A Quick Algorithm for Snapping 3D Objects in Augmented Reality

Trien V. Do
Game Interface Research Center
Sejong University
Seoul, Republic of Korea
triendv@vnn.vn

Jong-Weon Lee
Mixed Reality & Interaction Laboratory
Sejong University
Seoul, Republic of Korea
jwlee@sejong.ac.kr

Abstract— **This paper describes a quick algorithm for snapping a moving object (being moved by a user) to existing objects in Augmented Reality (AR) environment. Because in AR, human-computer interactions are in the real time mode, the algorithm should be fast enough not to affect the scene rendering but still assure accuracy. It is currently designed to deal with primitive geometries such as cubes, cones, cylinders, spheres. And it should work well even after these geometries are transformed with scaling, rotating, moving operations. The main idea of the algorithm is that for each kind of geometry, some "hot spots" are defined. Each hot spot is a sphere with a certain radius. If the center point of the moving object is within a hot spot of any existing geometry it will be snapped to this geometry at that hot spot. This algorithm is already employed in an AR modeling system to evaluate its efficiency.**

Keywords: 3D snapping, Hot spot, Real time Algorithm, Augmented Reality

I. INTRODUCTION

Today, Augmented Reality has become an active research topic. Many AR systems have been developed in various areas. These systems always run in the real time mode and users can often intuitively interact with them through tangible interfaces such as markers [1] or gloves [2]. User's interactions are captured by camera systems and displayed on screens together with superimposed 3D computer-generated objects. All the systems try to make these interactions as natural and realistic as possible. Among various areas, 3D modeling has gained much interest. In 3D modeling systems, normally, all parts of a model must have some relative relationships. This means a new component of a model should be added closely to a certain existing part. Users of this kind of systems often use their hands to move objects and place them at appropriate positions. However, precisely sensing positions of objects in 3D space is not easy because it is hard to feel the depth of objects in the augmented 3D space. Even if they can do this well, placing an object exactly at a certain position is still too difficult because of stability of tracking, small hand movements etc. Snapping function is a solution to this problem.

Snapping is not too complex in 2D. Most of vector-based graphics applications support this feature. However, in 3D, it is not a simple problem. Even in famous commercialized products such as AutoCAD [3], 3Ds Max [4] and Maya [5], using this feature is not easy and it often takes quite much time to go through several steps. And the situation is especially complex when an object is rotated around axes. So the question is how snapping 3D objects can be solved in real time systems where performance is a crucial criterion. This paper describes a quick algorithm for snapping a moving object (being moved by a user) to existing objects in AR environment. As in AR, human-computer interactions are in the real time mode, the algorithm should be fast enough not to affect the scene rendering but still assure accuracy. It is currently designed to deal with primitive geometries such as cubes, cones, cylinders, spheres. It should work well even after these geometries are transformed with scaling, rotating, moving, and resizing operations. The main idea of the algorithm is that for each kind of geometry, some "hot spots" are defined. Each hot spot is a sphere with a certain radius. If the center point of the moving object is within a hot spot of any existing geometry it will be snapped to this geometry at that hot spot. This algorithm is already employed in an AR modeling system to evaluate its efficiency.

II. RELATED WORK

In [6], an algorithm for fast picking and snapping using a 3D input device and 3D cursor is introduced. First a multi-level bounding box check is described for 3D object picking operation. And then snapping is done by continuously performing pick action. In principle, a pick action has to be performed with every cursor movement, to detect if the cursor has to be snapped to an object. However in the described picking procedure, the nearest point calculation is the crucial time-consuming factor. So although the algorithm works well in the field of Virtual Reality-like CAD systems, it may not be fast enough in Augmented Reality environment.

III. ALGORITHM DESCRIPTION

In this session we describe our algorithm and its application through a marker-based Augmented Reality modeling system, named 3DARModeler [7]. In this system, by moving their hands in the 3D space, users will step by step add 3D geometries and put them together to create 3D models. A model is often constituted from a number of objects. As a user is holding an object, moving it around, the algorithm has to snap this object to existing parts and suggest suitable position to place it down. To snap a "moving object" (which is being hold and moved by a user) to existing parts, the algorithm first needs to be able to recognize the "active

978-0-7695-3704-7/09 $25.00 © 2009 IEEE
DOI 10.1109/ISUVR.2009.23

object" (which a user wish to snap to) of the model, and then determines a suitable position to snap to. From this point two terms "moving object" and "active object" will be used to describe our algorithm.

