applied sciences

MDPI

Article

Virtual Object Replacement Based on Real Environments: Potential Application in Augmented Reality Systems

Yu-Shan Chen [1] and Chi-Ying Lin [1,2,*]

[1] Department of Mechanical Engineering, National Taiwan University of Science and Technology, No.43, Keelung Rd., Sec.4, Taipei 106, Taiwan; yusan2048@gmail.com

[2] Center for Cyber-physical System Innovation, National Taiwan University of Science and Technology, Taipei 106, Taiwan

* Correspondence: chiying@mail.ntust.edu.tw; Tel.: +886-2-2737-6484

Received: 22 March 2019; Accepted: 28 April 2019; Published: 29 April 2019

Abstract: Augmented reality (AR) is an emerging technology that allows users to interact with simulated environments, including those emulating scenes in the real world. Most current AR technologies involve the placement of virtual objects within these scenes. However, difficulties in modeling real-world objects greatly limit the scope of the simulation, and thus the depth of the user experience. In this study, we developed a process by which to realize virtual environments that are based entirely on scenes in the real world. In modeling the real world, the proposed scheme divides scenes into discrete objects, which are then replaced with virtual objects. This enables users to interact in and with virtual environments without limitations. An RGB-D camera is used in conjunction with simultaneous localization and mapping (SLAM) to obtain the movement trajectory of the user and derive information related to the real environment. In modeling the environment, graph-based segmentation is used to segment point clouds and perform object segmentation to enable the subsequent replacement of objects with equivalent virtual entities. Superquadrics are used to derive shape parameters and location information from the segmentation results in order to ensure that the scale of the virtual objects matches the original objects in the real world. Only after the objects have been replaced with their virtual counterparts in the real environment converted into a virtual scene. Experiments involving the emulation of real-world locations demonstrated the feasibility of the proposed rendering scheme. A rock-climbing application scenario is finally presented to illustrate the potential use of the proposed system in AR applications.

Keywords: augmented reality; simultaneous localization and mapping; point cloud segmentation; shape fitting; object replacement

1. Introduction

Two technologies are currently used to create virtual environments or connect cyberspace with the real world: virtual reality (VR) and augmented reality (AR). VR users wear a head-mounted display that creates the illusion that they are in a virtual environment [1]. This approach generally requires positioning devices to locate the user within the environments (real and simulated). VR technologies allow users to move within the confines of the space defined by the positioning scheme. However, the need to wear closed head-mounted displays makes it nearly impossible to obtain information from the outside world. In other words, all objects within a user's range of movement must be removed to avoid collisions.

AR [2] combines virtual objects within a scene of the real world, thereby removing the primary limitation of VR and eliminating the need to simulate every entity within the scene. This has led to the

development of numerous AR applications, such as navigation systems for the blind [3], devices to assist in component assembly [4], and methods to facilitate surgical procedures [5]. The fact that current AR technologies simply place a number of virtual objects within a real environment greatly limits the depth of the simulation. In this study, we developed a scheme by which to replace all of the objects in a real-world environment with equivalent virtual entities, i.e., creating an AR environment based entirely on scenes from the real world. This is meant to allow users to experience virtual environments without being limited by the real-world environment in which their bodies reside.

As shown in Figure 1, the proposed system comprises three parts: (1) mapping, (2) point cloud segmentation, and (3) shape fitting. Mapping uses the differences between two images captured by a moving camera to characterize the movement relationship and establish 3D environmental point clouds. Point cloud segmentation uses environmental information to characterize the relationship between points and segment the environment into independent objects during the mapping process. Shape fitting involves identifying the geometric features of segmented objects and using geometric similarities as the basis for the replacement of virtual objects in our system.

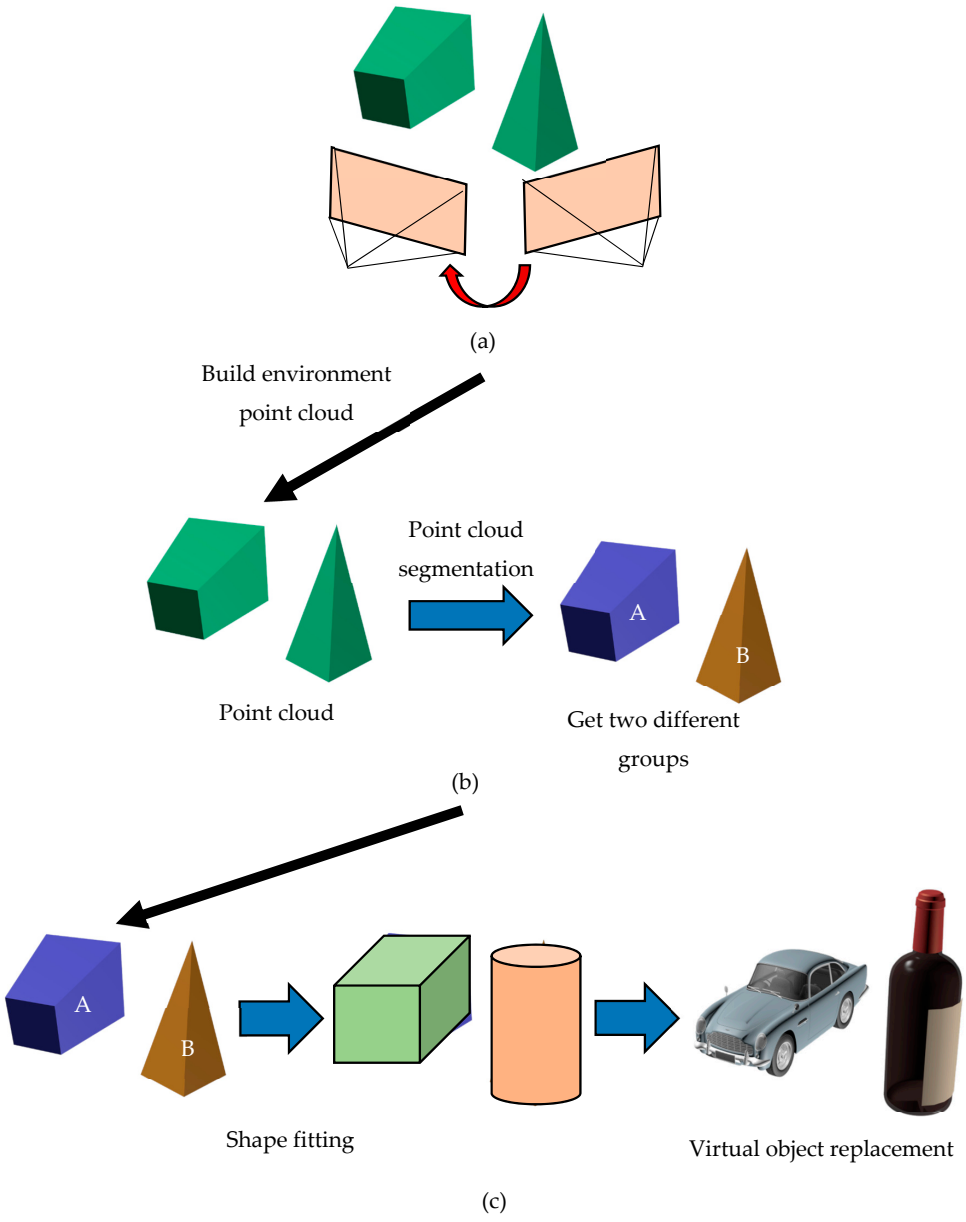


Figure 1. Flow chart of the proposed system. (**a**) Mapping, (**b**) point cloud segmentation, (**c**) shape fitting.

We developed an object replacement scheme that allows users to segment scenes and perform environment modeling even as the cameras are moving. This scheme also makes it possible to obtain shape information of segmented objects. The segmentation and shape fitting results can then be used to construct virtual objects to replace (within the virtual world) actual objects encountered (within the real world). Our ultimate objective was to eliminate limitations imposed by the real environment, which could otherwise limit the ability of the user to experience the virtual environment and enable the application of AR technology to a diversity of applications. The contributions of this study are as follows:

1. We developed a comprehensive rendering system combining mapping, point cloud segmentation, and shape fitting.
2. In point cloud segmentation, local loop closure optimization is used to reduce local errors, and global loop closure eliminates errors from the positions pertaining to the origin and destination, without affecting the segmented map.
3. Prior to point cloud segmentation, supervoxel generation and plane detection are used for preprocessing methods aimed at reducing the number of segments and overall computation time.

