Reality-Based Object Movement Techniques for 3D

Wolfgang Stuerzlinger, Darius Dadgari
York University
Toronto, Canada
http://www.cs.yorku.ca/{~wolfgang|~dariusd}

Ji-Young Oh
University of Arizona
Tucson, AZ, USA
http://3dvis.optics.arizona.edu

ABSTRACT
Scene layout and part assembly are basic tasks in 3D object manipulation. While movement methods based on 3D or 6D input devices exist, the most efficient 3D movement techniques are based on utilizing only two degrees of freedom. This poses the problem of mapping the motion of the 2D input device to efficient and predictable object motion in 3D. We present a new method to map 2D input to 3D motion in this paper. The object position follows the mouse cursor position closely, while the object always stays in contact with other surfaces in the scene. In contrast to existing techniques, the movement surface and the relative object position is determined using the whole *area* of overlap of the moving object with the static scene. The resulting object movement is visually smooth and predictable, while avoiding undesirable collisions. The technique also utilizes the fact that people easily recognize the depth-order of shapes based on occlusions. The proposed technique runs in real-time. Finally, the evaluation of the new technique with a user study shows that it compares very favorably to conventional techniques.

INTRODUCTION
Moving objects is one of the most basic tasks of 3D scene construction. When people design a scene with multiple objects, they repeatedly realign or adjust different parts, to explore the design space. Our goal is to provide an efficient and smooth object motion technique aimed at facilitating this explorative process in 3D manipulation systems such as Computer Aided Design (CAD), Virtual Reality (VR), and Augmented Reality (AR) systems.

In general, object motion is performed either with a mouse in CAD, with desktop-VR/AR systems, or with 3D or 6D input devices in VR and AR systems. We briefly discuss previous work by reviewing related work with input devices with two DOF as well as input devices with more DOFs.

Related Work – 2D Input Devices
Strauss categorized possible solutions to the problem of mapping 2D input to 3D movement [18]. He enumerates the idea of using handles, moving objects parallel to the view plane, using obvious structures to determine the plane of motion, and heuristics. The authors also claim that there is no "perfect" solution, as there is no approach that is both easy-to-use and robust at the same time. Consequently, users need to frequently check if the object is the desired position, which can become tedious.

Many commercial CAD systems utilize some form of handles to provide for 3D object motion. While this solution allows no room for failure or unexpected results, the task of moving an object may become tedious, as the user has to mentally separate the desired 3D movement into 1D or 2D components. Moreover, if objects are in contact, these objects may occlude handles and may make it difficult or even impossible to manipulate an object. The second category of movement techniques, moving objects parallel to the view plane, is rarely employed as the results of object movement depends strongly on the current view direction, which is undesirable. The third approach, which utilizes other structures in the scene, typically uses a ray from the eye point through the current pixel to identify the first intersection point with the scene. This intersection is then used to compute the position of the 3D object. E.g. Bier [1] used this approach in his snap-dragging technique, which snaps to the closest visual feature in a wire-frame display. However, this approach suffers from severe problems in complex scenes. As an example for a heuristic approach we list the idea of using a library of predefined objects with predefined movement behaviors. These behaviors are then used to constrain objects to particular places in a scene (such as an "on-wall" constraint). A ray along the current mouse position is then used to find the places in the scene where the constraints are fulfilled and the object is close to the cursor position. In the work on the MIVE system [17], we evaluated an implementation of this idea with several user studies and showed that such a technique allows naïve users to quickly populate 3D scenes with predefined objects.

Related Work – 3D Input Devices
A number of authors have investigated the performance of object manipulation with 3D input devices, such as a spaceball or a six degree-of-freedom tracker. Such devices enable direct interaction with a 3D scene and are typically used in VR/AR systems. One of the first researchers to present a system was Bolt in 1980 [2]. Subsequently many other researchers studied the creation and manipulation of 3D environments in VR. Bowman [3], Mine [8], and Pierce [13] proposed different 3D manipulation methods that can be applied in a variety of settings. For a more complete overview over work in this area, we refer the reader to a recent book about 3D user interfaces [4].

Several of these systems use collision detection to prevent interpenetration of objects. However, few utilize constraints for object manipulation and these support only the simplest

geometric constraints (e.g. "on-plane"). A general constraint-based system was presented by Multigen-Paradigm in the 'SmartScene' technology [16], which provides for arbitrary scripting of object behavior. Users can interact with the scene using tracked pinch-gloves, and the predefined object behaviors facilitate 3D object manipulation.

