Selecting and Sliding Hidden Objects in 3D Desktop Environments

Junwei Sun*
School of Interactive Arts & Technology
Simon Fraser University, Vancouver, Canada

Wolfgang Stuerzlinger†
School of Interactive Arts & Technology
Simon Fraser University, Vancouver, Canada

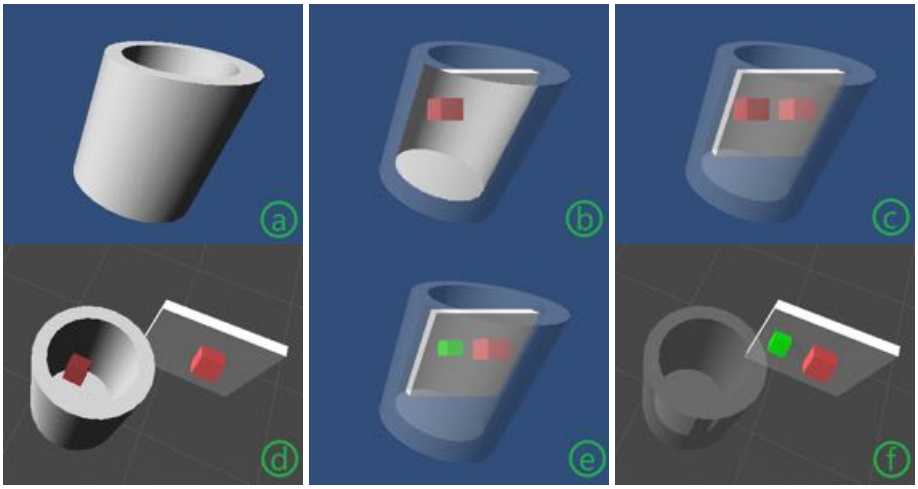


Figure 1: (a) A cup with standard opaque rendering. (b) The first layer rendered semi-transparent, revealing the red cube inside the cup. (c) The first two layers (the entire cup) rendered semi-transparent, revealing a second red cube behind the cup. (d) Top view corresponding to object positions in (a), (b), and (c). (e) The cube inside the cup moved to the surface behind the cup, highlighted in green. (f) Top view corresponding to object positions in (e).

ABSTRACT

Selecting and positioning objects in 3D space are fundamental tasks in 3D user interfaces. We present two new techniques to improve 3D selection and positioning. We first augment 3D user interfaces with a new technique that enables users to select objects that are hidden from the current viewpoint. This layer-based technique for selecting hidden objects works for arbitrary objects and scenes. We then also extend a mouse-based sliding technique to work even if the manipulated object is hidden behind other objects, by making the manipulated object always fully visible through a transparency mask during drag-and-drop positioning. Our user study shows that with the new techniques, users can easily select hidden objects and that sliding with transparency performs faster than the common 3D widgets technique.

Keywords: 3D positioning; selection; 3D interaction; transparency.

Index Terms: H.5.2. Information interfaces and presentation (e.g., HCI): User Interfaces - Graphical user interfaces (GUI)

1 INTRODUCTION

Selecting and posing a 3D rigid object, i.e., manipulating the position and orientation of an object, is a basic task in 3D user interfaces. Such tasks can be time-consuming, because 3D selection can require control of 3 degrees of freedom (DOFs), and full manipulation can involve 6 DOFs: Three DOFs for translation along three axes and another three for rotation around three axes.

In a virtual environment, the direct selection of 3D objects limits the user to the objects that are within their reach. Unless wireframe visualization is used, selection techniques typically limit selection to all visible objects [2], which requires only 2D input. In such systems, the only option to select hidden objects is to move the camera so that the desired object becomes visible. Another option, which is frequently used in computer-aided design systems, is to use multiple views, but (with hidden surfaces) even then there are situations where an object might not be visible in any of the views, e.g., if the object is contained in another one. Here, we present a new layer-based technique that enables users to select any occluded object, even if it is placed within non-convex parts of a scene.

Many manipulation techniques rely on 3 or 6 DOF input devices, based on a one-to-one mapping of input and object movement. Research has shown that 3DOF input devices outperform 2D devices in some contexts. Yet, people are more familiar with the form and function of a mouse [5].

For 3D manipulation of objects with 2D devices, it is challenging to provide efficient mappings between the 2D input and the 3D object movement. However, there is evidence that 2D input devices can outperform 3D devices for certain 3D manipulation tasks, through a smart mapping of user input to intuitive 3D object movement [3]. One such approach is constraint-based positioning, where the position of the manipulated object is constrained by the attributes of other objects of the scene.

* junweis@sfu.ca
† w.s@sfu.ca

Sliding is a prime example of a constraint-based 3D positioning technique [22]. Here, the object follows the mouse cursor and slides on the surfaces behind it. The object movement is then defined by the surface the object is in contact with. More advanced sliding methods perform better than the commonly used 3D widgets [24], even for objects that are not in contact with other surfaces. Yet, sliding assumes that the manipulated object is always (at least partially) visible. If an object becomes hidden, it is brought to the front, so that the users can see it. Here we address this limitation of the original sliding method through semi-transparency, by making the manipulated object always visible, even if the object is behind other ones.

We performed a user study to evaluate our new selection and sliding techniques. For this we used a task that requires the user to select a 3D object (initially) hidden in the main perspective view and position it into an (initially) hidden target position. We hypothesized that both our new layer-based selection and transparent sliding technique would perform better than 3D widgets.

