MR Environments Constructed for a Large Indoor Physical Space

Huan Xing[1,5], Chenglei Yang[1(✉)], Xiyu Bao[1], Sheng Li[2], Wei Gai[1], Meng Qi[3], Juan Liu[1], Yuliang Shi[1], Gerard De Melo[4], Fan Zhang[1], and Xiangxu Meng[1]

[1] School of Software, Shandong University, Jinan, Shandong, China
chl_yang@sdu.edu.cn
[2] Department of Computer Science and Technology, Peking University, Beijing, China
[3] School of Information Science and Engineering, Shandong Normal University, Jinan, Shandong, China
[4] Rutgers University, Piscataway, NJ, USA
[5] Taiyuan University of Technology, Taiyuan, Shanxi, China

Abstract. To resolve the problem that existing mixed reality (MR) apparatus are unable to scan and model a large and complex indoor scene at once, we present a powerful toolkit for constructing MR environment easily. Our toolkit establishes and maintains accurate mapping between the virtual and physical space, and sets the occlusion relationships of the walls. Additionally, we design spatial anchor deployment strategy supporting deviation correction between the real and virtual spaces, so that the spatial anchors can maintain a virtual object's location and orientation in the real world. Our experiments and applications show that the toolkit is convenient for constructing MR apps targeting large physical spaces in which users can roam in real time.

Keywords: Mixed reality · Reconstruction · Real-time roaming · Indoor scene

1 Introduction

Popular mixed reality (MR) head mounted display (HMD) enable users to interact with three-dimensional holograms blended with the real world. It is widely used in education, games, surgery and museums [4]. In MR museum deployments, visitors wearing HoloLens can roam in every corridor and room to see different virtual collection items embedded into the real space. When a visitor resides in one room, one cannot see through the physical walls. Evidently, the virtual scene should be rendered in real-time. Given these desiderata, how to

Supported by the National Key Research and Development Program of China under Grant 2018YFC0831003, the National Natural Science Foundation of China under Grant 61972233 and the National Natural Science Foundation of China under Grant 61902225.

N. Magnenat-Thalmann et al. (Eds.): CGI 2020, LNCS 12221, pp. 132–144, 2020.
https://doi.org/10.1007/978-3-030-61864-3_12

construct the virtual scenes of a large physical space is of principal concern. At present, the virtual models in most MR apps are designed by professionals, making use of specialized tools such as 3D Studio Max, and then deployed on the corresponding MR hardware by software engineers [10]. Additionally, to create a convincing MR experience, end-users need to spatially perceive the real environment. For example, HoloLens provides a spatial mapping module. Prior to using this module, one needs to complete a scan of all parts of the environment that have been observed [10]. For small-scale experience applications, this process is feasible and accurate. However, since the scan distance of a HoloLens is 0.8–3.1 m, it is time-consuming and laborious for a large-scale interior space with complex room structure. Additionally, for a large-scale interior space, as the virtual design content is enriched, the system performance will decline when roaming due to the limited resources provided by the HoloLens [12].

In order to provide users an immersive and real-time MR roaming experience, this paper proposes a toolkit that aids in conveniently and easily constructing MR apps that can provide a real-time experience in large indoor physical spaces via HoloLens. In the preliminary work [15], We have introduced our idea and toolkit briefly. Based on it, an indepth study is made. Overall, our main contributions include:

1) We present a toolkit for conveniently and easily constructing MR apps in large indoor physical spaces. The toolkit establishes and maintains accurate mapping relationships between the virtual and physical spaces.
2) We develop a method to deal with occlusion with respect to walls, and such occlusion information serves as a factor in selecting appropriate virtual models and rendering the virtual objects occluded by translucent walls.
3) We propose a custom data structure called VorPa, based on the Voronoi diagram (VD), to effectively implement path editing, accelerate rendering and collision detection, and correct deviation between the real and virtual spaces.
4) We propose spatial anchor deployment strategy supporting deviation correction between real and virtual spaces when roaming.