Observations about Real-World Object Manipulation

The following observations are based on data collected in a series of user studies on 3D object movement techniques targeted at naïve users. In these studies (see e.g. [10, 12, 17]) we experimented with many different alternative strategies, including the use of 2D, 3D, and 6D input devices as well as many forms of widgets to move objects in a 3D scene. Based on the results of these studies, we believe that the following set of rules captures the most important design decisions for 3D object movement techniques.

1. In the real world, (almost) all objects are attached or connected to other objects.
2. Objects do not interpenetrate each other.
3. Bringing objects in contact with each other is the natural way to position them relative to each other.
4. Only visible objects can be manipulated.
5. The most important cue for judging 3D position in real scenes is occlusion.
6. Many standard computer graphics techniques such as "handles", "wireframe", "3 orthogonal views", etc. tend to confuse users.
7. In general, 3D or 6D input devices provide less precision than 2D input devices.
8. Users seem to consider the entire area of visual overlap of the (moving) foreground object with the (static) background scene when deciding where the object is.

Please note that the above observations are based on experiments with naïve users, i.e. users who have no 3D computer graphics education or significant gaming experience.

In the following, we briefly discuss why each of the above points is important when dealing with naïve users.

1. Floating objects are exceptional in the real world and our observations in the user studies confirm that most users are surprised when an object starts to float when moved. That means that the correct *default* for any manipulation technique for 3D object motion is that objects should stay in contact with the rest of the world!
2. Most naïve users get confused if objects interpenetrate each other. This is easily solved with collision detection, something that can be computed nowadays in real-time even for complex scenes [5, 7].
3. The paradigm of sliding an object on the surface of another until it reaches the desired position is the most natural way to position objects – and in fact makes any task much easier. This is easily demonstrated by watching a child position toy blocks. The technical way to implement this is to choose a movement surface from the set of surfaces of the static scene and to displace the object on that surface. Often this is realized via the definition of constraints to object movement, see the section on previous work.
4. Naïve users don't try to manipulate objects that are not visible. They typically rotate the view/scene so that the part in question is visible. One indication for this is that a comparison of different interaction methods for 3D input devices found that the best techniques are based on the notion of ray casting [14]. Ray casting identifies the first object that is visible along an infinite ray from the manipulation device into the scene (much like a laser pointer). Hence, it is sufficient to allow the user to select all objects from a 2D image! And indeed, in [14], the authors hypothesize that all ray casting techniques can be approximated as 2D techniques.
5. As documented by research into visual perception, people judge 3D position based on several cues. Besides perspective, the most important cue for 3D position is occlusion [21]. In our studies, we found that for scenes without floating objects (see above), perspective and occlusion combined with the ability to quickly move the viewpoint are usually sufficient to allow humans to understand the 3D position of an object in relation to other objects. Finally, it is interesting to note that recent research confirmed that from an end-users point of view, most stereo technologies are not very mature and are tiresome and/or problematic to use on a daily basis (e.g. [6, 20]).
6. The idea that one has to use "handles" to move an object in 2D is an instance of an indirect manipulation technique. It is sufficient to point out that in the domain of (2D) desktop environments, this idea was very rapidly eclipsed by the idea of direct manipulation [15], as this paradigm proved to be much simpler to understand. Similarly, most naïve users can't readily interpret a wire frame view or three simultaneous views.
7. A human hand held in free space will "jitter" more than a hand that is supported by a real surface. That means that an input device that is limited to 2DOF provides more precision and hence usually affords also more efficient manipulation. In VR/AR research, this has been already realized through the adoption of techniques such as the Personal Interaction Panel [19], which effectively transforms a 3DOF input device into a 2DOF input device.
8. Practically all techniques for 3D object motion use the current position of the pointing device to compute the (new) 3D position. This effectively reduces the computation to a point mapping problem, and all current 3D object motion techniques are based on this idea. However, research into vision in primates discovered that

the perceptive field for an object that is being held in the hand covers the whole object [9]. In other words, there is strong evidence that the whole visual area of an object is used to judge 3D position. And indeed, we observed in our user studies that point-based techniques do not work as well as area-based techniques.

The presented list is based on our observations of naïve users. However, we would like to point out that while it may be possible that expert users can achieve higher performance by ignoring some of the above observations, we believe that for many kinds of routine scene modifications even expert users will greatly benefit from techniques that follow these guidelines.

REALITY-BASED 3D OBJECT MOVEMENT

Based on the above discussion, we designed a new 3D object movement technique that fulfills these criteria. Instead of using widgets, we allow the user to simply "grab" any object and slide it across the scene by dragging it to the desired position to visually assess the impact of a change. One of the main ideas behind this is that this will greatly facilitate exploration.