Below, we first review relevant object selection and manipulation methods. Then we discuss original sliding, our new extension and describe our user study. Finally, we discuss the results and mention potential future work.

1.1 Contributions

The main contributions we present here are:

(1) A new layer-based technique to select hidden objects in non-convex scenes. This technique reveals the scene layer by layer, which enables the user to select an object on any desired scene layer.

(2) A new technique to show the manipulated object during sliding, regardless if it is directly visible or behind other parts of the scene.

2 Related Work

There has been substantial research in the field of 3D manipulation. Overall, the choice of the "best" input device depends on the task and the hardware platform. Some 3/6DOF devices perform better than the mouse on specific tasks, e.g., the Control Action Table [16], the GlobeFish, and GlobeMouse [14]. However, the mouse is generally more efficient than 3/6DOF devices for accurate placement, despite the lack of a third DOF [3][26].

Mouse-based 3D selection and manipulation is not without its limitations. First, one can only select visible objects with a mouse. Second, simultaneous translations along all three directions are not possible, due to the 2D nature of the device. One way to compensate is through smart mappings that use constraints, either explicitly specified by the user in their interaction, or implicitly specified through the content of the scene. Third, 3D rotations are not efficiently supported. Most 2D based interaction techniques limit the rotations to one axis (or at most two axes) at a given time.

Transparency has been widely used in 2D and 3D user interfaces. Zhai et al. [28] surveyed the use of semi-transparency in 3D interaction. Their study showed that semi-transparency acted as an effective depth cue for 3D target acquisition. Bier et al. [4] introduced a 2D transparent lens. Harrison et al. [17] proposed to use semi-transparent user interface objects in 2D interfaces. Harrison et al. [18] studied how to minimize interference between transparent layers. Gutwin et al. [15] studied the effects of dynamic transparency on targeting performance. Ishak et al. [19] introduced a content-aware transparency mechanism that dynamically adapts opacity depending on the importance of various parts of a window.

There has been substantial research on employing transparency in the visualization of 3D scenes, often to support navigation. Chittaro et al. [7] studied if semi-transparency is useful for 3D navigation in virtual environments. They found some positive effects on user navigation performance. Diepstraten et al. [10] introduced a view-dependent transparency model. Coffin et al. [8] presented cut-away techniques that permit a user to look "through" occluding objects, by interactively cutting holes into the occluding geometry. Elmqvist et al. [11] proposed an image-space algorithm to achieve dynamic transparency for managing occlusion of important target objects in 3D. Their techniques yielded significantly more efficient performance in 3D navigation tasks.

Various techniques have been proposed to improve the visualization of overlapping objects. LayerFish [27], supported layering and manipulating overlapping content in a 2D design space on desktop surfaces. To reduce demands on effort and attention, it used the fisheye technique to render an in-place scene index. Ramos et al. [23] presented two techniques for 2D layering operations. The first provided a graphical representation of a cascaded stack of layers above the selected elements. The second used a 'splatter' effect to radially distribute overlapping elements. Davidson et al. [9] proposed a depth sorting technique that extended standard 2D manipulation techniques and combined the layering operation with a page-folding metaphor for more fluid interaction in applications requiring 2D sorting and layout. Luboschik et al. [21] proposed a weaving technique, which offered a new and effective alternative for picking any object from a set of overlapping objects, without using transparency. Javed et al. [20] presented a novel approach to manage occlusion between physical items resting on tabletop displays and virtual objects projected on the display.

Agustina et al. proposed XPointer [1], an X-ray telepointer technique for collaborative 3D selection, which enables users to select initially hidden objects. Their selection technique is object-based, as users can specify which of the objects intersected by ray-casting should be selected. The 3D editing operations for XPointer are standard widget-based editing techniques, inherited from Autodesk Maya. Their X-ray manipulator [1] is a technique for adjusting the XPointer's penetration depth, but this method only works correctly in scenes with convex objects. When the selection ray hits a concave object, that entire object is made semi-transparent by XPointer. Thus, all objects within or occluded by the concave object are then visible simultaneously. Relative to revealing the scene layer by layer, this decreases the number of depth cues available for object selection (and later manipulation). The existence of fewer depth cues also introduces an ambiguity in the selection process, which can even affect the subsequent positioning phase, as the user may not be able to perceive correctly where the manipulated object is in space.

Oh et al. [22] presented a sliding algorithm, where the object follows the cursor position directly and slides on any surface behind it, i.e., the moving object always stays attached to other objects. Sun et al. [25] improved Oh et al.'s work through Shift-Sliding and Depth-Pop, two techniques that significantly speed up common 3D positioning tasks, including tasks that require the object to float in mid-air. However, all such sliding algorithms build on a visibility assumption in two ways. First, users can only select objects that are at least partially visible, and second the object that the user manipulates must always be at least partially visible to the user. If an object becomes hidden, these sliding techniques (have to) bring the object to the front, so that users can see it. Similarly, Depth-Pop enables the user to place the object along any position along the mouse ray, as long as it is at least partially visible.

3 Selecting And Sliding Hidden Objects

Here, we first present our new method to select hidden objects. Then we detail our new technique to facilitate the sliding of objects even if they are hidden.

3.1 Control-Depth Selection

Selection from a specific layer of the scene is necessary in many situations. Figure 2 left shows a scene where the red and blue cubes are positioned behind different layers in the scene. Layer-based selection is useful when there is a need to select an object behind a specific layer.

