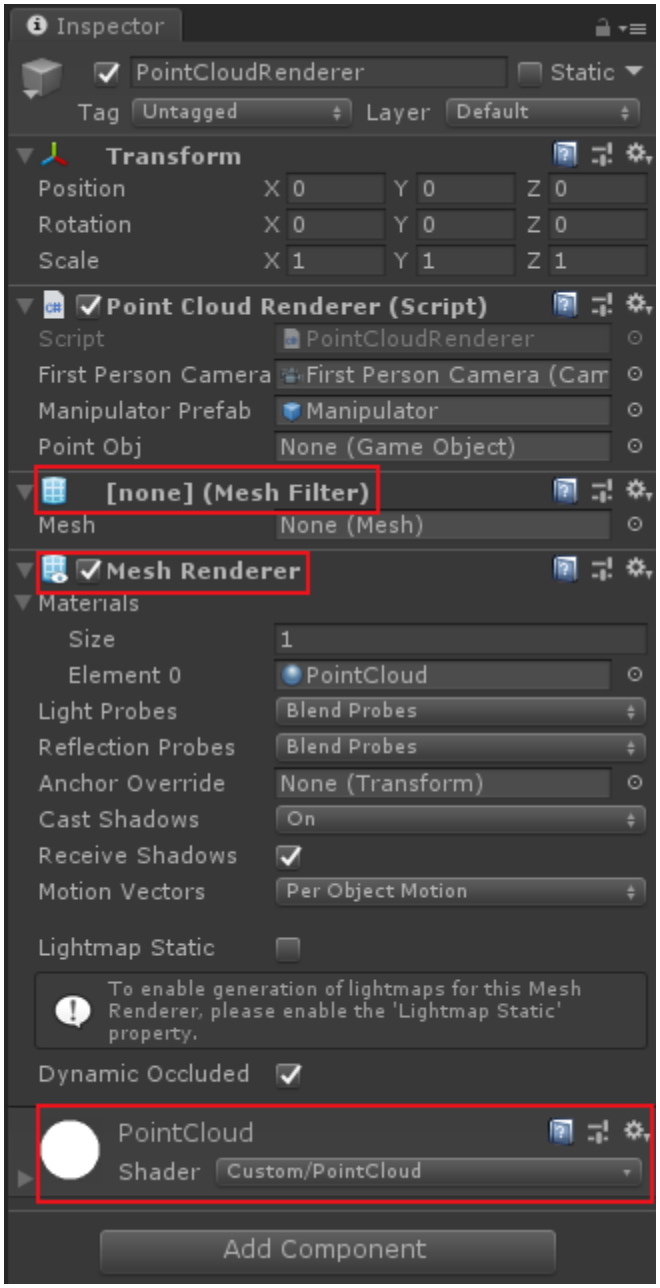
- In order to put 3D virtual objects on 2D plane, ARCore performs a raycast against detected planes.
- In this case, the direction of the raycast is determined by the hit position on the screen and the camera position.
- The point cloud models will be put into the scene based on this hit pose.

Implementation – Point Cloud Rendering

There are few things to do when rendering the point cloud models in the scene:

- Implement the selective query.
- Store the information of the selected points as Mesh.
  - The Mesh class can handle large number of points.
  - The Mesh class has some useful properties and functions, which can assist mesh generation.

```
mesh.vertices = points;
mesh.colors = colors;
mesh.SetIndices(indices, MeshTopology.Points, 0);
```