The remainder of this paper is organized as follows. Section 2 reviews the work relevant to the current study. Section 3 introduces the proposed mapping scheme. Section 4 outlines point cloud segmentation. Section 5 describes our approach to shape fitting using the superquadric method. Section 6 describes the replacement of actual objects with their virtual counterparts in a real-world scene. Conclusions are drawn at the end of the paper.

2. Related Work

The proposed system uses simultaneous localization and mapping (SLAM) [6–19] to generate point cloud information of the real environment for mapping. In [6], an extended Kalman filter (EKF) method was used in conjunction with a laser rangefinder and SLAM algorithm for use by a robot equipped with a monocular camera. In [7–9], multi-threading SLAM methods were used to divide tracking and mapping into two independent threads, while bundle adjustment (BA) was used to refine the accuracy of camera positions. In [10], Oriented Fast and Rotated BRIEF (ORB) feature detection was combined with additional threads for loop closure. In [11,12], dense point clouds were used for tracking and mapping. That scheme is referred to as a direct SLAM algorithm. In [13], a direct method was combined with a non-direct method (feature-based method) to enhance computation speeds. The above studies focused on obtaining camera poses and improving SLAM computation speeds. Other studies [14–19] investigated surface reconstruction from point clouds. Nevertheless, the aforementioned SLAM algorithms focused on mapping, the applications of which are limited. In this study, we sought to move beyond mere modeling of real environments. Our objective was to perform object segmentation in order to replace real-world entities with virtual replicas.

Researchers dealing with object segmentation have applied object recognition to mapping results to segment scenes [20–22]. Computer-aided design (CAD) models can be used as an alternative to object recognition [22]. However, the a priori completion of models (via training) is not conducive to the segmentation of unknown objects. Methods that require the a priori establishment of datasets are also greatly limited in terms of applicability. Other segmentation methods, such as region growing ones [23,24] are highly efficient. However, they require that users define the locations of seed points and tend to be susceptible to noise interference. The Hough transform [25] is highly robust against noise. However, it imposes a heavy computational burden. Thus, it is really only suitable for the segmentation of basic shapes, such as cylinders. Graph-based segmentation [26,27] does not require the definition of seed points and is not limited in terms of object shape. Therefore, training is not required. However, all of the above-mentioned methods are applicable only to single frames or segmentation results pertaining to an environment that has already been modeled. These methods have not been applied to the segmentation of objects in multiple consecutive frames. The massive

volume of information in real environments imposes a heavy computational burden, and segmentation cannot commence until the environment has been modeled entirely. In this study, we developed a segmentation process applicable to environments with multiple consecutive frames.

Following the completion of object segmentation, the physical meaning of the information (e.g., shape and dimensions) must be defined to serve as the basis for virtual objects subsequently inserted into the scene. In this study, we employed shape fitting to derive the shapes, dimensions, positions, and directions of segmentation point groups in order to create virtual objects that are representative of the real-world entities on which they were modeled. In [28,29], random sample consensus (RANSAC) and mean shift methods were used to identify objects and poses by group. However, these methods depend heavily on information from two-dimensional (2D) images and require off-line training. In [30,31], superquadrics were used to obtain shape information to avoid the need for off-line training by deriving shape information from three-dimensional (3D) information. In this study, we also adopted superquadrics to derive the shape of object point groups.

3. Mapping

In this study, we used a standard RGB-D camera (Microsoft Kinect) to obtain color images with depth information. In the mapping step, the color and depth images underwent preprocessing to derive features and feature descriptors, identify features similar to those in the previous frame, and then use said features to estimate the pose transformation from the previous image into the current image. Finally, loop closure was used to eliminate the degree of reprojection error to refine the camera pose. Figure 2 presents a flow chart of this process. The symbols used in the following rendering process can be found in the appendix Table as a reference.

3.1. Image Preprocessing

Principal component analysis (PCA) [32] was applied to 2D images to remove noise and decrease dimensionality as well as the number of features.

Given a 2D input image (640 × 480 pixels), we assumed that there were $n = s \times s$ patches from which to obtain m pieces of sample:

$$\{x_1, x_2, \ldots, x_m\} \in R^n \tag{1}$$

For each patch, we first calculated the mean μ:

$$\mu = \frac{1}{m}\sum_{j=1}^{m} x_j \tag{2}$$

where x_j denoted the jth piece of sample. To obtain the eigenvectors, we calculated the covariance matrix Σ:

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m}(x_j - \mu)(x_j - \mu)^T \tag{3}$$

After obtaining the eigenvalues and eigenvectors, we selected the top k eigenvectors as the basis for the construction of a low-dimensional space. The original data was mapped into the low-dimensional space, thereby completing dimension reduction.

For 3D images, we used the intrinsic parameters of the camera to obtain 3D point cloud coordinates of the depth values, as follows:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & u_0 & 0 \\ 0 & f & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = K[I|0]P \tag{4}$$

where $[u, v]^T$ were the coordinates of the image plane and also the perspective projection of points $\mathbf{P} = [X_c, Y_c, Z_c]^T$ in a 3D space on the image plane, **K** was the intrinsic parameter matrix of the camera, and f represented the focal length of the camera.

The relationship between the world coordinate system and camera coordinate system can be expressed using the extrinsic parameters of the camera, as shown in Equation (5), using rotation matrix **R** and translation matrix **t**. We generally assumed that the camera coordinate system applied to the first frame is the world coordinate system.

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} = [R|t] \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \quad (5)$$

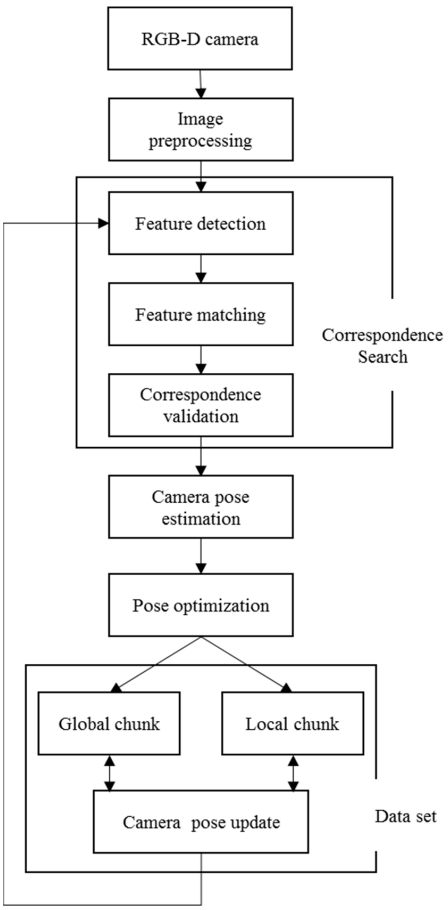


Figure 2. Flow-chart showing proposed mapping process.

3.2. Feature Extraction and Matching

We applied the speeded-up robust features (SURF) method [33] to each new frame to identify the features and feature descriptors in the current frame as well as the corresponding features in the previous frame. This information was then used to derive changes in posture that occur as the user moves.

Similar objects in an environment can lead to false matches, which must be removed. To this end, we used the RANSAC algorithm to obtain optimal feature matching results [34]. If the distance between two matched points exceeded δ_{depth}, then we deemed the two points as a false match. If the above condition was met, then the number of points matching the inliers must have exceeded a given threshold value in order to estimate the change in camera posture between the two frames.

3.3. Pose Optimization

Calculation errors and measurement errors from depth sensors can lead to the gradual accumulation of errors as the user moves, eventually resulting in environment modeling errors. Thus, we adopted loop closure [35] to enhance system stability by adjusting the camera postures and optimizing the movement trajectories. Suppose that there are n 3D points and m images in a given space. Let the intrinsic and extrinsic camera parameters of image j be vector $\mathbf{a}_j$, and 3D point i be vector $\mathbf{b}_i$; the point i seen in image j is $\mathbf{x}_{ij}$, $Q(\mathbf{a}_j, \mathbf{b}_i)$ denotes the projection point of object point $\mathbf{b}_i$ under camera $\mathbf{a}_j$. Thus, reprojection error E can be defined as follows:

$$\mathrm{E} = \sum_{\mathrm{i}=1}^{\mathrm{n}} \sum_{\mathrm{j}=1}^{\mathrm{m}} \left| \mathrm{Q}(\mathbf{a}_\mathrm{j}, \mathbf{b}_\mathrm{i}) - \mathbf{x}_\mathrm{ij} \right|^2 \tag{6}$$

We then solved the optimal intrinsic and extrinsic camera parameters $\mathbf{a}_j$ and the 3D feature coordinate $\mathbf{b}_i$ and minimized E in posture optimization as a non-linear least squares problem.