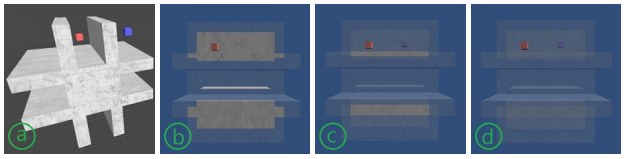


Figure 2: (a) A concave object with other objects in the cavities. (b) and (c) Seen from the side, the red and blue cubes are behind different layers of the object. Here, layer-based selection can reveal the desired object by making all content in front of it semi-transparent. (d) Object-based transparency makes the whole object transparent, which makes it harder to understand where the cubes are.

With all 2D input-based interaction methods that rely on a single view, users can only select visible objects. Hidden objects cannot be selected without moving the camera or transitioning to other view types, such as wireframe. In Figure 4 and with such techniques, if there is an object at position C, D or E in the scene, the object cannot be selected without camera navigation. To enable the user to select such a hidden object, we introduce our new "Control-Depth" selection technique, which involves the Ctrl-key and mouse wheel actions. We use the Ctrl-key to activate the depth selection mode. While users are holding the Ctrl-key pressed down (without having clicked any mouse buttons), they can push the mouse wheel forward to reveal the first, previously hidden, perspective depth layer of the scene, making the initially visible surfaces in front of them semi-transparent. Every additional mouse wheel push reveals the next layer behind the previous one. If the user pulls the mouse wheel backwards, i.e., towards them, we transition the next closest semi-transparent layer back to opaque. An occluded object will then become visible after all layers in front of it are made semi-transparent. This permits the user to simply "scroll" among all visible layers. The users can then simply select the desired object by clicking "through" the transparent layers of the scene. In comparison to XPointer [1], our new technique enables us to better deal with scenarios with concave scene objects. We use depth peeling [13] to identify the visible layers for Control-Depth selection.

Figure 3 and 4 show the same 3D scene with a concave object, with Figure 4 showing a side view. The position of the red cube in Figure 3 corresponds to position D in Figure 4. XPointer would make the whole base object semi-transparent, as illustrated in Figure 3(d), which makes it difficult to judge the position of the red cube, unless the user has the strong prior knowledge of the scene. Figure 1 shows an example where XPointer's design decision to make whole objects transparent introduces some ambiguity as to where objects are located. The first red cube is inside the cup and the other one is behind the cup. Their sizes are similar in the perspective view. If the whole cup is made semi-transparent, it is hard for the average user to distinguish which of the two red cubes is inside the cup.

The idea of using the mouse wheel to select objects that are behind other objects has previously been presented in COMSOL Multiphysics[1], which relies on a rendering mode that shows *all* objects *simultaneously* as semi-transparent. Yet, research into volume rendering methods, which are all based on semi-transparency, has shown that even the best volume rendering methods result in 25% error in terms of depth perception [12]. Standard volume rendering, which renders everything semi-transparent from back to front, performs even worse. Based on this result, we decided to pursue an approach that renders everything in front of the layer that the user is currently focusing on semi-transparent, and everything behind that opaque, as this makes it easier to understand the geometric context. Moreover, we enable the user to directly control which layer is shown, which makes it easier to understand the geometric relationships inside the scene.

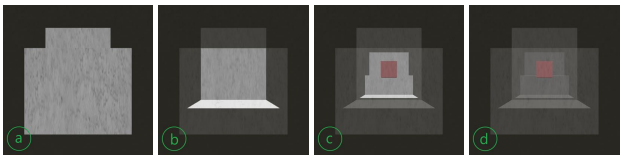


Figure 3: (a) A scene with standard opaque rendering. (b) The first layer rendered semi-transparent. (c) The first two layers rendered semi-transparent, revealing the red cube. (d) The entire object rendered semi-transparent, which corresponds to the visualization used by XPointer. It is hard to see if the cube is in front of the small ledge or behind it. In reality, the cube is in front of the ledge at position D, see Figure 4.

Figure 3 illustrates the process of Control-Depth selection. Figure 3(a) shows the original view of the scene. In Figure 3(b), the first layer of the scene was made semi-transparent by moving the mouse wheel forward once, while holding the Ctrl-key down. In Figure 3(c), the second layer was made semi-transparent, revealing the red cube. The user can then simply select that cube by moving the mouse cursor over it and selecting it through a click. For selection, we select the first non-transparent object below the cursor.

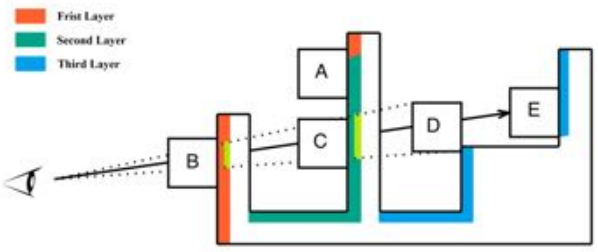


Figure 4: Illustration of Control-Depth selection and transparency sliding. Objects C, D, and E are fully occluded, but can be revealed with Control-Depth selection (and then selected). The first, second, and third (front-facing) layer of the concave object is shown in orange, green, and blue, respectively. All layers can be viewed by transparency sliding, which makes surfaces before the object semi-transparent, e.g., the two areas in light green when the object is sliding at position D.