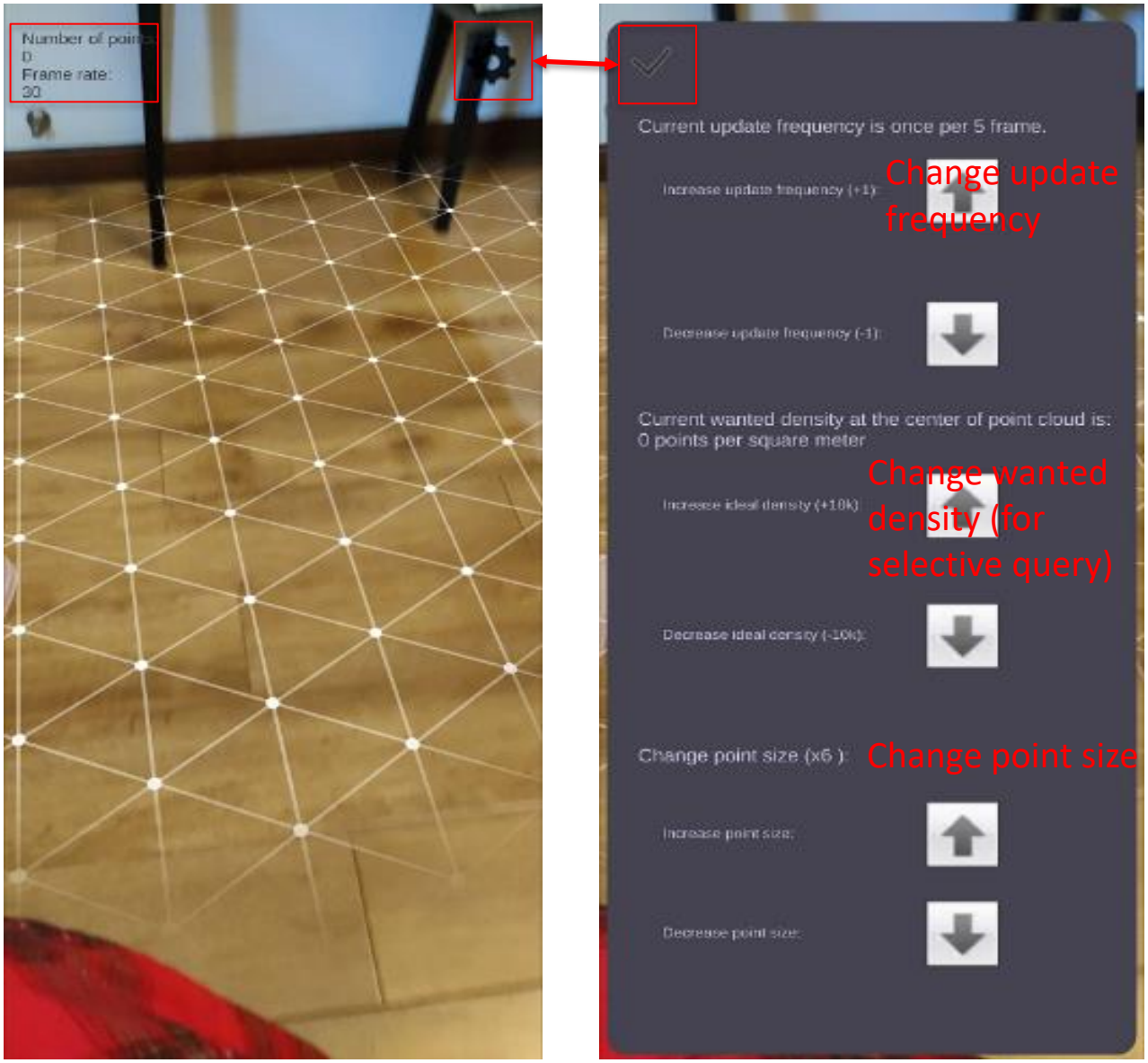
Implementation – Point Cloud Rendering

- Create a new *GameObject* and bind the Mesh to the *GameObject*.
- Create anchors based on the hit pose.
- Deliver the vertex positions, colours, indices to the GPU, and calculate the point sizes in the GPU.
- The point cloud model in the scene will be updated every X frames accords to new cLoD based selection result. X is a parameter called UpdateFrequency.

Implementation – Object Manipulation

- In order to scale and rotate the point cloud models, ARCore's object manipulation system is loaded.
- There are some changes in the pre-compiled scripts:
  - The script of Selection Manipulator is revised to avoid putting a new object into the scene when the user hits on an existing object.
  - The script of Scaling Manipulator is revised so that the number of points to be rendered will change while scaling.

Implementation – User Interfaces



Overview

1. Introduction
2. Methodology
3. Implementation
4. Results
5. Analysis
6. Comparison
7. Conclusion

Results – Source Code

- Source code and APK are available at: https://github.com/LiyaoZhang0702/AR_PointCloud (Only supports Android devices at current stage)

Results - Visualization

- Visualizing small point cloud models like furniture

Results - Visualization

- Visualizing large point cloud models (scanned point cloud of an Office with 1.5M points)

Results - Visualization

- Visualizing large point cloud models (Terrain point cloud with 4.7M points)

Results - UI

- Use different parameters to visualize point clouds
  - Different update frequency
  - Different wanted density
  - Different point size

Overview

1. Introduction
2. Methodology
3. Implementation
4. Results
5. Analysis
6. Comparison
7. Conclusion

Analysis

- In order to evaluate the capability of the rendering system, some additional experiments are carried using different types of test datasets.
- The test datasets can be mainly divided into three categories: furniture point clouds, architecture point clouds, and terrain point clouds.
- To assess the results under the same criterion, all of the results are of the following condition:
  - The distance from the centre of the point cloud model to the camera is around 1 meter
  - The spatial domain of the point cloud model is around 1 square meter
  - The rendering results are considered to have nice visual quality from the tester's perspective.

Analysis – Parameters

Wanted Density

- There is no clear relationship between the properties of the point clouds and the value of wanted density.

Model	Points	Original Density(pts/m^2)	Wanted Density(pts/m^2)
Table	10722	10442.77	14226.93
Chair	11525	11257.52	14223.77
Sofa	53055	51592.20	27548.31
Office1	1498092	1451341.0	171499.9
Garage	1002399	975602.6	146434.8
Office2	8893706	8515189.3	241319.1
Terrain1	2474522	2387759.1	166677.3
Terrain2	4436593	4436593.0	179822.5
Terrain3	3711573	3711573.0	197976.2

- The selection of the wanted density is affected by the original density, the distribution of the point cloud, and the spatial domain of the rendering result. So, it's quite challenging to find a proper value of wanted density for all datasets.
- Although there is a recommended value of the wanted density that is from 100,000 points/ m^2 to 200,000 points/ m^2. Manual adjustment is still needed, especially when visualizing sparse point clouds or unevenly distributed point clouds.