2 Related Work

2.1 VR/AR/MR Apps Design

Recently, more and more researchers focus on VR, AR and MR app design. Gai et al. presented a new genre and designed a toolkit for supporting users in easily constructing a VR maze and playing it [6]. However, the physical space is limited by the Kinect's tracking field and the genre is not targeting AR/MR. Hsiao et al. proposed a system in which users can manipulate furniture cards directly in the model house task space, get instantaneous multi-view 3D visual feedback, and re-adjust cards until they are satisfied [7]. Nevertheless, all manipulations of virtual object need additional tools like cards. Wei et al. introduced a conceptual design framework that integrated MR with a multi-touch tabletop interface,

which provided an intuitive and efficient interface [14]. Unfortunately, only a conceptual framework was introduced and the experience of MR scenes was not immersive.

2.2 Accelerate Rendering in AR/MR

Acceleration of rendering can improve mobile device's performance. Traditional methods include occlusion culling [3], potentially visible sets (PVS) [9], levels of detail (LOD) [5] etc. Some focus on how to utilize user's visual processing characteristics to accelerate rendering. Kim et al. proposed an accelerated rendering strategy for head-mounted augmented reality display devices and pointed out that during the rotation of the headset, the user's attention to the model in the field of vision will be reduced [8]. Therefore, the rendering performance can be improved by reducing the model quality in user's visual field when the users moves.

Other studies focused on how to optimize the rendering algorithm itself. A set of acceleration strategies such as parallelization of rendering algorithm are invoked to optimize the HoloLens's rendering performance [11]. To avoid rendering delays, the MR 3D GIS deploys HoloLens holographic remote player middleware on a PC to quickly render large virtual scenes, and sends the results to the HoloLens [13]. Despite transferring all calculations from the HoloLens to a higher performance PC, this method could lead to new network latencies.

3 System Design and Implementation

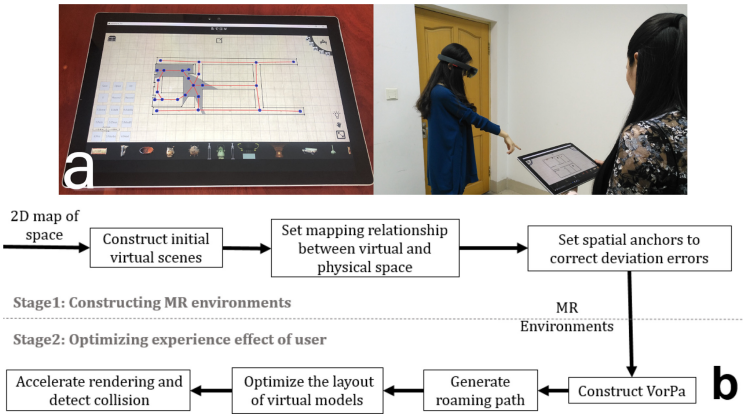


Fig. 1. (a) Overview of our toolkit; (b) Working pipeline of our toolkit.

We developed a toolkit for constructing MR apps with real-time roaming in large indoor physical spaces (see Fig. 1a). The toolkit consists of two parts: one is a 2D interface supporting users designing virtual scenes using multi-touch technology, such that a 2D map of the space can be drawn and the virtual

objects can be manipulated easily; the other one is MR experience part, on which the corresponding 3D scenes are automatically generated and displayed on MR devices, like HoloLens.

The overview of methods used in our toolkit is shown in Fig. 1b. Our toolkit has two stages: Stage1: Constructing MR environments. For an indoor physical space, we use 2D map of the space to construct initial virtual scenes and set mapping relationship between virtual and physical space. We set some spatial anchors to correct deviation errors appearing in the process of user roaming; Stage2: Optimizing experience effect of user. Based on a data structure VorPa, we provide some methods to improve user roaming experience and system performance, such as generating roaming path, optimizing virtual model layouts, accelerating rendering and detecting collision.