We developed a scheme to enable loop closure in a large-scale environment, while taking into account continuity in the subsequent segmentation of objects. We first established two chunks of data in which were stored the optimization parameters required for local loop closure and global loop closure.

The chunk for local loop closure stores n pieces of image data. The first-in-first-out strategy was used to obtain the new frame to ensure that the oldest frame is extracted. The information obtained from the chunk was used in subsequent processing, such as image segmentation, and was not altered after extraction. This process is illustrated in Figure 3.

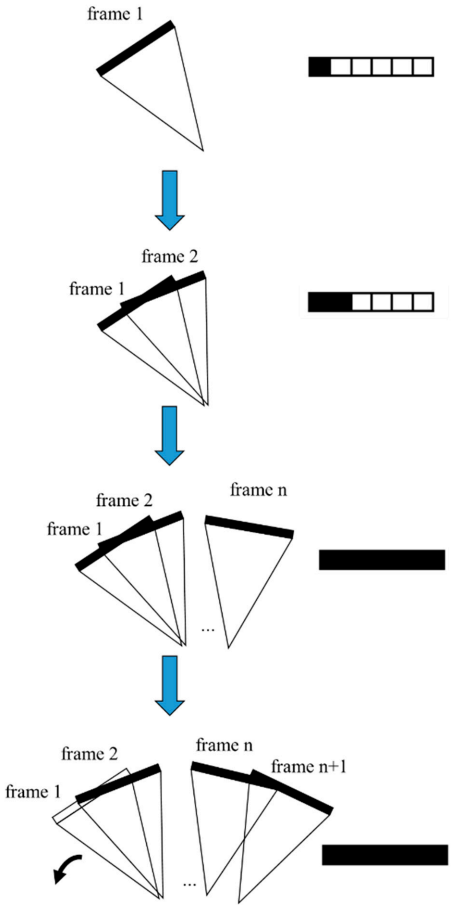


Figure 3. Flow chart showing the process of local loop closure.

The global loop closure chunk stores information from the first few images for use in optimizing the entire environment, i.e., the final step in environment modeling.

4. Point Cloud Segmentation

The point clouds obtained during mapping were insufficient to differentiate between different objects. Thus, the point clouds must have been segmented to facilitate subsequent processing. Figure 4 presents a flow chart showing the process of point cloud segmentation. The original point clouds were preprocessed prior to segmentation in order to reduce the amount of data required for segmentation. This involved the supervoxel generation and plane detection, as shown in Figure 4.

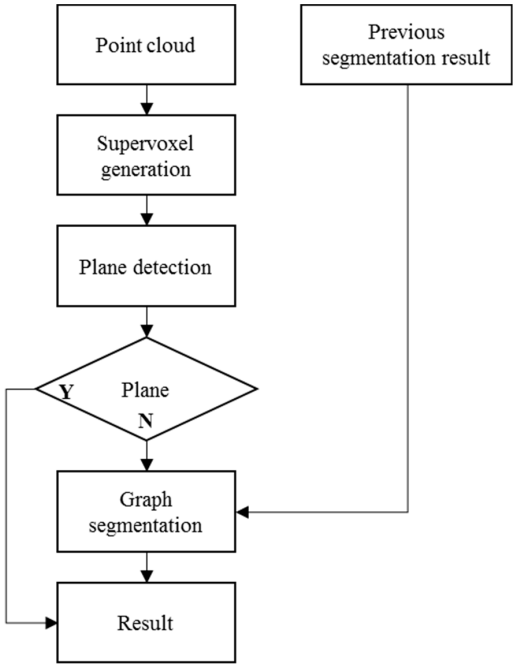


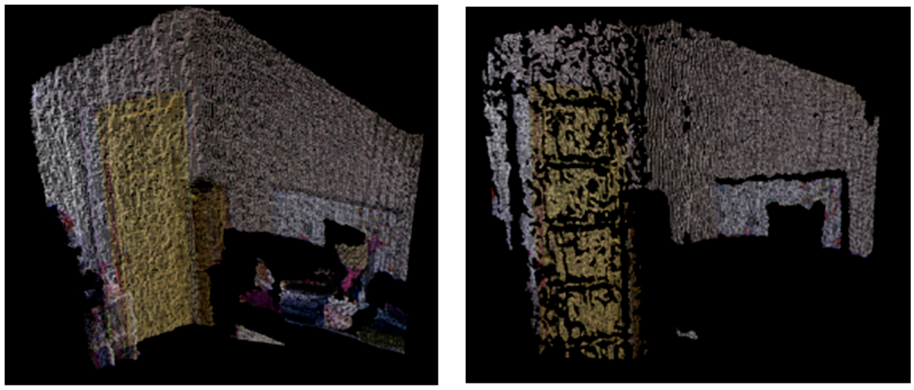
Figure 4. Flow chart showing the process of point cloud segmentation.

4.1. Supervoxel Generation

We generated supervoxels to reduce the number of point clouds and increase computation speed without losing adjacency relations between points. Supervoxel generation is based on the concept of voxel cloud connectivity segmentation (VCCS) [36], which uses color, the distance between points, and fast point feature histograms (FPFH) to divide voxels into groups. To enhance overall calculation efficiency, only the distance between points and the normal direction were used to generate supervoxels.

4.2. Plane Detection

Following the completion of supervoxelization, the system began segmentation of the scene. The scene was first divided into two parts: planar and non-planar point clouds. The purpose of this was to remove planes and thereby increase the efficiency of object segmentation. Furthermore, the planes were considered complete point groups that did not require further segmentation. In this study, planes were identified using the RANSAC method [34], in which the plane equation of three random points was calculated, and then other points were tested to see whether they belonged to this plane. When a set number of points did belong to a particular plane, they were considered a plane unto themselves. Plane detection results are shown in Figure 5.



(**a**) (**b**)

Figure 5. Plane detection results. (**a**) Original image, (**b**) obtained planes.

4.3. Graph-Based Segmentation

In this study, we used the methods outlined in [28,29] to segment objects. This approach involves the use of graphs to construct models, wherein the vertices in a graph are used as 3D points. The edges in graphs represent adjacency relations between points. Therefore, we drew edges between the point in question and adjacent points. The edge weights indicated the degree of geometric similarity between points.

4.3.1. Graph Construction

As shown in Figure 6, there are four types of adjacency relationship between points: stair-like, convex, concave, and invalid. In real scenes, stair-like adjacency is common between objects located at various distances from the viewer, such as two books stacked together. Convex adjacency is commonly seen in object surfaces, such as the corner of a box or the surface of a cup. Concave adjacency is typically found where two objects intersect, such as between the surface of a table and an object on the table. Invalid adjacency generally only occurs when two objects of different height are placed together.

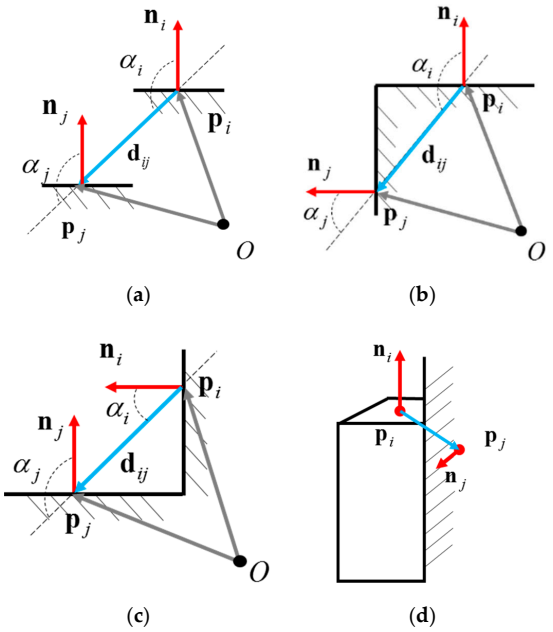


Figure 6. Types of adjacency between points. (**a**) Stair-like adjacency, (**b**) convex adjacency, (**c**) concave adjacency, (**d**) invalid adjacency.