Analysis – Parameters

Update Frequency

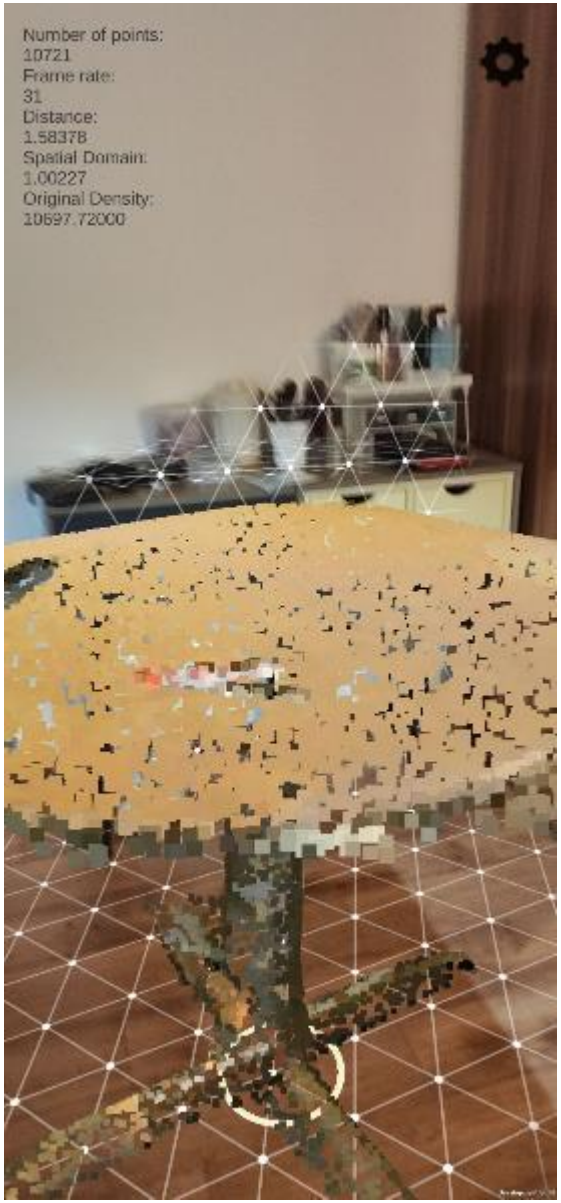
- When visualizing all the point clouds that are under the capability of the rendering system, the update can be implemented more frequently than once per 5 frames.
- The pauses between each update are not noticeable.

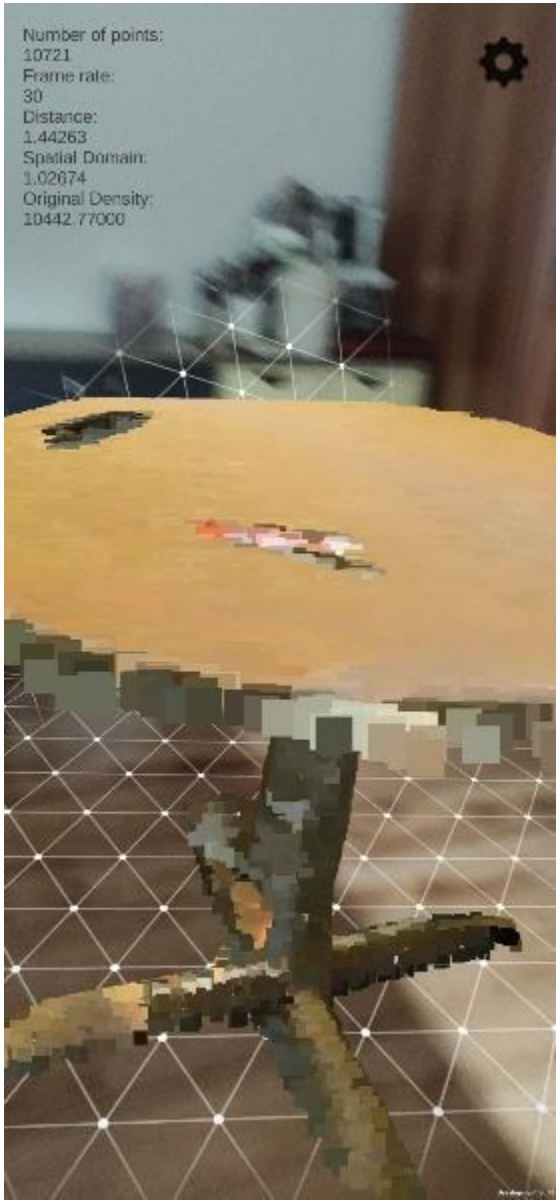
Model	Points	Update Frequency	Point Size
Table	10722	once per 1 frame	x12
Chair	11525	once per 1 frame	x14
Sofa	53055	once per 1 frame	x5
Office1	1498092	once per 2 frame	x5
Garage	1002399	once per 2 frame	x5
Office2	8893706	once per 5 frame	x5
Terrain1	2474522	once per 2 frame	x5
Terrain2	4436593	once per 3 frame	x5
Terrain3	3711573	once per 3 frame	x5

Point Size

- The ideal point size (x5) performs well when visualizing the dense point clouds.
- When visualizing the sparse furniture point clouds, the adjustment of point size is required.