3.1 Constructing MR Environments

To construct a MR environment for an interior space, especially for a large-scale space, our method includes the following steps:

1. Constructing the initial virtual scenes. We only consider the room structure, mainly walls. The arrangement of virtual objects will be introduced in Sect. 3.2. Using our toolkit, we can obtain a digital 2D map of the real space, which will be displayed on the 2D interface.
2. Setting the mapping between virtual and physical spaces. We set corresponding landmarks in virtual and real spaces, respectively. Through these landmarks, we can calculate the coordinate transformation formula. So for any point in virtual space, we can get its corresponding position in real space. Then, we can construct 3D scene used in HoloLens according to 2D map. The real space is three-dimensional and HoloLens can provide 3D coordinate like (x, y, z). Here, y-dimension can be neglected because the height of wall's location will be set according to ground automatically.
3. Setting transparency of walls. We set transparency value to handle the occlusion relationship of walls with different transparency. For opaque wall in the real space, we set transparency value as 1 and construct a corresponding virtual wall, to which a transparent texture is applied; For transparent or semi-transparent walls such as glass walls, we set the transparency value between 0 and 1. This value will serve as a parameter when rendering the virtual objects.
4. Setting spatial anchors to correct deviation errors. Sometimes, HoloLens is unable to recognize the environment and position of the user so that tracking loss occurs and there is deviation error between virtual models and the real space. We will describe how to solve this problem in Sect. 3.4.

3.2 Optimize the Layouts of MR Environments

As mentioned in Sect. 1, for a large-scale interior space, as the design content is enriched, the system performance will decline while roaming because of the

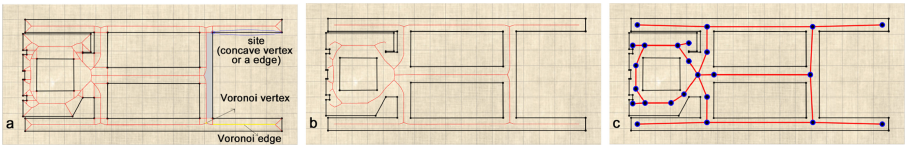


Fig. 2. (a) Voronoi diagram V in polygon P; (b) Initial roaming path R in polygon P; (c) Path editing.

limited computing resources afforded by the HoloLens. Generally, users are easily lost when roaming in complex large scenes. For the designer, predesignating a reasonable roaming path on the design side can effectively help users in better navigating unfamiliar scenes. Therefore, in this section, we propose a custom data structure called VorPa to effectively implement path editing, accelerate rendering, and provide for collision detection.

VorPa is designed based on the Voronoi Diagram (VD) of polygons. The VD is an important geometric structure in computational geometry. A VD records the regions in the proximity of a set of points. We refer to these points as sites. Unlike for a set of points, the regions in the proximity of a set of objects, such as line segments, circular arcs and polygons, also form a VD. VD of a polygon can subdivide the polygon into cells called Voronoi regions based on proximity characteristics [1] (see the grey region in Fig. 2a). In the VD of polygon P, the concave vertex and edge of a polygon are the sites; the common boundary of two adjacent regions is called a Voronoi edge; the points where Voronoi edges meet are called Voronoi vertex. A Voronoi skeleton path is composed of a Voronoi vertex and Voronoi edge.

For a line segment L in polygon P, a point s is weakly visible to L if L has at least one point that is visible to s. The weak visibility polygon (see grey part in Fig. 3a) of L is the set of all points in P that are weakly visible to L [2].

We define the structure of VorPa based on a VD as follows:

Class VorPa {
  List<Edge> P; //2D design wall list
  List<Path> V; //Voronoi edge in polygon
  List<Path> R; //Roaming path in polygon }
Class Path {
  Edge *edge*; //Current Voronoi edge
  List<Model> *modellist*; //Virtual models bound to path
  List<Edge> Vis; //Weak visibility region of path
  Bool *direction*; //Direction of edge, route guide for users }

We show the pseudocode of constructing VorPa. The time complexity of constructing VorPa is $O(E)$. The E represents the number of edges in polygon.

