Navidget for 3D interaction: Camera positioning and further uses

Martin Hachet*, Fabrice Decle, Sebastian Knödel, Pascal Guitton

INRIA - Université de Bordeaux, France

Abstract

This paper presents an extended version of Navidget. Navidget is a new interaction technique for camera positioning in 3D environments. This technique derives from the point-of-interest (POI) approaches where the endpoint of a trajectory is selected for smooth camera motions. Unlike the existing POI techniques, Navidget does not attempt to automatically estimate where and how the user wants to move. Instead, it provides good feedback and control for fast and easy interactive camera positioning. Navidget can also be useful for distant inspection when used with a preview window. This new 3D user interface is totally based on 2D inputs. As a result, it is appropriate for a wide variety of visualization systems, from small handheld devices to large interactive displays. A user study on TabletPC shows that the usability of Navidget is very good for both expert and novice users. This new technique is more appropriate than the conventional 3D viewer interfaces in numerous 3D camera positioning tasks. Apart from these tasks, the Navidget approach can be useful for further purposes such as collaborative work and animation.
© 2008 Elsevier Ltd. All rights reserved.

1. Introduction

Navidget is a 3D user interface (3DUI) designed for easy camera positioning in 3D environments using 2D inputs. This technique was initially published in 3DUI 2008 (Hachet et al., 2008). This paper is an extension of the previous publication. In particular, we propose new functionality to better control the camera. We also show that the Navidget approach can be useful for other interaction tasks, such as collaborative work and animation.

Camera movement control in 3D applications has been widely studied since the early years of interactive 3D graphics. Many different user interfaces have been proposed to optimize the user performance. However, controlling the camera in a 3D environment remains a difficult task, and innovative interaction techniques still need to be designed to make user interaction easier.

In this paper, we propose a new UI attempting to reach this goal. This interface, called Navidget, allows easy and fast camera positioning in 3D environments from 2D inputs, by way of an adapted widget. Navidget derives from the *point-of-interest* (POI) technique introduced by Mackinlay et al. (1990), where the user selects the endpoint of a trajectory in order to automatically fly to a corresponding location. This technique is also known as the "go to" function for a wide range of 3D viewers. Selecting the endpoint of a trajectory as a navigation metaphor has several advantages:

Generic inputs. Only a pointing device and a start signal are requested by POI techniques. No additional standard workstation devices, such as keyboards or joysticks, are needed. Consequently, a POI approach can be used with a wide variety of alternative equipments, including mobile devices (e.g. smartphones and PDAs) and large touch screens for collective work. Indeed, the possible inputs with such equipments are often limited to pointing operations (as illustrated in Fig. 1). Consequently, POI techniques are particularly well suited for optimizing 3D navigation in alternative visualization situations.

---

*Corresponding author. Fax: +33 5 40 00 66 69.

E-mail addresses: hachet@labri.fr (M. Hachet), decle@labri.fr (F. Decle), knoedel@labri.fr (S. Knödel), guitton@labri.fr (P. Guitton).
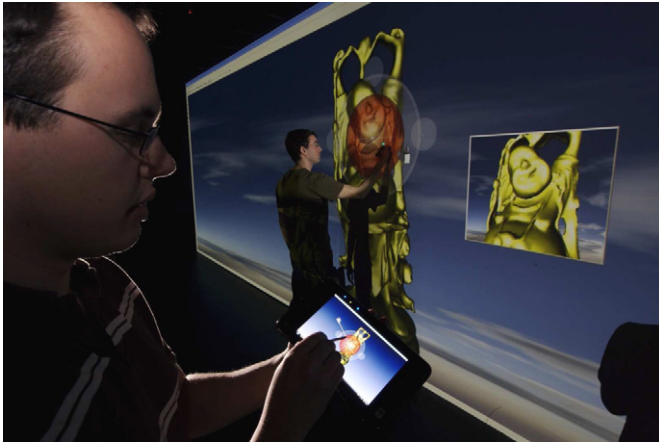
Fig. 1. Navidget on a UMPC and a large screen. ©*CNRS Phototheque / C. Lebedinsky.*

Ease-of-use. POI approaches are very simple and direct. Users simply have to select the point to which they want to fly. No learning time is required. All visible parts of the scene can be reached directly.