Analysis – Parameters





Analysis – Performance

Boundary

- The rendering system can process at most 10 million points in memory / CPU, and contain at most 5 million points in the scene by GPU.

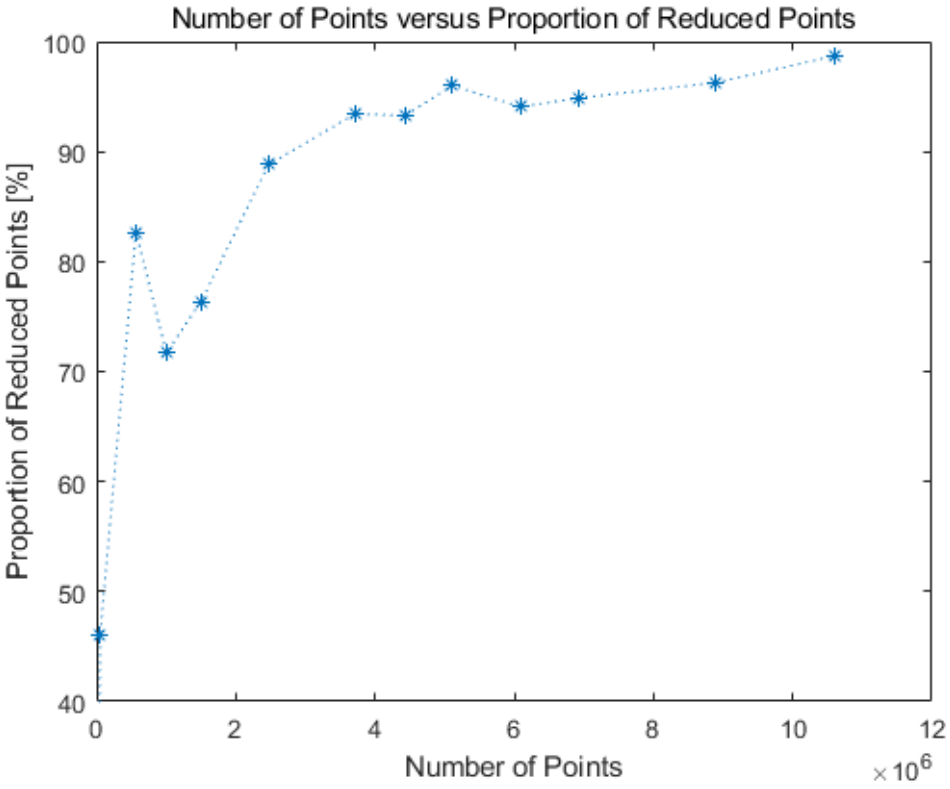
Frame Rate

- After experiments, we find that no matter how many points are processed and visualized by the rendering system, the frame rate stays at 30 fps stably.

Analysis – Performance

Proportion of Reduced Points

- When visualizing large point clouds in small area ($1\ m^2$), the proportion of reduced points is high, which is from 70% to 90%.
- The proportion of reduced points is determined by the spatial domain, distribution, and the original density of the point cloud.