Algorithm: Constructing VorPa
Input: 2D Polygon P, Model list M
Output: VorPa Vor
Init VorPa Vor (as shown in Class VorPa)

Copy P to $Vor.P$, Compute the VD V of $Vor.P$ and Copy V to $Vor.V$
For each v_i in $Vor.V$
If exist p_i in $Vor.P$, and $p_i.Startpoint$ equals $v_i.edge.Startpoint$ or $v_i.edge.Endpoint$
Continue;
Else
Add v_i to list $Vor.R$
For each r_i in $Vor.R$
Compute weak visibility polygon W of $r_i.edge$ to Polygon $Vor.P$ and Copy W to $r_i.Vis$

Generate Roaming Path. Based on VorPa, we provide a method to automatically generate a roaming path for user, which also can be modified interactively by the designer later.

As shown in the Constructing VorPa Algorithm, we obtain initial roaming path R (see Fig. 2b) by deleting all edges that contain vertices of the polygon in V. Path editing is an interactive operation, and the designer can modify the nodes in the path to optimize the roaming path R, as shown in Fig. 2c. This is an undirected connected graph G, which can be modified, such as by adding vertices, moving vertices, or merging vertices. Designers can also plan a recommended tour path for the users on the experience side by designating a direction. Once designers specify a direction for a path, the presentation of the route guide will be available in the corresponding path in the experience part.

Optimize the Layout of Virtual Models. Given that the system targets general users, VorPa also provides recommendations for the model layout. Designers can click on each sub-path R_i in the roaming path R (see the blue segment in Fig. 3a), and VorPa will calculate the weak visibility region $Visi$ of R_i in polygon P in advance, and then bind it to R_i. This is the grey part in Fig. 3a. Arranging models in $Visi$ helps avoid unreasonable layouts, such as dead corners or large variations in density in different fields of view.

VorPa also pre-processes model data during the layout phase. When the designer places a model, every model is bound to all sub-paths that contain this model in the weak visibility region and sites that contain this model in the Voronoi region. Thus, at the end of the design, each sub-path has multiple models that are within the weak visibility region of the sub-path. Each site also has multiple models, which are located in the Voronoi region corresponding to that site. This pre-processing operation will be used to accelerate rendering and collision detection in the next section.

3.3 Accelerate Rendering Based on the Optimized Deployment

As mentioned in Sect. 1, due to the complexity of the scene and the limited performance of some hardware in AR/MR devices, the over-calculation for rendering or positioning while roaming may lead to latency or even crashes. However, the

occlusion culling function in Unity3D cannot solve latency problems due to the high complexity of models in view. Traditional LOD needs extra real-time computing power, which is not desirable for the HoloLens with its limited hardware performance. In addition, we also consider the positioning problem. Traditional method for interaction with virtual objects is through adding collision detection to each object, which not only requires substantial calculations for large-scale scenes, but also has detrimental effects on the user experience. To solve these problems, we introduce some additional strategies based on VorPa.

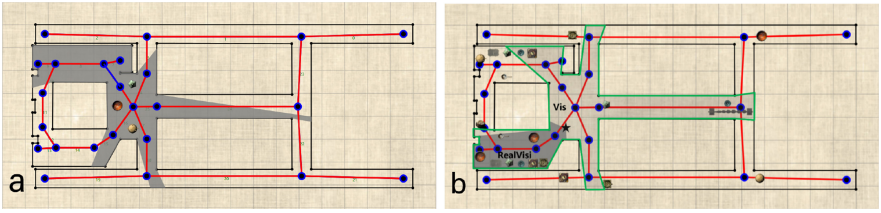


Fig. 3. (a) Laying out models in weak visibility region; (b) The visible region of the current view point. (Color figure online)

Accelerate Rendering. VorPa can locate the user with regard to the appropriate sub-path R_i with the help of the user's location received from the HoloLens and obtain the model list M_i bound to that sub-path. The visible regions of the FOV $RealVisi$ are calculated in real time [16], as shown in Fig. 3b.