Suppose there are two supervoxels, v_i and v_j. The centers of the supervoxels are p_i and p_j, and the normal directions of the two points are n_i and n_j. We determined whether the adjacency relation between the points was convex or concave as follows:

$$Convex(v_i, v_j) = \begin{cases} \text{true if}(\mathbf{n}_j - \mathbf{n}_i)\cdot(\mathbf{p}_j - \mathbf{p}_i) > 0 \\ \text{false} \quad \text{otherwise} \end{cases} \tag{7}$$

Given angles α_i and α_j, we denoted the adjacency relation between the points as stair-like when the two angles were nearly parallel and below the threshold value θ_{thresh}, as follows:

$$Step(v_i, v_j) = \begin{cases} \text{true if}(\alpha_i - \theta_{thresh})\cdot(\alpha_j - \theta_{thresh}) > 0 \\ \text{false} \quad \text{otherwise} \end{cases} \tag{8}$$

Using Equations (7) and (8), we considered the geometric relationships of most connecting edges. However, the real environment is extremely complex. Therefore, we must also take the rarer invalid adjacency into account. Using the external product of the normal directions $\mathrm{s} = (\mathrm{n_i} \times \mathrm{n_j})$ and the vector of the two points $\mathrm{d} = (\mathrm{p_j} - \mathrm{p_i})$, we defined angle θ_{ij} as

$$\theta_{ij} = \min(\angle(\mathbf{d}, \mathbf{s}), \angle(\mathbf{d}, -\mathbf{s})) \tag{9}$$

where $\angle(:,:)$ represented the angle between the two vectors. If θ_{ij} was less than the preset threshold value, then the two points were deemed to belong to different objects. The threshold value determined the weight value between two points.

4.3.2. Graph Construction

Once the setting of weights between points was completed, segmentation could commence. We employed the graph-based segmentation method using adaptive thresholds to determine the degree of similarity between two neighboring points. We first regarded points V_i of a supervoxel as a single group S_i. The image segmentation algorithm is outlined in the following:

Step 1. Calculate the degree of similarity between each supervoxel centroid and its neighboring centroids, and obtain a graph with n points and m edges;

Step 2. Arrange similarity e of the edges in ascending order to obtain $\mathbf{E} = (e_1, \ldots, e_m)$, where E is a set of similarity values $e_1, \ldots, e_m$;

Step 3. Apply Step 4 to edge q, $q = 1, \ldots, m$;

Step 4. Perform comprehensive judgment of e_q. Suppose that there are two connected vertices. If they do not belong to the same component, then the maximum degree of internal similarity between the two, τ, is calculated as follows:

$$\tau = \max(I_i + \frac{\delta}{|S_i|}, I_j + \frac{\delta}{|S_j|}) \tag{10}$$

where I_i and I_j denoted the internal difference of a component, δ indicated the threshold value, and $|S|$ represented the number of points within point group S.

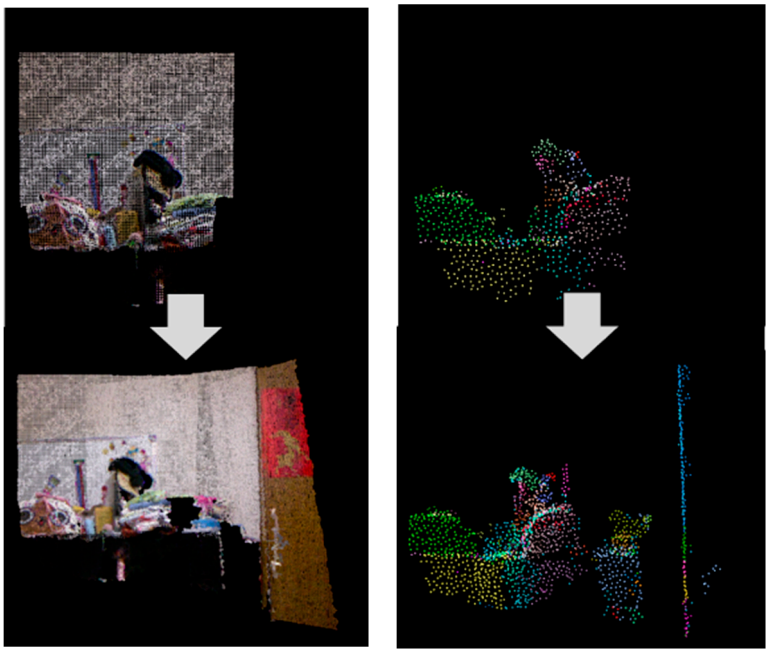
If ω_{ij}, (i.e., the degree of similarity between the two) exceeds τ, then the two groups were combined and updated as follows:

$$\begin{aligned} S_k &= S_i \cup S_j \\ I_k &= \omega_{ij} + \tfrac{\delta}{|S_k|} \end{aligned} \tag{11}$$

4.3.3. Segmentation Post-Processing

The process outlined in the previous section was used to complete the segmentation of the point groups currently used as inputs. The proposed system performed mapping and segmentation

simultaneously, rather than waiting for the user to finish an action before initiating segmentation. Thus, it was necessary to integrate the segmentation results that overlapped previous frames or were from the same object. Figure 7 presents the results obtained from two frames following point cloud segmentation and integration after supervoxel generation and plane removal.



(**a**) (**b**)

Figure 7. Results of post-processing for image segmentation. (**a**) Point cloud generation, (**b**) point cloud segmentation and post-processing.

In Figure 7, we can see that if the overlapping parts of two frames contained different groups, and the distance and normal angle were less than the threshold values, then the two groups were associated with the same object. This completed the point cloud segmentation of multiple frames during motion.

When segmentation and integration were applied to a set number of frames, the system began shape fitting based on the segmentation results for the range in question. However, before initiating shape fitting, it was necessary to conduct a final check of the segmentation results. In previous steps, large planes had already been removed via plane detection. However, smaller planes could still result in erroneous judgments due to the fact that the number of planes failed to reach the threshold value. The segmentation results may have contained fragmented point groups or planes resulting from measurement errors imposed by depth sensors and/or the creation of rugged shapes from planes distal to the depth sensors. We sought to overcome this problem by conducting a final correction and confirmation step, in which point groups that were too small, flat, or slim, were integrated with surrounding point groups or discarded.

5. Shape Fitting

Even after obtaining the results of point cloud segmentation for a given range, the point groups themselves still had no physical meaning. Thus, we must have identified the shapes represented by the point groups and then inserted objects with a similar form. In this study, we adopted the method described in [31] in which superquadrics are used to obtain the shape of unknown objects.

5.1. Inside-Outside Function

Superquadrics are a set of geometric shapes. In a basic superquadric, the center is assumed to be the origin, the parametric expression of which is written as follows:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a_1 \cos^{\varepsilon 1}\eta \cos^{\varepsilon 2}\omega \\ a_2 \cos^{\varepsilon 1}\eta \sin^{\varepsilon 2}\omega \\ a_3 \sin^{\varepsilon 1}\eta \end{bmatrix}, \begin{matrix} -\frac{\pi}{2} \le \eta \le \frac{\pi}{2} \\ -\pi \le \omega < \pi \end{matrix} \tag{12}$$

where x, y, and z are the coordinates of a point on the superquadric, a_1, a_2, and a_3 are the dimensions of the superquadric on the various axes, and ε_1 and ε_2 represent the shape of the superquadric.

Using the above equation, we could derive the inside-outside function F, as shown in Equation (13). This equation could be used to determine whether a given point was on the inside or outside of the superquadric. For instance, $F = 1$ meant that the point was on the surface of the superquadric, whereas $F < 1$ and $F > 1$ indicated that the point was on the inside and outside of the superquadric, respectively.

$$F(x, y, z) = \left(\left(\frac{x}{a_1} \right)^{\frac{2}{\varepsilon 2}} + \left(\frac{y}{a_2} \right)^{\frac{2}{\varepsilon 2}} \right)^{\frac{\varepsilon 2}{\varepsilon 1}} + \left(\frac{z}{a_3} \right)^{\frac{2}{\varepsilon 1}} \tag{13}$$

However, the above equation relied on the fact that the center of the superquadric lied at the origin and that the coordinate axes were aligned with the world coordinate axes. In the real world, the world coordinates of an object are different from the model coordinates, which means that a rotation matrix $\mathbf{R}$ and translation matrix $\mathbf{t}$ are required to convert the world coordinates of the point cloud of the real object into model coordinates. Suppose that world coordinate point $\mathbf{X}$ becomes model coordinate point $\mathbf{x}$ after conversion:

$$\mathbf{X} = [\mathbf{R}|\mathbf{t}]\mathbf{x} \tag{14}$$

$$\mathbf{x} = [\mathbf{R}|\mathbf{t}]^{-1}\mathbf{X} \tag{15}$$

Thus, expressing the superquadric model of an object in the real world requires at least 11 parameters:

$$\begin{matrix} F(x_w, y_w, z_w) = F(x_w, y_w, z_w; \\ a_1, a_2, a_3, \varepsilon_1, \varepsilon_2, f, \theta, \psi, p_x, p_y, p_z) \end{matrix} \tag{16}$$

where θ, f, ψ are the parameters of the rotation matrix, and p_x, p_y, p_z are the parameters of the translation matrix. Below, we express these 11 parameters using the following set: $\Lambda = \{\lambda_1, \lambda_2, L, \lambda_{11}\}$.