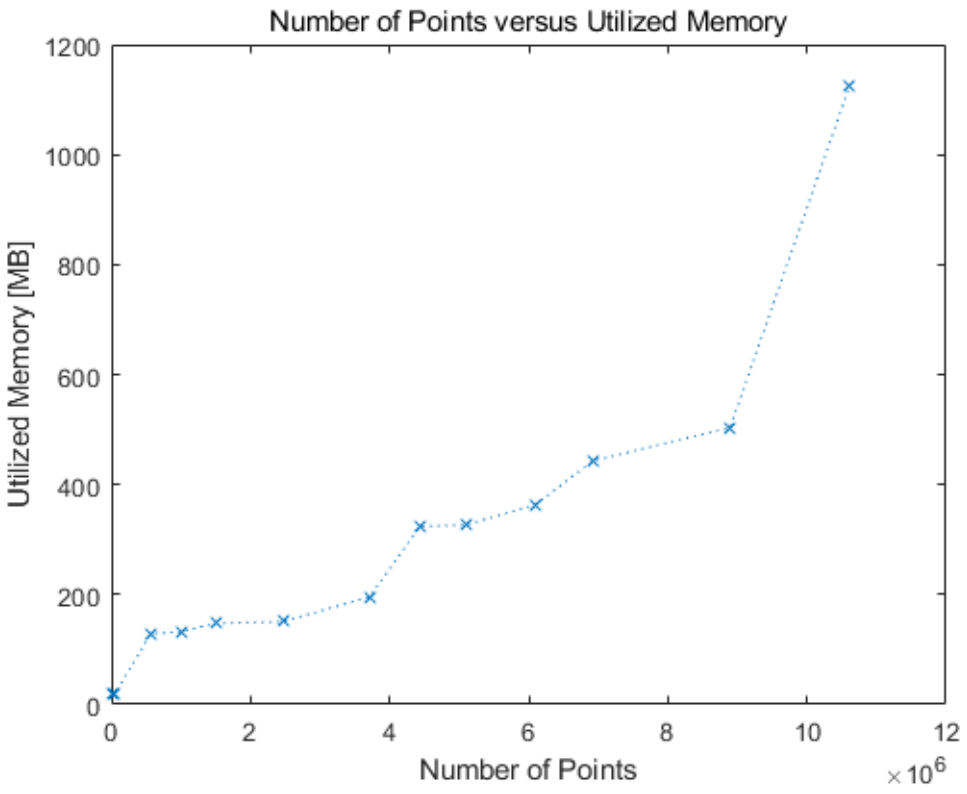


- For most of the indoor applications that usually have small field of vision, the number of points can be reduced significantly when visualizing large point clouds.

Analysis – Performance

Utilized Memory

- The utilized memory increases when the number of points to be processed increases.
- When visualizing large point clouds, the consumption of memory is quite high.
- The consumption of memory is too high for a simple point cloud renderer.