First, the HoloLens stops rendering all models that do not belong to M_i. Then, for models in M_i, the HoloLens as well stops rendering the models in $Visi$ but not in $RealVisi$. Finally, we calculate the model complexity F that is in M_i and $RealVisi$. In our tests, we use triangular facets to measure the model complexity. Through experiments, we obtain rough estimates of the upper limit threshold of the recommended rendering effect. According to the distance between the model and the user's location, the accuracy of the model is tuned from far to near so as to guarantee $F < threshold$ and that the average frame rate is at least 15.

For transparent or semi-transparent walls (e.g. glass wall), special handling is introduced. As mentioned in Sect. 3.1, we set a transparency value between 0 and 1 as a parameter to select different levels of detail of the models. When users see through transparent or semi-transparent walls, we reduce the complexity of the models behind the walls according to the distance between models and users' locations and the transparency value of the wall. If the transparency value is 1, the real wall is opaque. For this case, a virtual wall model is needed. The system renders a transparent texture applied to the corresponding virtual wall, so that the virtual objects occluded by the real walls are invisible to users.

Detect Collision. As mentioned above, VorPa can locate users with respect to the corresponding sub-path R_i, which is a Voronoi edge in the VD of the polygon P. Additionally, we bind the model to the site corresponding to its Voronoi region. VorPa only needs to traverse the Voronoi region near R_i within $O(n)$ time complexity and obtain all model lists M in each region within $O(1)$ time. When a user roams in the R_i, we only need to set collision detection on M, which only takes up a small part of the models in the whole scene in real time, thus effectively reducing the calculation cost of location.

3.4 Correct Deviation Using Spatial Anchors

Our approach uses the characteristics of spatial anchors to correct the deviations between the virtual and real space while a user roams around. HoloLens can trace spatial anchors, update their location and coordinates to ensure that the spatial anchor can always mark one point in the real world. Therefore, we place some anchors in the scene and bind the virtual model to the nearest anchor coordinate system. The virtual model constantly updates its position with the anchor. According to the characteristics of spatial anchors, the accurate positioning range of each anchor is 3 m. Moreover, the spatial anchor should be visible at the position of the virtual model. To obtain high efficiency, we should minimize the number of spatial anchors during deployment, and each spatial anchor should cover as many virtual models as possible.

Our spatial anchor layout should meet the following conditions: 1) All virtual models in the scene can be covered by spatial anchors, including virtual wall models and virtual object models placed in the scene; 2) Each spatial anchor is responsible for virtual models within a 3-m radius around the spatial anchor; 3) A spatial anchor is visible at the position of the anchored virtual model; 4) A minimum number of spatial anchors.

Therefore, the input to the spatial anchor deployment algorithm is: the virtual wall set in scene $W = w_1, w_2, w_3, \ldots$, a collection of virtual objects placed in scene $R = r_1, r_2, r_3, \ldots$; the output is: the spatial anchor layout $S = s_1, s_2, s_3, \ldots, s_q$ and the spatial anchor number q.

The wall model is regarded as a special virtual object whose position is the midpoint of the wall. After combining the virtual wall and virtual object sets, we use M to represent the set of all virtual models that need to be covered by spatial anchors in the scene is: $M = W \bigcup R = \{m_1, m_2, m_3, \ldots, m_n\}$

Before using our algorithm, this paper has a pre-processing step based on the data structure VorPa:

1. The value v_i $(i \in n)$ is set for each virtual model m_i. Each v_i represents the probability that the virtual model m_i will be seen in the scene. The calculation method is as follows:

$$v_i = \frac{\sum_{path\ in\ VR(m_i)} Length(path)}{\sum Length(p_j)}, \tag{1}$$

where $VR(m_i)$ represents the visible area of m_i in the scene, and p_j represents the roaming sub-path j in the scene. The calculation of v_i is based on the roaming path length, that is, the ratio of the roaming path length falling into the visible area of the relic m_i and the total roaming path length of the scene.