3.2 Transparency Sliding

We base our design for our new transparency sliding method on a basic sliding algorithm [22]. By enhancing sliding through semi-

[1] https://www.comsol.com/

transparency, we make it possible to keep objects visible during sliding. In basic sliding, the manipulated object moves along the surface behind it that it is in contact with. During such sliding, the object is always (at least partially) visible to the user. Users can then only select and manipulate visible objects, such as objects at A or B in Figure 4. They could not move an object to position C, D or E, as it would be fully occluded.

For our new transparency sliding method, we use transparency to make the manipulated object always fully visible during manipulation. When the user slides an object to a position where it is partially (or fully) hidden, we make the parts of the scene in front of the manipulated object semi-transparent. In Figure 4, the object is fully occluded at position C, D, or E. To enable the user to manipulate the object at these positions, we make the parts that are occluding the object semi-transparent. Thus, the two areas in light green are made semi-transparent so the users see and manipulate the whole object when it is at position D. With Depth-Pop [25], users can then place the object at positions B, C, D, or E, while still being able to see it due to the transparency mask (for C, D, and E).

Figure 5: Transparency masks around a manipulated object.

Figure 5 shows the transparency masks around a manipulated object. With it, the object is always fully visible during sliding. Figure 5 left shows a combination of Control-Depth transparency and transparency mask. Through Control-Depth transparency, the first layer of the scene is made semi-transparent. If users slide the cube behind the second and third layer of the scene, the scene in front of the object is made semi-transparent with the transparency masks.

3.3 Implementation

We exploit the computing power of GPUs and use the frame buffer for the computations. The alpha value of textures in the scene depends on the depth values of the manipulated object. If the manipulated object is farther away from the camera than an object in the scene, each occluder is shown semi-transparent at pixels on or around the manipulated object. The border size around the object (in screen space) is predefined but can be easily adapted to different scenarios. We set the border size to 3 pixels in our experiments.

To implement our new transparency mask, we use an algorithm that is an extension of the one used for Control-Depth selection. We apply a surface shader on all objects in the scene. We pass the current scene depth texture and the depth texture of the manipulated object (if there is one) to the surface shader. With Control-Depth selection, we update the current scene depth texture with depth-peeling. Initially, the current scene depth texture is defined from the first layer of the scene, which represents the current visible layer. If the user pushes/pulls the mouse wheel forward/backward, the scene depth texture is set to the next, respectively previous layer. In the surface shader, when the depth value of the pixel on the current object is less than the current depth texture (i.e., closer to camera), that pixel's alpha value is set to a constant, currently 0.25. This is the core of the technical implementation of Control-Depth selection. To implemented the transparency mask, we set the alpha values of a pixel that are closer to the camera than the manipulated object to 0.25 and also spread that transparency value to the 3x3 surrounding pixels.

To enable the transition between different visibility layers during sliding, we use a generalized version of the Depth-Pop algorithm [25], which does not check if the object remains visible. As shown in Figure 5, the cube in the left image and the chair at the right are both fully hidden yet made visible by transparency masks. Users can use the generalized Depth-Pop technique to transition between different layers.

3.4 Combined Selection and Sliding

During prototyping, we identified that it is useful to enable Control-Depth even during sliding. Once users have selected an object, if they hold the Ctrl-key and left mouse button down at the same time, we perform both Control-Depth and Depth-Pop operations simultaneously, which reveals depth layers while positioning the object between them. However, note that even with this combination, the manipulated object can become partially hidden behind parts of the scene. In other words, even with the layer-based method there are situations when the transparency mask is needed to show the manipulated object and the context around it correctly.

Finally, we point out that our new transparency sliding technique is independent of the selection technique. In other words, it is possible to use the new sliding technique even if object selection is limited to visible objects or if object selection is performed through some other method. In this case, the manipulated object will then still always be visible. Conversely, it is possible to use only the new transparency sliding technique without enabling the selection of hidden objects.

4 User Study

We performed a user study to evaluate the performance of Control-Depth selection as well as transparency sliding. We initially considered a comparison with the X-Ray manipulator in XPointer. Yet, we chose to not to do this, as the XPointer technique [1] cannot handle the positioning of objects in scenes with concave parts appropriately, as discussed above. Also, XPointer requires a side view visualization for the cursor, which increases the time spent in the selection phase, as the user must move the cursor over larger distances (between windows). Moreover, Sun et al. [25] had already shown that sliding with Depth-Pop is more efficient and accurate than 3D widgets, for cases when the manipulated object is at least partially visible. Still, Depth-Pop cannot handle cases when the object is hidden.

We also considered a comparison between basic sliding with camera navigation and transparent sliding. When we performed a pilot study with novices, camera navigation turned out to be a major challenge, as they had no experience with 3D editing systems. While they could move the camera to a position where they could see the object, they then struggled to (slowly) find a viewpoint that shows the target position in the same view, which is the only option that enables users to perform the task with basic sliding. As our interaction techniques are targeted at novices, we decided to disable camera navigation in our experiment, as navigation would dominate the timings and thus pose a confound.

Based on these arguments, we limited our investigation to the performance of selection and precise positioning for hidden objects. Given that the XPointer technique and most 3D packages use 3D widgets for manipulation, we decided to compare our new technique with 3D widget-based manipulation. As our new user interface is designed to make the task of both selecting and

manipulating hidden objects faster, we hypothesized that we would get similar results as Sun et al. [25], i.e., transparency sliding would outperform 3D widget-based manipulation for hidden objects.

4.1 Participants

We recruited 12 (7 female) unpaid undergrad students from the local university population. We did not screen participants for 3D experience. Our participants had varying game expertise, with 42% playing games regularly.