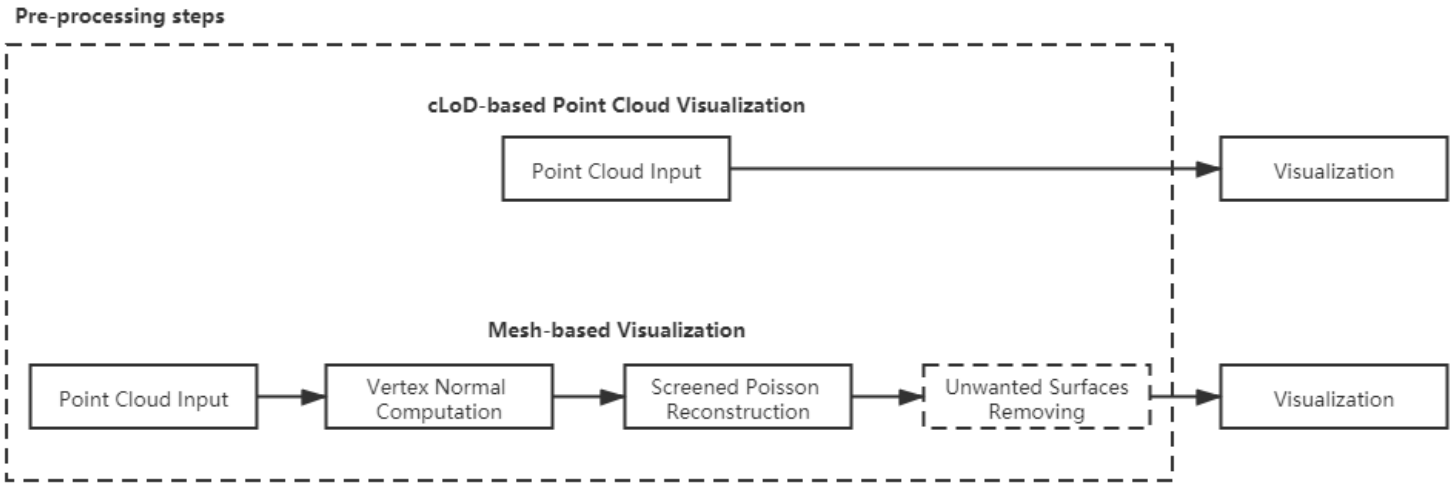


Overview

1. Introduction
2. Methodology
3. Implementation
4. Results
5. Analysis
6. Comparison
7. Conclusion

Comparison - Preprocessing

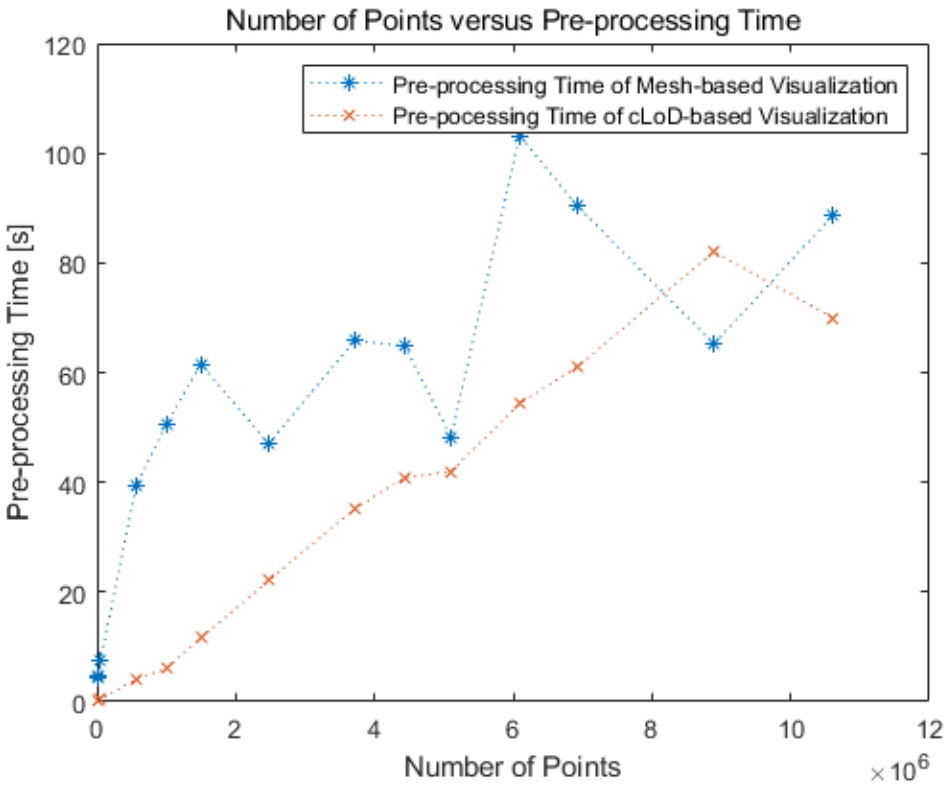
Pre-processing Workflow



Comparison - Preprocessing

Pre-processing Time

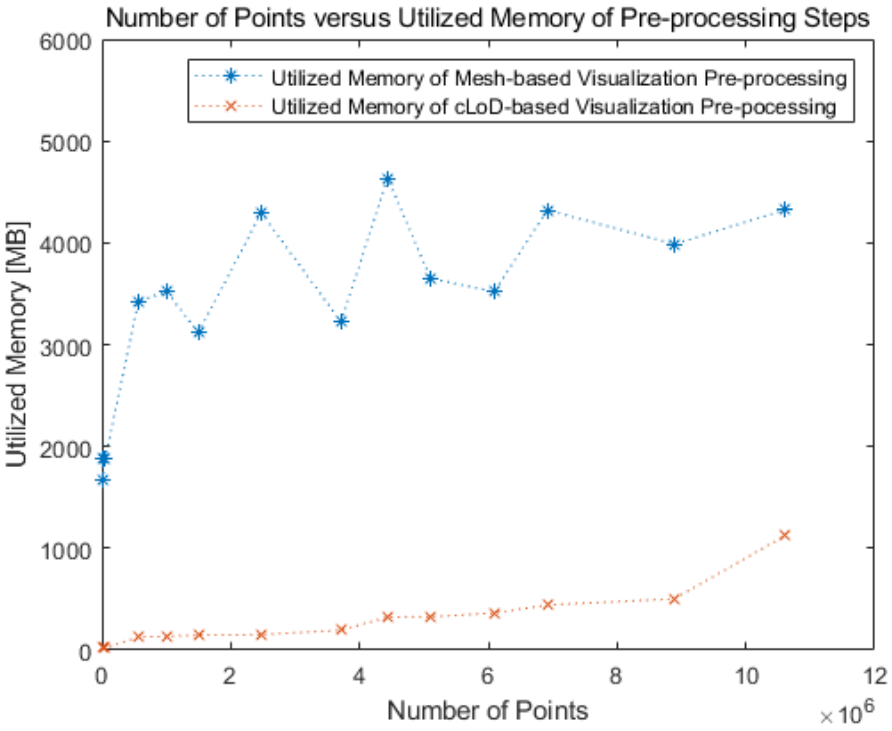
- The pre-processing time of the mesh-based visualization is the sum of the time to import point cloud, time to calculate the normals, and the time to implement the Screened Poisson algorithm.
- Although the pre-processing steps of the mesh-based visualization are implemented on the computer, for most cases the pre-processing time of our cLoD-based visualization is still less than the pre-processing time of mesh-based visualization.



Comparison - Preprocessing

Utilized Memory of Pre-processing Steps

- The utilized memory of our cLoD-based visualization pre-processing is from 0 to 1200 MB, which is much less than the utilized memory of mesh-based visualization pre-processing (from 1600 to 4600 MB).