2. Set the value v_{p_j} for each sub-path. In the data structure VorPa, if the virtual model m_i is in the weakly visible region of path p_j, the relic m_i is bound to the modelList of path p_j. The value of all models in the list of p_j models in modelList is the value of path p_j:

$$v_{p_j} = \sum_{m_i \ in \ modelList \ of \ p_j} v_i \tag{2}$$

We convert this problem into an optimization problem seeking to minimize the number of spatial anchors. Accordingly, we propose an algorithm based on integer linear programming.

1) Get the initial set of spatial anchors. Before constructing the mathematical model, the initial set of l spatial anchors that can cover all the virtual models in the scene is obtained through the following process:
Create a matrix C that contains the whole scene and is initialized with 0s. Find the intersection area between the circle with $m_i(x_i, y_i)$ as the center and $3m$ as the radius and visibility region $VR(m_i)$ of model m_i. Then add v_i to the value of the corresponding array position in intersection area. Traverse the matrix C, find all the spatial anchors that can cover the different virtual model combinations, and add them to the initial set of spatial anchors without repeating.
2) We build a mathematical model based on the previous problem description:

$$minZ = \sum_{i=1}^{l} s_i, \qquad \begin{cases} s_i = 0 \ or \ 1, \ i = 1, 2, \ldots, l \\ \sum_{i=1}^{l} \ F_{ij} s_i \geq 1, \ j = 1, 2, \ldots, n \end{cases}$$

where s_i represents the spatial anchor i, assuming that there are initially l spatial anchors that can cover all the virtual models in the scenario, F_{ij} indicates whether the spatial anchor i can cover virtual model j. If it can be covered, the value is 1; if not, the value is 0. We minimize the number of spatial anchors to be placed and satisfy two constraints: 1) the value of spatial anchor s_i is 0 or 1, where 0 means that the spatial anchor i is not placed, and 1 means that it is placed; 2) all models in the scenario can be covered, that is, for any model, the number of spatial anchors that can be covered by this model is greater than or equal to 1. 3) The integer linear programming method is used to solve the model. Specifically, IBM's integer programming solver CPLEX is used to solve the optimized spatial anchor set.

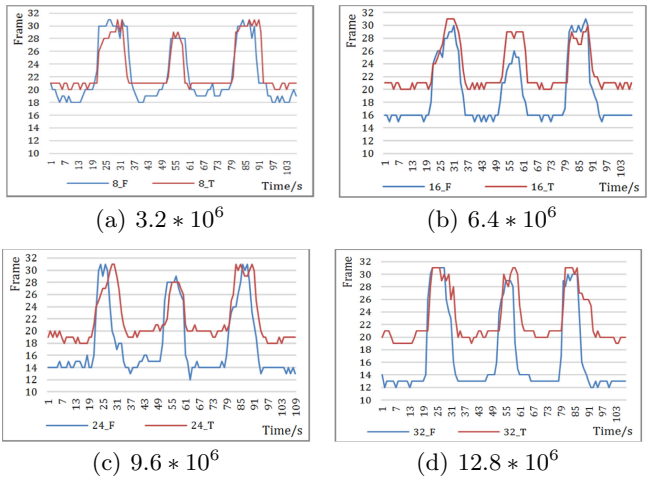
4 Experiments

Fig. 4. (a) Screenshot of 2D design interface; (b) A visitor's view corresponds to position marked with a red dot in (a); (c) Model layout and route design of experiment. (Color figure online)

To test our toolkit and methods, we designed a MR museum (see Fig. 4a and 4b) to conduct a series of experiments. For every exhibition hall in the MR museum, we design different themes, and visitors can choose different halls without interference from other halls. When a visitor is in one hall, the virtual exhibits in another hall will be occluded by the wall via a special texture. In addition, while visitors roam within the MR museum, the system can present the path on the ground as a tour guide while lowering their probability of getting lost. When visitors approach one exhibit, the system automatically plays the introductory audio of the respective exhibit, following the VorPa location algorithm to help visitors better understand the display content.