4.2 Apparatus

We built our system in the Unity game engine. We used a desktop computer with 3.5 GHz i7 processor, 16 GB of memory, and two NVIDIA GeForce GTX 560 SLI graphics cards. We used a mouse and a keyboard as input devices.

4.3 Procedure

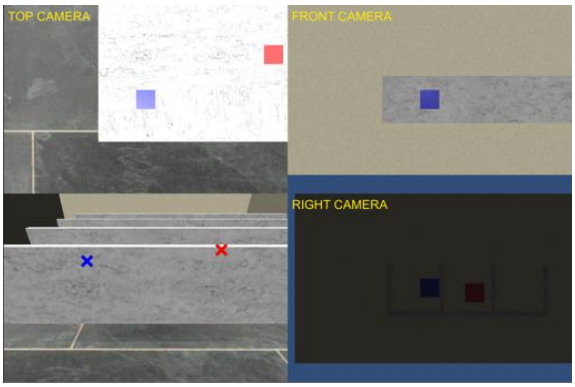
We designed a 3D object positioning experiment and asked participants to move an object from a hidden start position to a hidden target position in various scenes. We used two different levels of depth complexity for the trials. All target positions were in contact with other surfaces in the scene. In all conditions, the scene was shown in a 4-view display, with one perspective view and three orthogonal views. Figure 6 shows two sample tasks in the sliding condition. The object and target position are both hidden in the perspective view, yet the source and target object locations are marked with a red and blue marker respectively. The object is the red cube, whereas the target position is rendered as a semi-transparent blue cube, a 3D copy of the object. The horizontal distance on screen between object and target positions are roughly one half of screen width in the perspective view. With this and in the sliding condition, the users need to use Control-Depth selection to select and to move the object through sliding in the perspective view.

The geometry of the "building" in Figure 6 (b) consists of walls and the floor as a single, large, non-convex object. In this task scenario, the XPointer technique [1] makes the entire non-convex object transparent, which then makes it challenging (or impossible) to position an object at a location hidden by a wall onto the floor (or even another wall) behind that first wall. Our layer-based selection method works properly in this scene. In the 3D widgets condition, and since the object is hidden in perspective, the users had to select it in one of the orthogonal views. After the users click on the object, the 3D widgets appear on the object, and the 3D widgets can be used in all four views for manipulation. Having to use different views typically slows the selection process down, yet the only other option to select a hidden object would be to permit camera navigation, something that we wanted to avoid due to the results of our pilot study mentioned above.

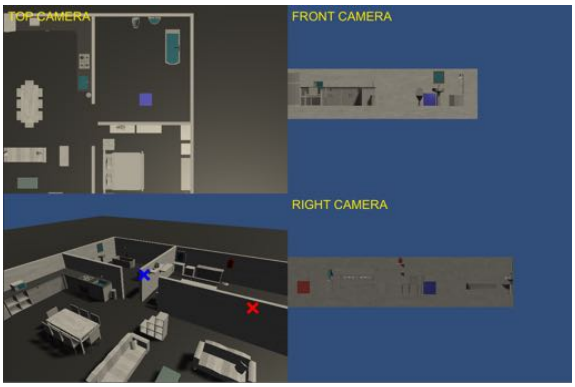
There was a 2-minute training session before each condition, which introduced participants to the techniques in a playground environment but did not include any version of the experimental tasks. We asked the participants to perform the tasks as quickly and as accurately as possible. After the participants finished all the tasks, we asked them to fill a questionnaire about the usability of the two techniques. We also asked them about potential improvements. In total, the study took about 30 minutes for each participant.

4.4 Experimental Design

The experiment compared transparency sliding and 3D widget-based positioning in a within-subjects design. Technique was the only independent variable. When the user had positioned the object at the target position, they needed to press the space bar for confirmation, and then went on to the next trial. The users were allowed to use their non-dominant hand to press the space bar. Each participant performed 20 trials, ten trials for sliding and ten for 3D widgets.



(a)



(b)

Figure 6: Two sample tasks in the sliding condition a) task scene with simple geometry and b) a task scene with richer geometry. The object and target positions are both hidden in the perspective view. They are marked by red and blue markers respectively. The target position is centered in the three orthogonal views. The object (red cube) is visible in at least one orthogonal view.

4.5 Data Generation

We measured the combined "visual search & selection" time, the completion time, and relative error distance from the ideal target position. Time was measured in seconds. For the "visual search & selection" time, the timing starts when the previous trial ends, and ends when the correct object is selected. For the completion time, the timing starts when a mouse button or a key is pressed, and the timing includes both the selection and positioning stages. The timer for completion time ends when the user releases the mouse button for the last time. The error measure was calculated as the absolute distance to the target (center to center) over the object size (length of any side of the cube). We recorded all actions of each user.

4.6 Results

We performed paired t-tests to compare the mean in "visual search & selection" time, the completion time, and relative error distance from the ideal target position for sliding and 3D widgets.

The average time for "visual search & selection" with the 3D widget-based method was 3.07 seconds, while Control-Depth selection took 4.65 s. Looking only at the time users took for the

selection (i.e., the time from pressing the Ctrl key until the correct object was selected, which includes the mouse movement to select the object) in the Control-Depth condition, it took users only 2.31 s.