Recognizing the active object when the moving object is around can be classified to the "collision detection" category. Much effort has been put into this area. There are several algorithms to solve the problem [8]. Therefore this step is skipped to focus on the second step: determining a suitable position to snap the moving object to the active object.

We use another term "hot spot" in order to describe our algorithm. Each "hot spot" is a sphere with a certain radius. The details of the algorithm are as follows:

- ***Step 1:*** defining hot spots. For each type of geometry, some hot spots are defined with attributes: radius, center point's coordinate in a global coordinate system [9]. (We know both the geometry coordinate in a global coordinate system and the relative position of hot spots to the geometry. So we can calculate the center point's coordinate of a hot spot in the global coordinate system). In the 3DARModeller, six hot spots are defined for a geometry. For example, each cube has six hot spots locating at center points of each side with the radius is one-sixths of its length (Fig. 1). Certainly, these parameters could be customized to suit with each application's characteristic.
- ***Step 2:*** Updating hot spots' attributes. Because the algorithm has to work well even after a geometry has been transformed, coordinates of each hot spot need to be updated whenever there is a change made to the geometry. It is not a big problem with scaling, moving and, and resizing operations. However, with rotating operation, the normal vectors of each surface of the geometry should be particularly taken into account. In 3D geometry, the order of rotating an object around axes effects the final position, however in computer graphics, most of applications implement a certain sequence of rotating objects. Due to this, we can use transformation matrix [10] to update hot spots' coordinates. Because this step is done whenever the geometry is transformed, hot spots' coordinates are always available for comparing step. This improves the overall performance of the algorithm.

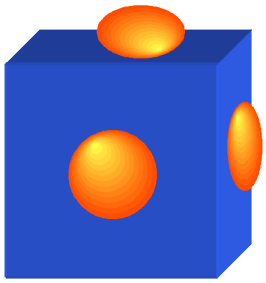

Figure 1. A cube with six hot spots.

- ***Step 3:*** Comparing distance. When the moving object is moving around the active object, distances from the center point of the moving object to the center point of each hot spot are calculated. This can be done by using the familiar distance formula in analytic geometry [11]. If a distance is smaller its hot spot's radius, that hot spot is where the moving object should be snapped to. Obviously, the performance of the algorithm only depends on the number of hot spots.

IV. EXPERIMENTAL RESULTS

Experimental results of the algorithm are illustrated through the 3DARModeler. In order to create a model, users first attach a geometry to a marker. Then they can customize attributes of this object such as size, position, orientation, color, and texture. After an appropriate position is located, this object can be placed to a main cardboard. Step by step other geometries are added to constitute a complete 3D model. Whenever the geometry on the marker (the "moving object") is close to any existing part of the model, the system will give a suitable position suggestion based on the snapping algorithm. If the users decide to add the moving object to the suggested position, it is automatically snapped with the existing part. And the suggestion position should be correct even after the existing parts are translated, rotated, and scaled. This feature is very important and useful in helping quickly create precise models. The results show that even with more than 50 objects, our snapping algorithm does not affect the scene rendering. Human movements are still smooth and natural. Fig. 2 shows the results of our snapping algorithm in the 3DARModeler system. The yellow box is the result of rotating and scaling operations from a cube. Fig. 2.a, b, c show snapping positions of moving object to the yellow box (wire frames next to/ above it) and Fig. 2.d, e, f display geometries after having been placed at the snapping locations.