During roaming, we accelerate rendering based on the design of VorPa. To verify its effectiveness, we design experiments as follows: First, we test the threshold in advance and obtain the result that when the total number of triangles is no more than 10^5, the frame rate is sufficient for a fluid visitor experience; Then, we compare the performance of VorPa with regard to different capacities of scenes. In our experiment, we use the total number of triangles in a scene to quantify its capacity. We rely on frame rates to evaluate the user experience in terms of fluidity. High frame rates entail a low latency and better experience.

We select a room with appropriate lighting as experiment site (to ensure the stable tracking and position effect of HoloLens) and design the model layout and route for experimenter, as shown in Fig. 4c. The red line with arrow is the roaming path, and we place virtual models in region A, B, C, D near the path. Through changing the number of models in region A, B, C, D, we assess four kinds of scene capacities, namely $3.2 * 10^6$, $6.4 * 10^6$, $9.6 * 10^6$, and $12.8 * 10^6$. In each case, we experiment with two conditions, with and without our VorPa algorithm. In the course of the experiment, we request the participant to walk along the recommended path at a constant speed for a specified time and



(a) $3.2 * 10^6$ (b) $6.4 * 10^6$

(c) $9.6 * 10^6$ (d) $12.8 * 10^6$

Fig. 5. Results of four scenes capacities. $3.2 * 10^6$ (8_F/8_T denotes result obtained with/without using our algorithm, respectively), $6.4 * 10^6$, $9.6 * 10^6$, $12.8 * 10^6$.

to browse the models placed near the route. To ensure the same experimental conditions, all the experiments were completed by one experimenter in the same time. The system will record the number of frames per second.

We provide the experimental results under each scene capacity condition by means of line charts. By integrating the participant's route, we can explain the peaks and valleys in every line chart. When the participant walks past the areas that feature numerous models, the frame rate declines and reaches the valleys. In circumstances with fewer models, the frame rate is higher. As Fig. 5 shows, the difference in the frame rates between these two conditions (with and without our algorithm) varies in different phases. When the frame rate peaks, the difference is small. In contrast, when the frame rate becomes lower, the difference becomes more apparent. The difference of frame rates between these two conditions is smaller in Fig. 5a than in any other figures, which might stem from the fact that the number of triangles in a participant's view does not exceed the threshold most of the time.

We compare different experimental results in Fig. 6. Overall, the difference in the valley is larger with an increase of capacity. This means that our algorithm effectively improves the rendering effect and keeps it real-time. More specifically, the effect is more obvious as the capacity of the scene increases.

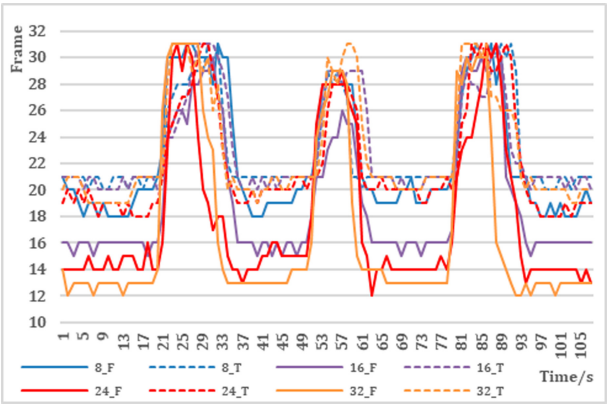


Fig. 6. Frame rate comparison. Solid lines denote results without our algorithm; while dotted lines represent results using our algorithm.