We call the total of the selection and positioning time the completion time. This timer starts when the user pressed the mouse/keyboard for the first time. The results showed that sliding (M = 12.84s, SD = 9.82) is overall significantly faster than 3D widgets (M = 24.31s, SD = 12.74), $t(119) = 8.3192$, $p < .0001$ in terms of completion time. For the sliding condition, the above-mentioned average selection time of 2.31 seconds represents approximately a fifth of the positioning time. The selection time for the 3D widgets condition is zero, as timing started with the first mouse click. See Figure 7.

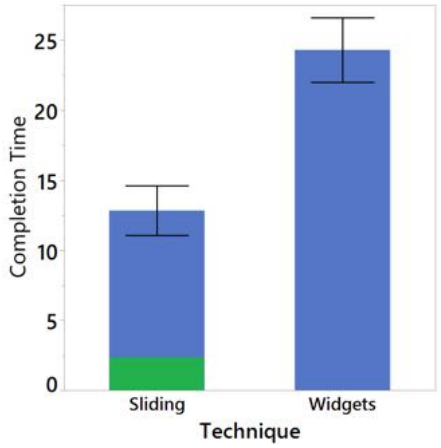


Figure 7: Average completion times (in seconds) for two techniques. Each error bar is constructed using a 95% confidence interval of the mean. The green area in the sliding condition represents the selection time of 2.31 seconds, whereas the corresponding time with 3D widgets is 0.

In terms of error measure, there was no significant difference between sliding (M = 0.130, SD = 0.291) and 3D widgets (M = 0.133, SD = 0.264), $t(119) = 0.09$, $p = 0.93$. See Figure 8.

Almost all, 11 out of 12, participants thought sliding was easy to use, while 10 participants thought 3D widgets were easy to use. Still, 10 out of 12 participants preferred sliding over 3D widgets.

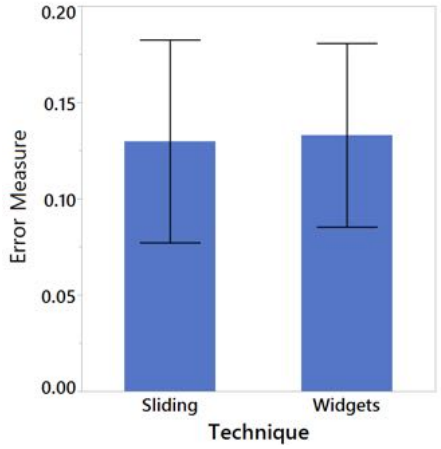


Figure 8: Average error measures for two techniques. Each error bar is constructed using a 95% confidence interval of the mean.

5 Discussion

We first compared only the "visual search & selection" time. The results showed that click-based selection used in the 3D widgets condition is 34% faster than Control-Depth selection. This is not surprising, as Control-Depth selection requires more actions from the users. However, the difference in time is only 1.58 s. According to Brown et al.'s results [6], an average mouse movement takes about 0.8 s. Therefore, we can approximate the visual search time as (3.07 - 0.8 =) 2.27 s for click-based selection, and (4.65 - 2.31 =) 2.34 s for Control-Depth selection, which are unlikely to be significantly different.

Although selection takes slightly more time with the Control-Depth technique, the results in terms of the total completion time support our hypothesis. Even though the timing for 3D widgets did not include the mouse movement time before selection, sliding was still faster than 3D widgets, even for hidden objects. Users' prior knowledge of the task scenes might have an impact on the performance, but none of the participants seemed to have been familiar with the scenes. Naturally, if users know which layer the object is on beforehand, they could perform better in the tasks. As we targeted our interaction design at novices, we only recruited novice users unfamiliar with 3D editing software. Obviously, experienced users might be able to complete the investigated tasks with a combination of basic sliding movements or 3D widget manipulation together with camera navigation. Yet, we believe that experienced users will still benefit from our new technique, as their stronger mental model of the geometry of a scene will help them know/remember where an object is located, even if it is not visible. This knowledge, together with Control-Depth selection, will permit them to avoid camera navigation altogether in many instances, which will make their workflow more efficient.

In our experiment, users seem to have had little issues with understanding the scene geometry in the sliding condition. They showed no signs of confusion with the Control-Depth selection, transparency masks, or their combination, even though they both introduced transparency to the scene. The two techniques did not conflict with each other in any of the task scenes, e.g., Figure 5 left. Additionally, the border of the transparency mask provided cues for users to understand which layer the manipulated object is at. It also helps that the Control-Depth selection is associated with discrete actions in the selection stage, while the transparency mask involves a continuous sliding action during the manipulation stage. From our observations during the experiment, we can also confirm that the techniques are easy to understand and learn and all users were able to use them after a short training session.

Most participants preferred sliding over 3D widgets in our task scenarios, which involved situations where the object is hidden at both the source and target locations. Some participants identified correctly that sliding required less mouse movement and thus rated it more positively. Also, with sliding, the object was always under the mouse cursor and fully visible, which led the participants to feel they had a better control over the object's position. Some of them even asked for the semi-transparency feature in the 3D widgets condition, as the object was usually hidden in the perspective view during object movement in that condition. We see this as motivation to include our semi-transparent techniques in standard 3D software. This would help 3D designers improve their efficiency during 3D modelling.

We use mouse wheel operations together with the Control modifier key (Ctrl). This is very similar to how 3D packages use mouse operations together with modifier keys, e.g., to move objects in three dimensions. After being exposed to our new technique once, all participants had no difficulties using it during the study. Depending on the application scenario, other activation

methods can be used, including through side buttons on a mouse (or controller) instead of the Ctrl key, which then also obviates the need for a keyboard. Mappings for touch screens, e.g., through the use of two-finger gestures, are equally possible.