As the user moves an object, he/she utilizes his/her knowledge of the other surface(s) hidden by an area covered by the moving object. We implement this by always moving the object on one of the surfaces that it occludes. This implicitly guarantees that the object always stays attached to other objects. More precisely, we look for the closest visible surface behind the moving object and move the manipulated object onto it. Finding visible surfaces can be done very efficiently with graphics hardware, as can the detection of collisions [5, 7].

The attractiveness of the first alternative is its simplicity, as we only need to determine the foremost surface of the (static) scene, regardless of the current position of the moving object. Furthermore, this method ensures that the moving object is always closer to the viewer than the rest of the scene. This implicitly guarantees that there are no collisions with other objects. The downside of this method is that when a scene is cluttered with many objects, and there are consequently many surfaces, then the moving object will jump frequently in depth, and positioning the object requires more attention from the user. A pilot study of an implementation of this idea showed that this is indeed a problem that users encounter in practice.

The second alternative again identifies the first surface behind the moving object, but ignores any surface closer to the viewer than the moving object. In this method, the moving object does not immediately pop out to the surface in front of the user, unless the moving object becomes the one closest to the viewer. Usually, this conforms better to the intentions of the user. The limitation of this alternative is that when a small object moves forward, it may penetrate another object in front. To address this problem, we employ a collision detection method. Once a collision is found, the object jumps also in front of the colliding object, as with the first alternative.

Figure 1 depicts several movement sequences with the original mouse-ray techniques and the two new techniques mentioned in this section. Here the goal is to slide the chair under the table. Figure 1(a) shows the movement based on the foremost surface behind the mouse position. As soon as the mouse pointer overlaps with the surface of the table, the chair moves on top of it. However, when the mouse pointer moves off the table again, the chair drops immediately to the floor and ends up in a position where it collides with the table (fourth image). There are two ways to get the chair under the table: One is to change the viewpoint - Figure 1(c). The other alternative is to "grab" the chair by the top part of the backrest (an area that is visually quite small) and to move it towards the table while avoiding overlap with the table itself. However, this is very non-intuitive, and most users do not realize that this is possible – especially since the position that needs to be "grabbed" is not at all obvious.

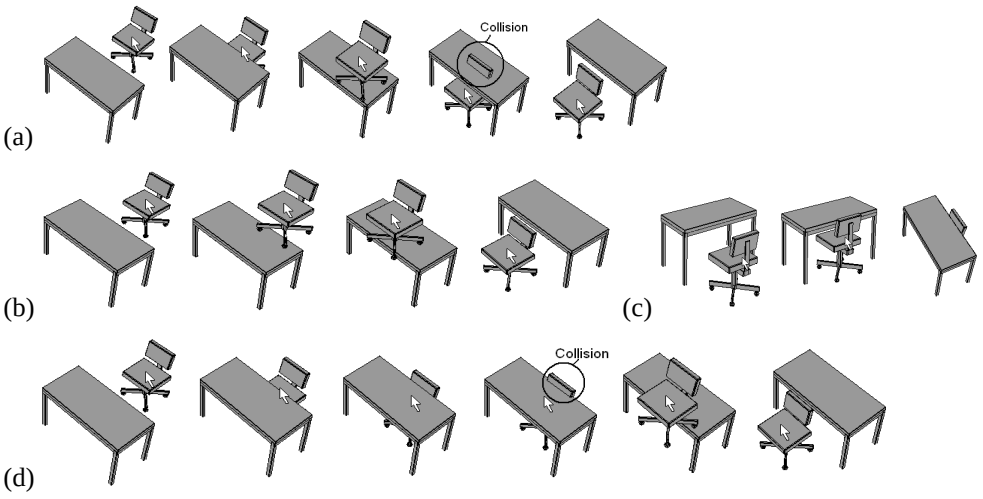

Figure 2. Image sequences illustrating object movement based on (a) the mouse position technique, (b) & (c) a technique based on the foremost visible surface, and (d) our new technique based on the foremost surface behind the moving object. For a detailed explanation please refer to the text.

The image sequence depicted in Figure 1(b) illustrates the technique that utilizes the foremost surface behind the image of the moving object. As soon as the image of the chair overlaps with the

table (second image), the chair starts to slide on the table surface. Only when the image of the chair does not overlap with the table anymore, does the chair drop down to the floor. Note that no collision occurs with this technique, but the only way to drag the chair under the table is again to change the viewpoint, as in Figure 1(c).

Finally, Figure 1(d) illustrates the new technique that utilizes the first surface behind the moving object itself. As the chair slides on the floor it continues to move underneath the table, because the first surface visible *behind the chair* is the floor. In the third image, the chair is clearly in the desired position and the user is finished. For illustration purposes, we continue this sequence with the fourth image from the left, where a collision occurs, which prompts the technique to move the chair onto the table.