5 Conclusion

This paper presents a toolkit to aid general users in rapidly and conveniently designing MR applications, especially in large indoor spaces. Above all, the MR applications can provide a fluid and immersive roaming experience in real-time. To satisfy this requirement, we introduce VorPa, a data structure based on VD, which is beneficial to the user experience with regard to the strategies of path editing, accelerated rendering, collision detection, and correcting deviations. We apply this toolkit to an application and evaluate it in a set of experiments. Our evaluation shows that such a toolkit effectively improves real-time rendering efficiency of HoloLens.

In the future, we will design an algorithm to automatically calculate the upper limit of the number of rendering polygons for different types of CPU, memory, and other hardware state signals.

References

1. Aurenhammer, F.: Voronoi diagrams—a survey of a fundamental geometric data structure. ACM Comput. Surv. (CSUR) **23**(3), 345–405 (1991)
2. Chen, D.Z., Wang, H.: Weak visibility queries of line segments in simple polygons. Comput. Geom. **48**(6), 443–452 (2015)
3. Coorg, S., Teller, S.: Real-time occlusion culling for models with large occluders. In: Proceedings of the 1997 Symposium on Interactive 3D Graphics, p. 83-ff (1997)
4. Damala, A., Marchal, I., Houlier, P.: Merging augmented reality based features in mobile multimedia museum guides. In: Anticipating the Future of the Cultural Past, CIPA Conference 2007, Athens, Greece, 1–6 October 2007, pp. 259–264 (2007). https://halshs.archives-ouvertes.fr/halshs-00530903
5. Erikson, C.M.: Hierarchical levels of detail to accelerate the rendering of large static and dynamic polygonal environments. Ph.D. thesis. Citeseer (2000)

6. Gai, W., et al.: Supporting easy physical-to-virtual creation of mobile VR maze games: a new genre. In: Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, pp. 5016–5028 (2017)
7. Hsiao, F.J., Teng, C.J., Lin, C.W., Luo, A.C., Yang, J.C.: Dream home: a multiview stereoscopic interior design system. In: The Engineering Reality of Virtual Reality 2010, vol. 7525, p. 75250J. International Society for Optics and Photonics (2010)
8. Kim, J.B., Choi, J.H., Ahn, S.J., Park, C.M.: Low latency rendering in augmented reality based on head movement. In: International Conference in Central Europe on Computer Graphics (2016)
9. Laakso, M.: Potentially visible set (PVS). Helsinki University of Technology (2003)
10. Liu, Y., Stiles, N.R., Meister, M.: Augmented reality powers a cognitive prosthesis for the blind. bioRxiv, p. 321265 (2018)
11. Morley, C., Choudhry, O., Kelly, S., Phillips, J., Ahmed, F.: Mixed reality visualization of medical imaging data (2017)
12. Shearer, A., Guo, L., Liu, J., Satkowski, M., LiKamWa, R.: Characterizing bottlenecks towards a hybrid integration of holographic, mobile, and screen-based data visualization
13. Wang, W., Wu, X., Chen, G., Chen, Z.: Holo3DGIS: leveraging Microsoft Hololens in 3D geographic information. ISPRS Int. J. Geo-Inf. **7**(2), 60 (2018)
14. Wei, D., Zhou, S.Z., Xie, D.: MTMR: a conceptual interior design framework integrating mixed reality with the multi-touch tabletop interface. In: 2010 IEEE International Symposium on Mixed and Augmented Reality, pp. 279–280. IEEE (2010)
15. Xing, H., et al.: Rotbav: a toolkit for constructing mixed reality apps with real-time roaming in large indoor physical spaces, pp. 1245–1246 (2019)
16. Zhou, K., Pan, Z., Shi, J.: A real-time rendering algorithm based on hybrid multiple level-of-detail methods. J. Softw. **12**(1), 74–82 (2001)