We employed the Levenberg–Marquardt (LM) algorithm to obtain the optimal solution for the parameters of point group shapes by minimizing the value of the equation:

$$\min_{\Lambda} \sum_{i=1}^{n} \left(F(x_i, y_i, z_i; \lambda_1, \ldots, \lambda_{11}) - 1 \right)^2 \tag{17}$$

However, the LM approach produces a local optimal solution and is extremely sensitive to initial parameter settings. To illustrate the influence of the initial optimization parameters, we randomly selected the following initial parameters:

$$\begin{matrix} a_1 = 0.5, a_2 = 0.5, a_3 = 0.5, \varepsilon_1 = 1, \varepsilon_2 = 1, \\ f = 0^{\circ}, \theta = 0^{\circ}, \psi = 0^{\circ}, p_x = 0, p_y = 0, p_z = 0 \end{matrix}$$

This means performing optimization without choosing suitable initial parameter settings. Figure 8 presents the simulation results, in which the white points indicate the model that needs to be estimated and the red points are the estimation results. Clearly, deviations in the initial parameters and model to be estimated can result in significant simulation errors.

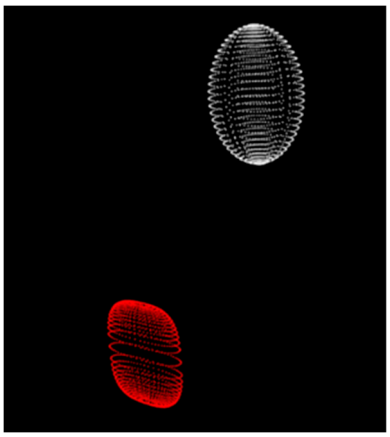


Figure 8. Simulation results of shape fitting using unsuitable initial parameters.

5.2. Initial Parameter Settings

To obtain estimates of the initial parameters for the point group, we used PCA to derive the main direction of the point group and the directions of the other two orthogonal axes, as shown in Figure 9, where $\mathbf{v}_1$, $\mathbf{v}_2$, and $\mathbf{v}_3$ are the eigenvectors of the point group.

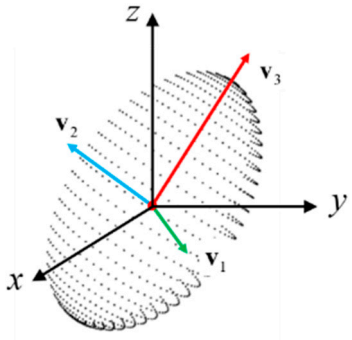


Figure 9. Diagram of point cloud eigenvectors.

We first used the eigenvectors to obtain approximate values of the rotation parameters θ, f, ψ. The lengths of the projections of the point group on the eigenvectors determined the size parameters of the point cloud shape: a_1, a_2, and a_3. Information pertaining to the center point of the point group could be used to derive translation parameters p_x, p_y, and p_z. As for the shape parameter settings, we initially made the assumption of an ovoid. Figure 10 presents the simulation results obtained using these parameter settings. Again, the white points indicate the model that needs to be estimated, and the red points indicate the simulation results. Clearly, the adoption of suitable initial parameter settings enhances the accuracy of shape locations and dimensions.

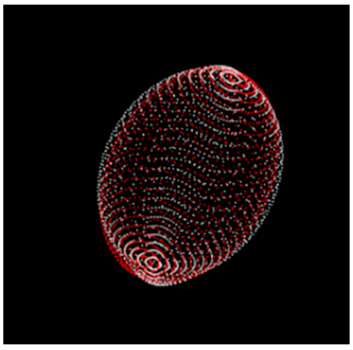


Figure 10. Simulation results of shape fitting using suitable initial parameter settings.

5.3. Point Group Normalization

Under real-world conditions, the size of the input point group varies. The use of projection relationships allows us to obtain rough estimates of the shape dimensions. However, dimensions that are overly large or small can still produce incorrect results. To enhance the stability of the output results, we first normalized the point groups prior to shape fitting to ensure that all of the dimensions of the point groups fell within a set range.

In the following, we respectively present coordinate $\mathbf{X}_i$ of the point group before and after normalization:

$$\begin{bmatrix} x_j \\ y_j \\ z_j \\ 1 \end{bmatrix}_{\text{Normalized}} = \mathbf{T} \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix}_{\text{Original}} \tag{18}$$

$$\mathbf{T} = \begin{bmatrix} s_x & 0 & 0 & -s_x\overline{x} \\ 0 & s_y & 0 & -s_y\overline{y} \\ 0 & 0 & s_z & -s_z\overline{z} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{19}$$

where (s_x, s_y, s_z) are the scaling ratios, and $\overline{\mathrm{X}} = \begin{bmatrix} \overline{x} & \overline{y} & \overline{z} & 1 \end{bmatrix}^T$ is the center of the point group. Let l be the mean distance between the point and the center, and s be the scaling rate. Assuming that the point group contains n points, we derive the following:

$$\begin{aligned} s_x = s_y = s_z &= \tfrac{s}{l} \\ l &= \tfrac{1}{n}\sum_{i=1}^{n} \left\| \mathbf{X}_i - \overline{\mathbf{X}} \right\| \end{aligned} \tag{20}$$

Performing normalization before estimating and solving the initial parameters enables the generation of a geometric shape that is highly similar to that of the point group.

For AR applications, we can replace the identified objects with similar virtual objects using pre-established databases, model database, and scene database. The model database records the shape features of object models, and the scene database stores the index values and probability of the objects relevant to various scenes. Assume that the scene to be generated is known. The step after acquiring the size, shape, and position of the object point group is to search the model information from the database and identify the relevance between the object and the scene for a best-fitted model. The object replacement procedure can then be finished by importing this fitted model into the real scene (or virtual scene) and moving the model to the position of object point group with the correct orientation.

6. Experiments and Discussion

We applied mapping, point cloud segmentation, and shape fitting procedures in succession to a variety of typical bedroom scenes to verify the feasibility of the proposed virtual object replacement scheme.

6.1. System Description

The cameras commonly used for SLAM can be divided into monocular, multiocular, RGB-D, and dynamic vision sensors (event cameras). RGB-D cameras such as Kinect are inexpensive and provide depth information quickly and efficiently. We, therefore, adopted an RGB-D camera as our main instrument for capturing images. Figure 11 presents the framework of the system proposed in this study. Kinect for Windows v1 was used to obtain color and depth images, which were then input into a personal computer equipped with an Intel Core i7-7700 CPU 3.60 GHz processor for program development and algorithm execution. The results were then output to a display.

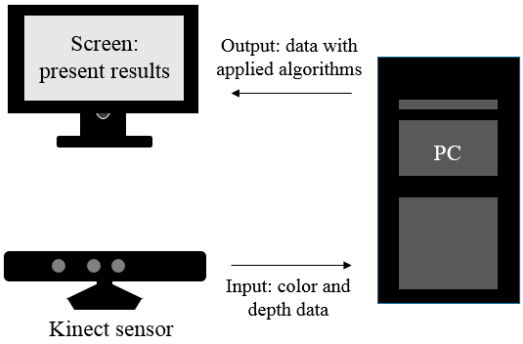


Figure 11. Hardware used for system realization.

6.2. *Verification of the Proposed Rendering System*

Figure 12 presents the results of the mapping experiment involving an actual private bedroom. During the experiment, the user held the camera, while continuously moving throughout the room. Local loop closure and optimization were performed simultaneously. When the user completed one circuit around the room, global loop closure was performed to prevent the accumulation of errors. In the mapping experiments, the data acquisition and reconstruction time were approximately 1~2 s by using CPU for computation only, and the reconstruction accuracy was within the range of 10 mm. The results demonstrate that the proposed loop closure strategy (Section 3.3) is highly effective in modeling an indoor environment for subsequent object segmentation and shape fitting.