Implementation

This algorithm can be efficiently implemented by using the features of modern graphics hardware. In particular, we can detect the foremost hidden surface by rendering the background scene. Rendering the moving object into a separate area and using the depth information computed by the hardware for both images aids in the identification of how much the object has to be moved in 3D. To avoid problems due to discretization, the implementation also uses an item buffer, which encodes where each object is visible on the screen, to compute a more precise answer. Further details of the implementation are described in [11]. Finally, our technique allows objects to move freely in 3D space, when an object is seen over the background (i.e. there is no visual overlap with other objects). In this case, the movement surface is chosen to be the axis-aligned plane that is most orthogonal to the viewing direction. This provides users with the option to quickly create "floating" objects, if necessary.

Evaluation

The presented technique works well for general shapes, even for objects that have large concavities or curved surfaces. For curved surfaces, the API's of current graphics cards necessitates an approximation of the curved surface into many small planar surfaces, which allows our algorithm to work without issue.

We evaluated the described scheme in several user studies that asked the participants to assemble various objects with a mouse in a desktop 3D modeling system [12]. We found that users had little problem in understanding how the new technique works, were significantly faster with it, and were quickly able to utilize the technique to its full potential.

CONCLUSION AND FUTURE WORK

Based on the results of a series of users studies, we presented a list of guidelines for techniques to move objects in 3D scenes. Then we presented a new reality-based technique to move objects in 3D scenes that provides intuitive 3D object motion, which naïve users found easy to understand and easy to utilize effectively. As this technique uses graphics hardware, it can handle even complex scenes efficiently. In the near future, we plan to evaluate this technique in IVY, our six-sided fully immersive VR system as well as other VR/AR setups.

REFERENCES

1. E. Bier, Snap-dragging in three dimensions. SIGGRAPH 1990, 193-204.
2. R. Bolt, Put-that-there, *SIGGRAPH 1980*, 262-270.
3. D. Bowman, et. al, An evaluation of techniques for grabbing and manipulating remote objects in immersive virtual environments. *Symp. on Interactive 3D Graphics*, 1997, 35-38.
4. D. Bowman, E. Kruijff, J. LaViola, I. Poupyrev, *3D User Interfaces: Theory and Practice*, Addison-Wesley, 2004.
5. K. Dave, K. P. Dinesh, CInDeR: Collision and Interference Detection in Real-time using graphics hardware, Graphics Interface, 2003.
6. D. Diner, D. Fender, Human Engineering in Stereoscopic Viewing Devices. Plenum Press, 1993.
7. N. K. Govindaraju, et al, CULLIDE: interactive collision detection between complex models in large environments using graphics hardware. SIGGRAPH Workshop on Graphics Hardware 2003, 25-32.
8. M. Mine, F. Brooks, C. Sequin, Moving Objects in Space: Exploiting proprioception in virtual environments SIGGRAPH 97, pp. 19-26.
9. S. Obayashi, et al., Functional brain mapping of monkey tool use, NeuroImage 14: 853-861, 2001.
10. J.-Y. Oh, W. Stuerzlinger, A system for desktop conceptual 3D design, Virtual Reality, 2004 7: 198-211.
11. J.-Y. Oh, W. Stuerzlinger, Moving objects with 2D input devices in CAD Systems and Desktop Virtual Environments, Graphics Interface 2005, 141-149.
12. J.-Y. Oh, Desktop 3D conceptual design systems, Ph.D Thesis, 2005. York University.
13. J. Pierce, *et al.*, Image plane interaction techniques in 3D immersive environments. *Symp. on Interactive 3D Graphics*. 1997. 39-43.
14. I. Poupyrev, et al., Egocentric object manipulation in virtual environments: empirical evaluation of interaction techniques. Eurographics 1998.
15. B. Shneiderman, Direct manipulation: A step beyond programming languages, *IEEE Computer 16*, 8, 1983, 57-69.
16. SmartScene, promotional material, Multigen-Paradigm, 1999.
17. G. Smith, et. al., 3D Scene Manipulation with 2D Devices and Constraints, Graphics Interface 2001, 135-142.
18. P. S. Strauss, et al., The design and implementation of direct manipulation in 3D. SIGGRAPH 2002 Course Notes, 2002.
19. Z. Szalavári, M. Gervautz, The Personal Interaction Panel, A Two-Handed Interface for Augmented Reality, Eurographics 1997, 335-346.
20. Z. Wartell, L. F. Hodges, W. Ribarsky, A geometric comparison of algorithms for fusion control in stereoscopic HTDs, IEEE TVCG, 8, 129-143, 2002.
21. C. Wickins, J. Hollands, Spatial displays, in Engineering psychology and human performance, Prentice-Hall, 1999.