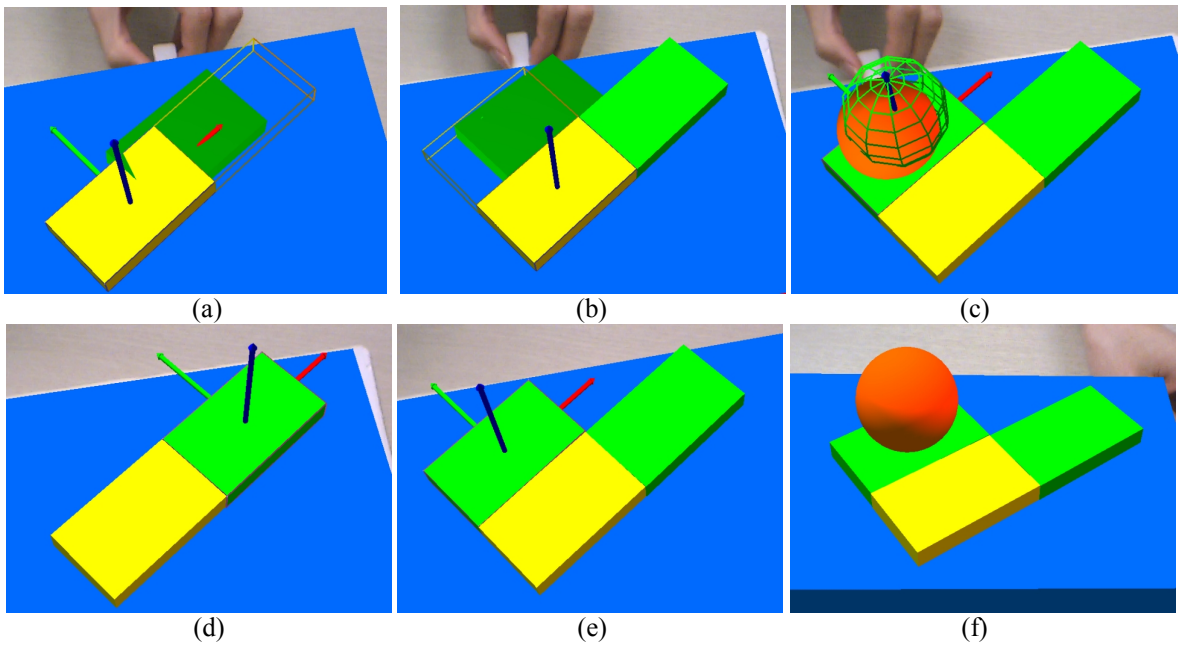

Figure 2. Snapping and suggesting suitable positions

V. Discussions and Conclusions

In this paper we describe a quick algorithm for snapping 3D objects in real time mode. It was proved to be fast enough not to affect the rendering process through a 3D modeling system in Augmented Reality environment. It is also flexible to adjust to other systems. Because the algorithm mainly deals with hot spots, its runtime behavior is independent on geometry type. We plan to improve the algorithm to apply to more complex 3D models.

In the current implementation, to gain the best performance, we only compare the center point of the moving object with hot spots. This implementation is not very intuitive especially when its size is big. Therefore, a posed question is should we also define hot spots for the moving object and then the algorithm turns to dealing with only hot pots. This would reduce the performance of the algorithm. So we should trade off between the performance and intuition for each system. The questions such as how many hot spots for each geometry are suitable, where to put these hot spots and what are their radius should be carefully considered when apply the algorithm for a specific case.

Acknowledgment

This work was sponsored and funded by Seoul R&BD Program (20070387).

References

[1] G. Lee, M. Billinghurst, and G. Kim, "Occlusion based Interaction Methods for Tangible Augmented Reality Environments", VRCAI 2004, Singapore, pp. 419-426, 15-18 Jun 2004.

[2] H. Kaufmann, "Construct3D: An Augmented Reality Application for Mathematics and Geometry Education", ACM Multimedia Conference 2002, DOI= http://doi.acm.org/10.1145/641140.

[3] Autodesk – AutoCad http://usa.autodesk.com/adsk/servlet/index?siteID=123112&id=2704278

[4] Autodesk – 3Ds Max http://usa.autodesk.com/adsk/servlet/index?siteID=123112&id=5659302

[5] Autodesk – Maya http://usa.autodesk.com/adsk/servlet/index?id=7635018&siteID=123112

[6] A. Stork, "An Algorithm for Fast 3D Picking and Snapping using a 3D Input Device and 3D Cursor", Source CAD Tools and Algorithms for Product Design, Springer-Verlag London, UK, 2000, pp. 113 – 127.

[7] T.V. Do, J.W. Lee, "3DARModeler: a 3D Modeling System in Augmented Reality Environment", International Journal of Computer Systems Science and Engineering, Volume 4 Number 2, 2008, pp. 145-154.

[8] P. Jiménez, F. Thomas, and C. Torras, "3D collision detection: a survey", Computers & Graphics, Volume 25, Issue 2, April 2001, pp. 269-285.

[9] J. D. Foley et al, "Computer graphics: principles and practice", Addison-Wesley, 1995, ISBN 0201848406, 9780201848403, pp. 222

[10] Homogenous Coordinates, http://bishopw.loni.ucla.edu/AIR5/homogenous.html.

[11] Wikipedia - Distance formula, http://en.wikipedia.org/wiki/Distance