Comparison - Visualization

Visual Quality

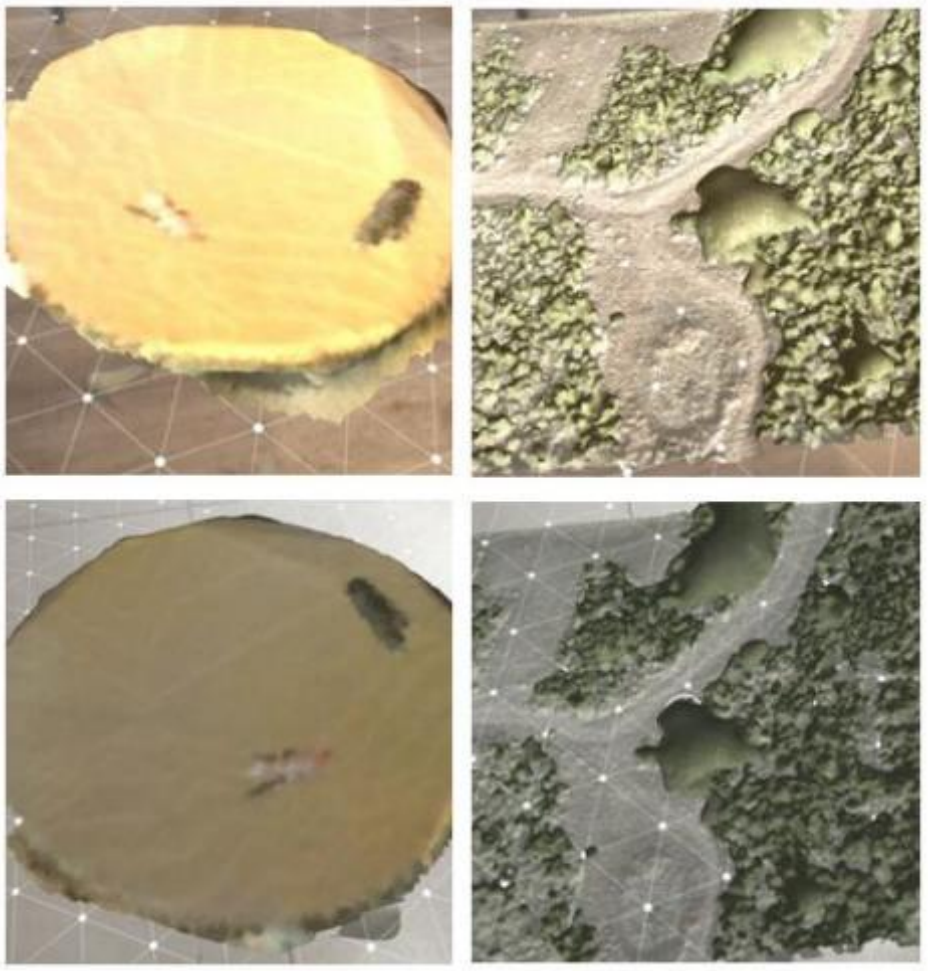
- The overall visual quality of our cLoD-based point cloud visualization and the mesh-based visualization is quite close.

Point clouds (top) and mesh models (bottom)

Comparison - Visualization

Visual Quality

- The major drawback of our cLoD-based point cloud visualization is that the point cloud models don't have complete geometry and topology.
- Some complex behaviours like shading and adding shadows are not feasible for point cloud models, which can performs well on the mesh models.

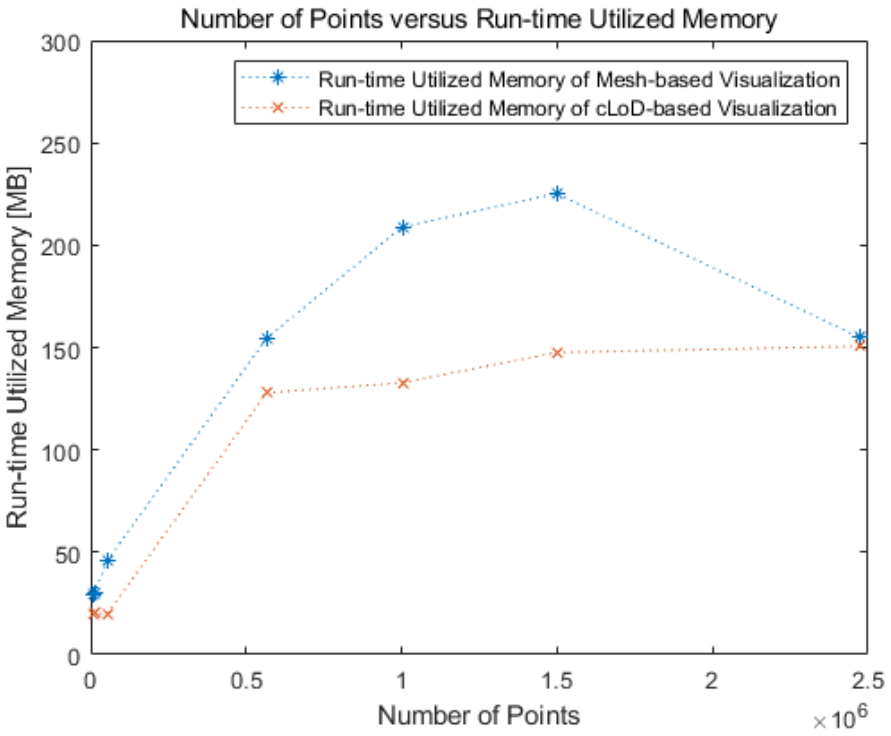


(Different rendering results of the same mesh model due to different environment)

Comparison - Visualization

Runtime Utilized Memory

- Mesh models contains more information, such as uv texture coordinates, texture, and normal.
- The run-time utilized memory of mesh-based visualization is always more than that of our cLoD-based point cloud visualization.