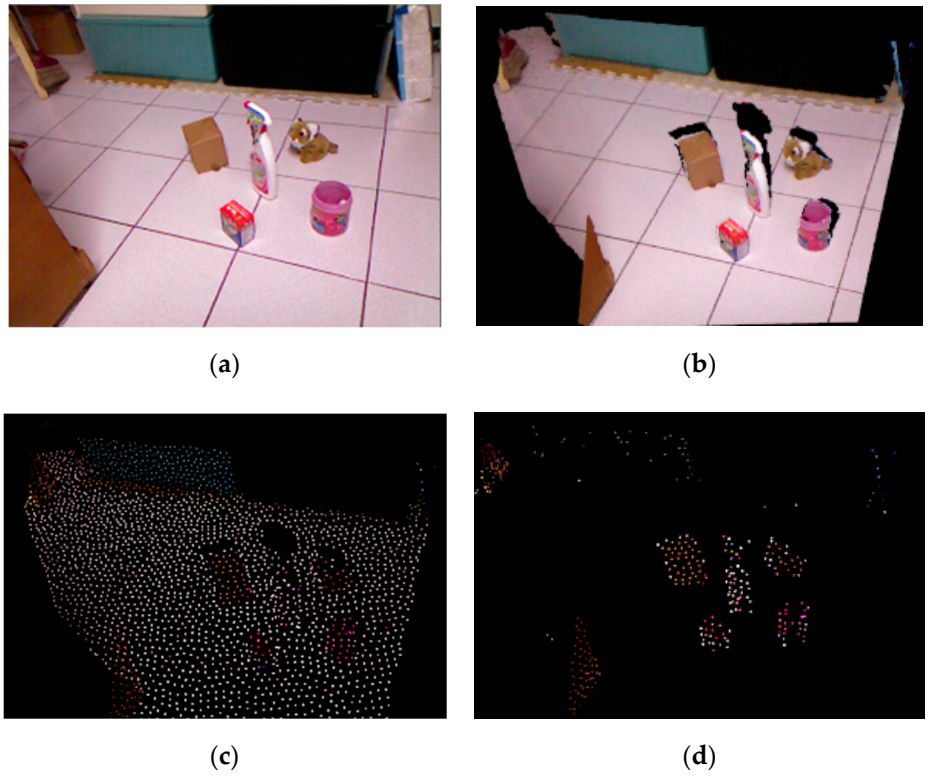
Figure 12. Mapping results (top view).

Figure 13a presents the test scene used for point cloud segmentation. As previously mentioned, the supervoxels and plane detection used in this study reduce the number of point clouds to decrease computational overhead. As shown in Table 1, we compared the segmentation times resulting from the various procedures to elucidate the influence of these two steps on computation time. Clearly, the use of these two steps can significantly reduce (by approximately 60%) the number of point clouds. This greatly reduces segmentation time and overall computation time, i.e., enhancing the efficiency of point cloud segmentation.

Table 1. Comparison of segmentation times.

Original Number of Point Clouds	25023		
	Test 1	**Test 2**	**Test 3**
Supervoxelization Time spent	No N/A	Yes 1.674 s	Yes 1.674 s
Plane detection Time spent	Yes 4.997 s	No N/A	Yes 0.64 s
Number of segmentation point clouds	3284	3215	952
Segmentation time	49.674 s	15.178 s	2.207 s
Total time spent	54.671 s	16.852 s	4.521 s

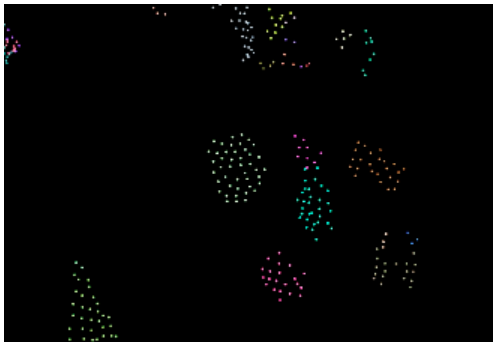
We then conducted an experiment on object segmentation using the simple scene in Figure 13a. Figure 13b–e illustrate the segmentation process using information derived from a single frame. Figure 13b presents the 3D image obtained using a Kinect camera. Figure 13e presents the results after supervoxelization and plane removal. From this, we can clearly see that the proposed method is able to differentiate between a variety of objects. Note that some of the parts show signs of over-segmentation, which could be resolved by integrating smaller groups.



(**a**) (**b**)

(**c**) (**d**)

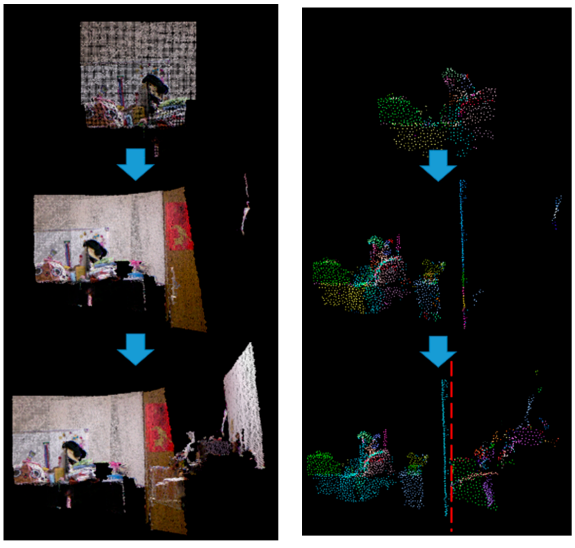
Figure 13. *Cont.*



(**e**)

Figure 13. Results of point cloud segmentation using a simple test scene. (**a**) test scene (2D image), (**b**) 3D point clouds obtained using Kinect camera, (**c**) supervoxelization results, (**d**) plane removal, (**e**) point cloud segmentation.

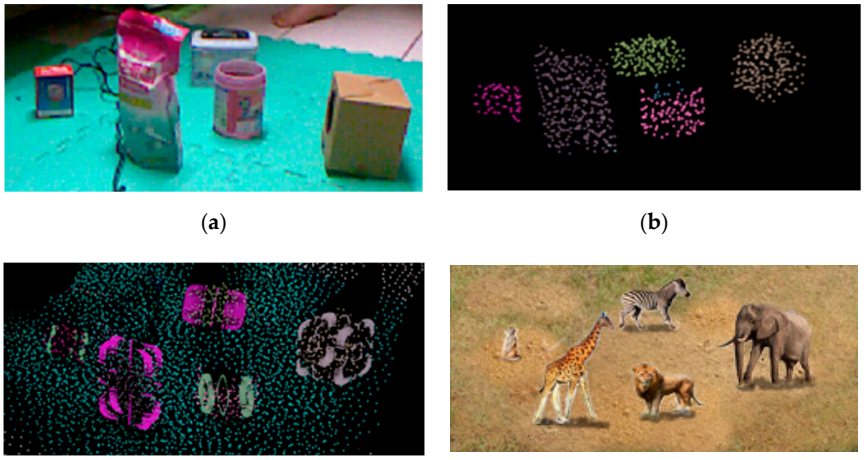
In the experiment above, the point clouds in each frame were integrated with the previous point clouds after segmentation was completed. This was because the images captured by the camera presented a consecutive sequence, which means that most of the objects in one frame were identical to those in the next frame. This allowed us to treat the overlapping portions or protrusions where two-point groups intersected as the same object suitable for integration. Furthermore, we could not be sure about the size of the environment, and computation time increased with the number of frames. Thus, the system began shape fitting immediately after a threshold number of segmentation frames had been met. Point groups were also checked at that time. In the event that the check results did not indicate a plane and the point groups in the current frame are connected to previously segmented point groups, we could assume that the object was no longer connected to the object in the following frame. Otherwise, we kept the point groups and integrated them with the point clouds in the next frame. Figure 14 illustrates the point cloud segmentation process applied to multiple frames captured in a complex environment. When the number of frames reached the threshold value, the overly fragmented or flat point clouds on the left side of the red line in Figure 14b were re-integrated, and the results applied directly to the subsequent shape fitting procedure. The regions on the right side of the red line were kept to serve as the basis for integration following completion of segmentation in the next image.