In the basic sliding technique, an object is automatically popped to the front whenever occluded. Novice users might find this behaviour unexpected. The new semi-transparent sliding technique presented here does not share this automatic behaviour, which gives the user better control during interaction. If the users want to pop an object to a layer in front, they can explicitly use Depth-Pop during sliding. Our participants also commented positively on how the new interaction technique handles occlusion.

Unlike object-based selection techniques, our layer-based technique allows users to select an object from a specific layer. This avoids any potential ambiguity in perception of object depth, especially when there are concave objects in the scene. With our techniques, users can perform object selection and manipulation continuously within a single perspective view. Any object (visible or not) within the view frustum can be selected and positioned to any desired position that meets the sliding assumptions [25], without having to change the camera view. This makes our new techniques very general.

The transparency mask technique could trivially be combined with Shift-Sliding [25]. When the user lifts an object up into a floating state, i.e., without any contact to other surfaces, and then starts sliding parallel to the original contact plane, we could use transparency masks to make the manipulated object fully visible if the object becomes partially hidden. This generalizes transparency sliding to floating objects.

We used depth buffers to implement the Control-Depth selection method. We reveal one scene layer at a time with a mouse wheel action. Most computations are performed in the graphics hardware and for the scenes used in the experiment, it takes approximately at most an extra 4 ms to reveal each layer. This temporarily caused the frame rate to drop from 69 fps (14.5 ms per frame) to 55 fps (18.2 ms per frame). During sliding with transparency masks, the frame rate never dropped below 69 fps. During Depth-Pop, the frame rate dropped to as low as 47 fps (21.2 ms per frame), as it requires rendering more object/scene layers. Users did not seem to be affected by the small latency introduced by the techniques.

6 Limitations

Our technique is efficient, as users can quickly identify, select and slide the desired object. However, we anticipate that on lower-end graphics hardware the current implementation of Control-Depth selection could slow down in scenes with high depth complexity, as we focused on the interaction and did not realize all potential optimizations in the code. This can be addressed with a redesign of the computations.

One implementation problem we face is z-fighting, where two polygons have similar values in the z-buffer at the same pixel. In Control-Depth selection, we constantly compare the object depth with the scene depth. Insufficient precision in the depth buffer then introduces z-fighting, which makes it hard to identify layers. One option to reduce this problem is to adjust the length of the camera frustum for each specific scene.

For Control-Depth selection and transparency mask, we use a fixed alpha value with the standard transparent rendering approach to achieve a semi-transparent effect, where the target/manipulated object is shown with a blended colour. This effect generates different results for objects with different textures. Moreover, image quality potentially degrades with an increasing number of depth layers in the scene, which might make it more difficult for users to judge the manipulated object's position. We also considered using cutaways. Yet, using such visualization techniques would cause the object to slide on "ghost" surfaces, i.e., would involve sliding on surfaces that are then completely hidden, which likely would surprise novices. Finally, our current work focused on the interaction aspects. Thus, we consider optimized semi-transparency effects that maximize the visibility of one or more objects to be out of scope for our current work, but plan to revisit this in the future.

7 Conclusion And Future Work

For applications where users are positioning objects with a mouse in a 3D scene, we presented our new Control-Depth selection method, which enables users to select hidden objects in arbitrary scenes without resorting to (potentially time-consuming) camera navigation, by iteratively "peeling away" layers in front of a target object and then simply selecting it. We also proposed a new extension to the sliding technique, which uses a transparency mask to facilitate object sliding, so that the user can see the object even if it slides behind other objects.

We performed a user study to compare the performance of our new selection and transparency sliding techniques with common 3D manipulation widgets. Users had to select hidden objects and move and precisely position them at hidden target positions. The results showed that even though it took an extra step to complete the task in the sliding condition (selection and positioning), the combined time was still significantly faster than with common 3D widgets. Users found our techniques easy to learn and use.

In the future, we plan to investigate methods to optimize our implementation of Control-Depth selection and reduce the impact of z-fighting. We may also investigate how to optimize rendering of semi-transparency effects for different textures and geometries. Additionally, we will check if more complex scene geometry affects the users' perception of the scene when using our techniques.

In order to further evaluate the efficiency and simplicity of the techniques, we would also like to perform another experiment with both novices and participants that are familiar with desktop-based 3D interaction. There we would evaluate how much benefits our techniques have for experts.

References

[1] Agustina and C. Sun, 2013, February. Xpointer: an X-ray telepointer for relaxed-space-time Wysiwis and unconstrained collaborative 3D design systems. In *Proceedings of the 2013 Conference on Computer Supported Cooperative Work* (pp. 729-740). ACM.

[2] F. Argelaguet and C. Andujar, 2013. A survey of 3D object selection techniques for virtual environments. *Computers & Graphics*, *37*(3), pp.121-136.

[3] F. Bérard, J. Ip, M. Benovoy, D. El-Shimy, J.R. Blum and J.R. Cooperstock, 2009, August. Did "Minority Report" get it wrong? Superiority of the mouse over 3D input devices in a 3D placement task. In *IFIP Conference on Human-Computer Interaction* (pp. 400-414). Springer, Berlin, Heidelberg.

[4] E.A. Bier, M.C. Stone, K. Pier, W. Buxton and T.D. DeRose, 1993, September. Toolglasses and magic lenses: the see-through interface. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques* (pp. 73-80). ACM.