Overview

1. Introduction
2. Methodology
3. Implementation
4. Results
5. Analysis
6. Comparison
7. Conclusion

Conclusion - Contributions

- With this method, the rendering system can handle point cloud models with at most 10M points, contain 5M points in the scene and visualize them at 30 fps. *Visualizing such large point clouds is not possible without the cLoD method*, and we can even visualize multiple large point cloud models only if there are less than 5M points in the scene after selection.

- We add the concept of ideal density to the cLoD method and choose to use a uniform threshold of cLoD to filter the points, which makes the improved cLoD method fit the mobile AR environment.

- Besides mobile AR applications, the improved cLoD method can also be used in other applications that need to render point clouds on mobile phones.

Conclusion - Contributions

- In the rendering system, basic operations of an mobile AR application, interactions, and friendly user interfaces are realized.

- The rendering system has pretty nice usability. No matter the point clouds are unevenly distributed or are very sparse, the users can always find proper value of parameters to improve the rendering results in the end.

- Compared to the mesh-based visualization, our cLoD-based point cloud visualization doesn’t need much pre-processing steps, once the file is loaded into the system, the point cloud models can be visualized without delay.

- The final visual quality is close to the mesh-based visualization as well.

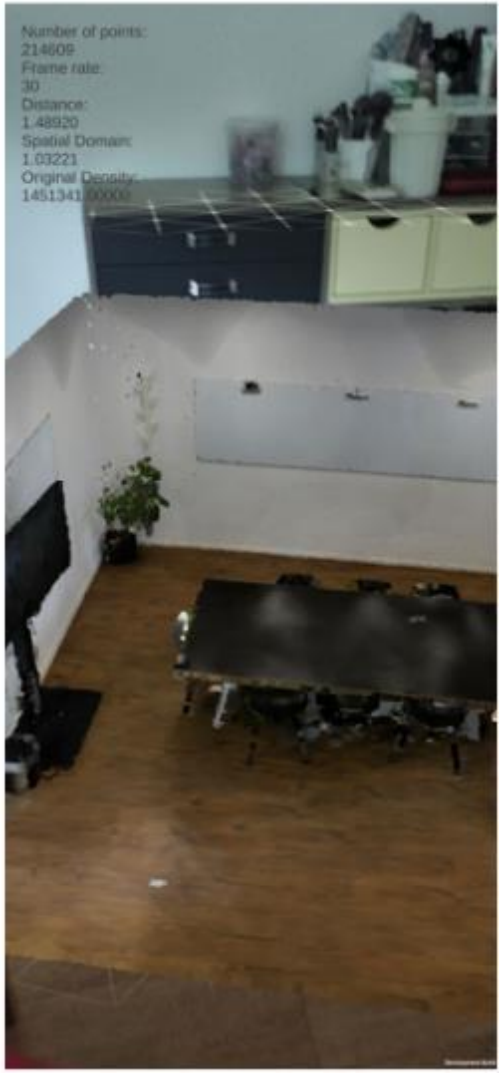
Conclusion - Limitations

- The quality of the automatic estimation needs to be improved. Manual operations are sometimes required to find the proper value of parameters.
- The usability of the rendering system in the outdoor environment is under question.
- Due to the lack of valid geometry and topology, shading models can not be implemented and shadows can not be generated for the point cloud models.
- The memory bandwidth is overused. The utilized memory is too much for a simple mobile point cloud renderer.
- The phone will be overheating when running the rendering system for more than 20 minutes.

Conclusion - Applications

- Strength of this method:
  - Can directly get use of the easily obtained point clouds
  - The materials and shaders used can be easily changed

- Potential Applications:
  - Outdoor: architecture, industrial design
  - Indoor: home renovation, estate sales, architectural design
  - …



Conclusion – Future Work

- Improve the quality of automatic estimation.
- Upgrade the formula of selective query, using distance from the camera to individual points and nD data structure to improve the results.
- Optimize the app, reduce the utilized memory and solve the overheating problem.
- Test our method with more datasets and different devices to see its applicability.
- Explore the potential of our method to visualize larger point clouds like city or nation wide point clouds.
- Apply more interactions to the rendering system.

References

- Azuma, R., Baillot, Y., Behringer, R., Feiner, S., Julier, S., and MacIntyre, B. (2001). Recent advances in augmented reality. IEEE computer graphics and applications, 21(6):34–47.
- Schutz, M., Krosl, K., & Wimmer, M. (2019). Real-Time Continuous Level of Detail Rendering of Point Clouds. 2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR). doi: 10.1109/vr.2019.8798284van Oosterom, P., 2019. From discrete to continuous levels of detail for managing nD-PointClouds. Keynote presentation at the ISPRS Geospatial Week 13 June 2019, Enschede, The Netherlands.
- Virtanen, J.-P., Daniel, S., Turppa, T., Zhu, L., Julin, A., Hyyppa, H., Hyyppa, J., 2020. Interactive dense point clouds in a game engine. ISPRS Journal of Photogrammetry and Remote Sensing, 163, 375–389.
- Xuefeng, G., 2019. 三维点云可视化系统技术报告.

Thank you, any questions?