(**a**) (**b**)

Figure 14. Results of point cloud segmentation in a complex environment. (**a**) Mapping, (**b**) point cloud segmentation.

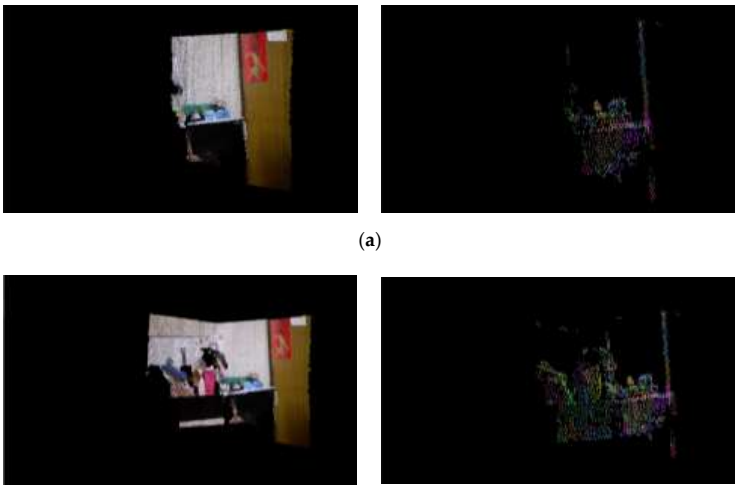
We then performed a shape fitting test using the simple scene, the results of which are presented in Figure 15. The results of modeling the scene using multiple frames and object segmentation are presented in Figure 15b. Application of the algorithm in Section 5 to the segmentation results for shape fitting generated the result in Figure 15c. If we place the required images on the flat point clouds and use related virtual objects in place of the shape fitting results, then elements in the real environment can be replaced with virtual ones. Suppose that we want to create a scene from the African plains. The final result might look like that shown in Figure 15d. The original objects can be replaced with objects presenting a similar geometric shape, such as an elephant, a tiger, a giraffe, a monkey, and a zebra. To verify the versatility of the presented methods, we conducted an additional experiment involving a complex scene in which the camera was moved from right to left, the results of which are presented in Figure 16. The proposed scheme has a number of advantages over existing object segmentation results (i.e., using machine-learning or graph-based methods). As shown in Figure 16a–d, our system is able to perform object segmentation during the mapping process (i.e., online object segmentation) and does not require *a priori* training to deal with objects in actual scenes. The shape-fitting results in Figure 16e demonstrate the effectiveness of the proposed methods in AR applications.



(**a**) (**b**)

(**c**) (**d**)

Figure 15. Results of virtual object replacement using simple scene involving segmentation of multiple frames and shape fitting. (**a**) Scene modeling (2D image, side view), (**b**) results of point cloud segmentation using multiple frames (top view), (**c**) shape fitting and plane point cloud (top view with green dots indicating the plane), (**d**) virtual scene.



(**a**)

(**b**)

Figure 16. *Cont.*

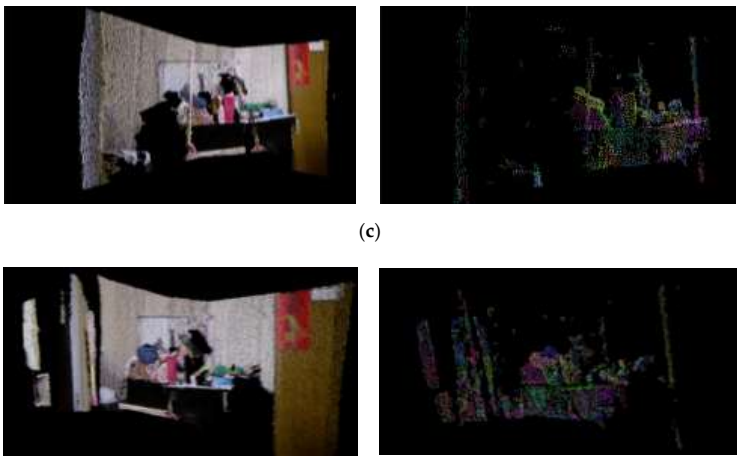


Figure 16. Mapping and the segmentation of objects in a complex scene by moving the camera from right to left. (**a**) Initial camera view, (**b**) intermediate camera view (I), (**c**) intermediate camera view (II), (**d**) final camera view, (**e**) shape fitting results obtained from the scanned scene.

7. Conclusions

In this study, we developed a comprehensive rendering scheme that includes real-time mapping, point cloud segmentation, and shape fitting for use in AR. We derived the extrinsic parameters of the RGB-D camera based on differences between image frames. We then modeled an accurate point cloud environment using the degree of similarity between points and segmented point clouds. We then added a threshold value to enable the system to complete point cloud segmentation and integration on its own, prior to shape fitting. Multi-frame shape fitting was used to obtain the primary shape, dimensions, and posture of the point groups.

7.1. Implementation Scenario

The proposed scheme could be used to replace real objects with corresponding virtual objects in game engines or simulation platforms. For example, a natural 3D wall with rocks could be reconstructed and projected through head-mounted displays to a gym wall to allow climbers to "feel" real physical environments. This would give the users a more realistic experience and eliminate many of the limitations on the size of the virtual environment imposed by elements in the environment in which the user resides.

7.2. Limitations

Currently, the proposed system is suitable only for single-room venues and applicable mainly to static environments.

7.3. Future Work

In the future, global loop closure information could be established in crucial areas, such as doorways, to enhance the scope of the system. Furthermore, Dynamic environment detection could be added in the future to make the mapping process more robust.

Author Contributions: Y.-S.C. and C.-Y.L. conceived and designed the experiments; Y.-S.C. performed the experiments and analyzed the results; C.-Y.L. wrote the paper.

Funding: This work was supported by the Ministry of Science and Technology, Taiwan, under grant number MOST-105-2628-E-011-004 and partially supported by the "Center for Cyber-physical System Innovation" from The Featured Areas Research Center Program within the framework of the Higher Education Sprout Project by the Ministry of Education (MOE) in Taiwan.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviation

Symbol	Definition
a_1, a_2, a_3	Dimensions of the superquadric on the various axes
$\mathbf{a}_j$	Intrinsic and extrinsic camera parameters of image j
$\mathbf{b}_i$	3D point i
$\mathbf{d}$	Vector of two points
E	Reprojection error
$\mathbf{E}$	A set of similarity values
e	Similarity of the edges
$F(x,y,z)$	Function to determine whether a given point is on the inside or outside of the superquadric
I_i, I_j	Internal difference of a component
$\mathbf{K}$	Intrinsic parameter matrix of the camera
k	Number of top eigenvectors
l	Mean distance between the point and the center
n_i, n_j	Normal directions of two points
$\mathbf{P}$	Points in a 3D space on the image plane
$\mathbf{p}_i$, $\mathbf{p}_j$	Centers of the supervoxels
p_x,p_y,p_z	Parameters of the translation matrix
$Q(\mathbf{a}_j,\mathbf{b}_i)$	Projection point of object point $\mathbf{b}_\mathrm{i}$ under camera parameters $\mathbf{a}_\mathrm{j}$
$\mathbf{s}$	External product of the normal directions
s	Scaling rate
$\|S\|$	Number of points within point group S
S_i	A single group
s_x,s_y,s_z	Scaling ratios
u,v	Coordinates of the image plane
$\mathbf{v}_1$,$\mathbf{v}_2$,$\mathbf{v}_3$	Eigenvectors of the point group
V_i	A supervoxel point
v_i, v_j	Supervoxels
x,y,z	Coordinates of a point on the superquadric
$\overline{\mathrm{X}}$	Center of the point group
$\mathrm{X_c}$, $\mathrm{Y_c}$, $\mathrm{Z_c}$	Camera coordinate of points
$\mathbf{x}_\mathrm{ij}$	Point i seen in image j
x_j	jth piece of image sample
X_w, Y_w, Z_w	World coordinate of points
α_i, α_j	Angles used to define the type of adjacency between points
δ	Threshold value used in image segmentation
δ_{depth}	Threshold value used in image matching
ε_1, ε_2	Shape of the superquadric
η,ω	Surface parameters of superquadric
θ,f,ψ	Parameters of the rotation matrix
θ_{thresh}	Threshold value used to determine the type of stair-like adjacency
τ	Maximum degree of internal similarity
ω_{ij}	Degree of similarity between two neighboring points

References

1. LaValle, S.M. *Virtual Reality*; Cambridge University Press: Cambridge, UK, 2016.
2. Dieter, S.; Tobias, H. *Augmented Reality: Principles and Practice*; Addison-Wesley Professional: Boston, MA, USA, 2016; ISBN 0321883578.
3. Yang, L.; Noelle, R.B.; Markus, M. Augmented Reality Powers a Cognitive Prosthesis for the Blind. Available online: https://doi.org/10.1101/321265 (accessed on 22 May 2018).