[5] D. Bowman, E. Kruijff, J.J. LaViola Jr and I. Poupyrev, 2004. *3D User Interfaces: Theory and Practice, CourseSmart eTextbook*. Addison-Wesley.

[6] M.A. Brown and W. Stuerzlinger, 2014, May. The performance of un-instrumented in-air pointing. In *Proceedings of Graphics Interface 2014* (pp. 59-66). Canadian Information Processing Society.

[7] L. Chittaro and I. Scagnetto, 2001, November. Is semitransparency useful for navigating virtual environments?. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology* (pp. 159-166). ACM.

[8] C. Coffin and T. Hollerer, 2006, March. Interactive perspective cut-away views for general 3D scenes. In *3D User Interfaces, 2006. 3DUI 2006. IEEE Symposium on* (pp. 25-28). IEEE.

[9] P.L. Davidson and J.Y. Han, 2008, October. Extending 2D object arrangement with pressure-sensitive layering cues. In *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology* (pp. 87-90). ACM.

[10] J. Diepstraten, D. Weiskopf and T. Ertl, 2002, September. Transparency in interactive technical illustrations. In *Computer Graphics Forum* (Vol. 21, No. 3, pp. 317-325). Blackwell Publishing, Inc.

[11] N. Elmqvist, U. Assarsson and P. Tsigas, 2007. Employing dynamic transparency for 3D occlusion management: Design issues and evaluation. *Human-Computer Interaction–INTERACT 2007*, pp.532-545.

[12] R. Englund, and T. Ropinski, 2016. Evaluating the perception of semi-transparent structures in direct volume rendering techniques. *In SIGGRAPH ASIA Symposium on Visualization* (page 9). ACM.

[13] C. Everitt, 2001. Interactive order-independent transparency. *White paper, nVIDIA*, *2*(6), p.7.

[14] B. Froehlich, J. Hochstrate, V. Skuk and A. Huckauf, 2006, April. The Globefish and the Globemouse: two new six degree of freedom input devices for graphics applications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 191-199). ACM.

[15] C. Gutwin, J. Dyck and C. Fedak, 2003. The effects of dynamic transparency on targeting performance. In *Graphics Interface* (pp. 105-112).

[16] M. Hachet, P. Guitton and P. Reuter, 2003, October. The CAT for efficient 2D and 3D interaction as an alternative to mouse adaptations. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology* (pp. 225-112). ACM.

[17] B.L. Harrison, H. Ishii, K.J. Vicente and W.A. Buxton, 1995, May. Transparent layered user interfaces: An evaluation of a display design to enhance focused and divided attention. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 317-324). ACM Press/Addison-Wesley Publishing.

[18] B.L. Harrison, G. Kurtenbach and K.J. Vicente, 1995, December. An experimental evaluation of transparent user interface tools and information content. In *Proceedings of the 8th Annual ACM Symposium on User Interface and Software Technology* (pp. 81-90). ACM.

[19] E.W. Ishak and S.K. Feiner, 2004, October. Interacting with hidden content using content-aware free-space transparency. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology* (pp. 189-192). ACM.

[20] W. Javed, K. Kim, S. Ghani and N. Elmqvist, 2011, September. Evaluating physical/virtual occlusion management techniques for horizontal displays. In *IFIP Conference on Human-Computer Interaction* (pp. 391-408). Springer, Berlin, Heidelberg.

[21] M. Luboschik, A. Radloff and H. Schumann, 2010, May. A new weaving technique for handling overlapping regions. In *Proceedings of the International Conference on Advanced Visual Interfaces* (pp. 25-32). ACM.

[22] J.Y. Oh and W. Stuerzlinger, 2005, May. Moving objects with 2D input devices in CAD systems and desktop virtual environments. In *Proceedings of Graphics Interface 2005* (pp. 195-202). Canadian Human-Computer Communications Society.

[23] G. Ramos, G. Robertson, M. Czerwinski, D. Tan, P. Baudisch, K. Hinckley and M. Agrawala, 2006, May. Tumble! Splat! helping users access and manipulate occluded content in 2D drawings. In *Proceedings of the Working Conference on Advanced Visual Interfaces* (pp. 428-435). ACM.

[24] P.S. Strauss and R. Carey, 1992, July. An object-oriented 3D graphics toolkit. In *ACM SIGGRAPH Computer Graphics* (Vol. 26, No. 2, pp. 341-349). ACM.

[25] J. Sun, W. Stuerzlinger and D. Shuralyov, 2016, October. Shift-sliding and Depth-pop for 3D positioning. In *Proceedings of the 2016 Symposium on Spatial User Interaction* (pp. 69-78). ACM.

[26] J. Sun, W. Stuerzlinger and B.E. Riecke, 2018, August. Comparing input methods and cursors for 3D positioning with head-mounted displays. In *Proceedings of the 15th ACM Symposium on Applied Perception* (p. 8). ACM.

[27] A.M. Webb, A. Kerne, Z. Brown, J.H. Kim and E. Kellogg, 2016, November. LayerFish: Bimanual Layering with a Fisheye In-Place. In *Proceedings of the 2016 ACM International Conference on Interactive Surfaces and Spaces*(pp. 189-198). ACM.

[28] S. Zhai, W. Buxton and P. Milgram, 1996. The partial-occlusion effect: Utilizing semitransparency in 3D human-computer interaction. *ACM Transactions on Computer-Human Interaction (TOCHI)*, *3*(3), pp.254-284.