4. Evans, G.; Miller, J.; Pena, M.I.; MacAllister, A.; Winter, E. Evaluating the Microsoft HoloLens through an augmented reality assembly application. In Proceedings of the SPIE Defense, Anaheim, CA, USA, 7 May 2017; pp. 1–16.
5. Cui, N.; Kharel, P.; Gruev, V. Augmented reality with Microsoft HoloLens holograms for near infrared fluorescence based image guided surgery. In *Molecular-Guided Surgery: Molecules, Devices, and Applications III*; SPIE: Bellingham, WA, USA, 2017; Volume 10049. [CrossRef]
6. Davison, A.J.; Reid, I.D.; Molton, N.; Stasse, O. MonoSLAM: Real-time single camera SLAM. *IEEE Trans. Pattern Anal. Mach. Intell.* **2007**, *29*, 1052–1067. [CrossRef] [PubMed]
7. Klein, G.; Murray, D. Parallel tracking and mapping for small AR workspaces. In Proceedings of the Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality, Nara, Japan, 1 November 2007; pp. 225–234.
8. Klein, G.; Murray, D. Improving the agility of key frame-based SLAM. In Proceedings of the European Conference on Computer Vision, Marseille, France, 12–18 October 2008; pp. 802–815.
9. Klein, G.; Murray, D. Parallel tracking and mapping on a camera phone. In Proceedings of the Eighth IEEE and ACM International Symposium on Mixed and Augmented Reality, Orlando, FL, USA, 19–22 October 2009; pp. 83–86.
10. Mur-Artal, R.; Montiel, J.; Tardos, J. ORB-SLAM: A versatile and accurate monocular SLAM system. *IEEE Trans. Robot.* **2015**, *31*, 1147–1163. [CrossRef]
11. Engel, J.; Sturm, J.; Cremers, D. Semi-Dense Visual odometry for a monocular camera. In Proceedings of the IEEE International Conference on Computer Vision, Sydney, NSW, Australia, 1–8 December 2013; pp. 1449–1456.
12. Engel, J.; Schöps, T.; Cremers, D. LSD-SLAM: Large-scale direct monocular SLAM. In Proceedings of the European Conference on Computer Vision, Zurich, Switzerland, 6–12 September 2014; pp. 834–849.
13. Forster, C.; Pizzoli, M.; Scaramuzza, D. SVO: Fast semi-direct monocular visual odometry. In Proceedings of the IEEE International Conference on Robotics and Automation, Hong Kong, China, 31 May–7 June 2014; pp. 15–22.
14. Whelan, T.; Kaess, M.; Johannsson, H.; Fallon, M.F.; Leonard, J.J.; McDonald, J.B. Real-time large scale dense RGB-D SLAM with volumetric fusion. *Int. J. Robot. Res.* **2015**, *34*, 598–626. [CrossRef]
15. Kerl, C.; Sturm, J.; Cremers, D. Dense visual SLAM for RGB-D cameras. In Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems, Tokyo, Japan, 3–7 November 2013; pp. 2100–2106.
16. Meilland, M.; Comport, A.I. On unifying key-frame and voxel-based dense visual SLAM at large scales. In Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems, Tokyo, Japan, 3–7 November 2013; pp. 3677–3683.
17. Stuckler, J.; Behnke, S. Multi-resolution surfel maps for efficient dense 3D modeling and tracking. *J. Vis. Commun. Image Represent.* **2014**, *25*, 137–147. [CrossRef]
18. Whelan, T.; Leutenegger, S.; Salas-Moreno, R.F.; Glocker, B.; Davison, A.J. ElasticFusion: Dense SLAM without a pose graph. In Proceedings of the Robotics: Science and Systems, Rome, Italy, 13–17 July 2015.
19. Dai, A.; Nießner, M.; Zollofer, M.; Izadi, S.; Theobalt, C. BundleFusion: Real-time globally consistent 3D reconstruction using on-the-fly surface re-integration. *ACM Trans. Graph.* **2016**, *36*, 24. [CrossRef]
20. Chang, A.; Dai, A.; Funkhouser, T.; Halber, M.; Niessner, M.; Savva, M.; Song, S.; Zeng, A.; Zhang, Y. Matterport3d: Learning from RGB-D data in indoor environments. In Proceedings of the International Conference on 3D Vision (3DV), Qingdao, China, 10–12 October 2017; pp. 667–676.
21. Dai, A.; Chang, A.X.; Savva, M.; Halber, M.; Funkhouser, T.; Nießner, M. Scannet: Richly-annotated 3D reconstructions of indoor scenes. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 2432–2443.
22. Qi, C.R.; Su, H.; Mo, K.; Guibas, L.J. Pointnet: Deep learning on point sets for 3D classification and segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 77–85.
23. Rabbani, T.; Van Den Heuvel, F.A.; Vosselman, G. Segmentation of point clouds using smoothness constraint. In Proceedings of the ISPRS Commission V Symposium, Dresden, Germany, 25–27 September 2006; pp. 248–253.

24. Holz, D.; Behnke, S. Fast range image segmentation and smoothing using approximate surface reconstruction and region growing. In Proceedings of the International Conference on Intelligent Autonomous Systems, Jeju Island, Korea, 26–29 June 2012; pp. 61–73.
25. Ballard, D.H. Generalizing the Hough transform to detect arbitrary shapes. *Pattern Recognit.* **1981**, *13*, 111–122. [CrossRef]
26. Karpathy, A.; Miller, S.; Li, F. Object discovery in 3D scenes via shape analysis. In Proceedings of the International Conference on Robotics and Automation, Karlsruhe, Germany, 6–10 May 2013; pp. 2088–2095.
27. Xu, Y.; Hoegner, L.; Tuttas, S.; Stilla, U. Voxel-and graph-based point cloud segmentation of 3D scenes using perceptual grouping laws. In Proceedings of the ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Hannover, Germany, 6–9 June 2017; pp. 43–50.
28. Collet, A.; Berenson, D.; Srinivasa, S.S.; Ferguson, D. Object recognition and full pose registration from a single image for robotic manipulation. In Proceedings of the IEEE International Conference on Robotics and Automation, Kobe, Japan, 12–17 May 2009; pp. 48–55.
29. Collet, A.; Srinivasa, S.S. Efficient multi-view object recognition and gull pose estimation. In Proceedings of the IEEE International Conference on Robotics and Automation, Anchorage, AK, USA, 3–7 May 2010; pp. 2050–2055.
30. Duncan, K.; Sarkar, S.; Alqasemi, R.; Dubey, R. Multi-scale Superquadric fitting for efficient shape and pose recovery of unknown objects. In Proceedings of the IEEE International Conference on Robotics and Automation, Karlsruhe, Germany, 6–10 May 2013; pp. 4238–4243.
31. Strand, M.; Dillmann, R. Segmentation and approximation of objects in pointclouds using superquadrics. In Proceedings of the International Conference on Information and Automation, Zhuhai, Macau, China, 22–24 June 2009; pp. 887–892.
32. Shangjie, S.; Wei, Z.; Liu, S. A high efficient 3D SLAM algorithm based on PCA. In Proceedings of the IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems, Chengdu, China, 19–22 June 2016; pp. 109–114.
33. Bay, H.; Ess, A.; Tuytelaars, T.; Gool, L.V. SURF: Speeded Up Robust Features. *Comput. Vis. Image Underst.* **2008**, *110*, 346–359. [CrossRef]
34. Fischler, M.A.; Bolles, R.C. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM.* **1981**, *24*, 381–395. [CrossRef]
35. Angeli, A.; Doncieux, S.; Meyer, J.A. Real-Time visual loop-closure detection. In Proceedings of the IEEE International Conference on Robotics and Automation, Pasadena, CA, USA, 19–23 May 2008; pp. 1842–1847.
36. Papon, J.; Abramov, A.; Schoeler, M.; Worgotter, F. Voxel cloud connectivity segmentation—Supervoxels for point clouds. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Portland, OR, USA, 23–28 June 2013; pp. 2027–2034.