Fast. Camera movement in the 3D environment operates within controlled completion times. Hence, users can quickly travel long distance.

Cognitive-friendly. Cognitive maps, which play an important role in wayfinding tasks, are sometimes difficult for the user to construct when navigating in large 3D environments. POI users benefit from continuous movements (i.e. the whole environment is traveled over from the current position to the desired location). Moreover, the speed of camera movement gives an indication of the distance traveled.

Real-time independent. True real-time cannot always be achieved in 3D applications, particularly when very large scenes have to be displayed or when low computation devices such as handhelds are used. In this case, direct camera controls must be avoided, since the time-lag between the user's actions and the related camera movement in the 3D scene can be very disturbing. With POI approaches, the user simply selects a target and waits for the camera movement, so the time-lag is much less objectionable as an immediate feedback is not required.

On the other hand, POI techniques have two significant drawbacks:

Surface-dependent. The target is always a point on a surface. Consequently, the only possible camera movements are translations towards a surface of the 3D scene. This is restrictive as users may want to move to other locations. For example if two objects are displayed, users have to choose the one they want to fly to. It is not possible to move to a location where both objects can be seen within a closer view.

Limited control. Users can only give the point to which they want to fly. They do not control how they want to go there. In particular, users do not control parameters such as the distance between the endpoint of the camera

trajectory and the target, and the angle from which they want to view the target. These parameters are generally fixed or automatically computed. This results in some awkward situations. For example, facing a large surface that occludes the whole scene is very common when using classical POI techniques.

In spite of these two drawbacks, POI techniques are still widely used as they are often the easiest techniques. "Go to" is one of the standard functions of Web3D viewers. In many cases, users navigate in 3D environments by simply pointing to some part of the scene. POI techniques are useful in many visualization contexts for novices as well as for expert users. In this paper, we propose to extend the POI approach by providing additional controls to the user.

In the next section, we present some previous work that are related to this approach. We then describe the Navidget technique in detail in Section 3. In Section 4, we present the user study that has been carried out. We present how Navidget has been extended in Section 5. In addition to camera positioning tasks, we show in Section 6 that the Navidget approach can be beneficial to the user for many other tasks. Finally, in Section 7, we give our conclusion and suggest some directions for future work.

2. Previous work

Control of the virtual camera in 3D environments requires at least 6 degrees of freedom (DOF). These DOF can be directly controlled by means of 6-DOF input devices using adapted metaphors. For example, Ware and Osborne (1990) proposed the *scene-in-hand*, *eyeball-in-hand*, and *flying vehicle* metaphors. However, such 6-DOF devices are not that common. Consequently, the challenge that researchers have tried to resolve for many years is how to control the camera using only the 2 DOF available on conventional input devices, such as mice.

With 2-DOF devices, the standard techniques are *orbiting*, *flying* and POI. Orbiting is used for observation of an object from an exocentric point of view. The principle is to map the motions of the controller to motions over the surface of a sphere (Chen et al., 1988), or other 3D primitives. 3D motions can also be achieved by means of external widgets, as is the case with Open Inventor-like viewers (Strauss and Carey, 1992). Other advanced techniques such as the HoverCam (Khan et al., 2005) and the StyleCam (Burtnyk et al., 2002), are based on camera movements along predetermined surfaces or trajectories. Orbiting techniques are mainly used for observing objects.

The flying approach, which is widely used in video games, is related to the *flying vehicle* egocentric metaphor, but with only 2 DOF. These 2 DOF are generally assigned to forward/backward movements and left/right camera rotations. If a keyboard is available, the forward/backward movements can be controlled using certain keys, leaving up/down 2D controller movements free to control top/bottom camera rotations. Some advanced techniques allow

automatically constraining camera trajectories according to the topology of the scene (e.g. Hanson and Wernert, 1997), or according to the speed of the movements (Tan et al., 2001). Flying techniques are mainly used for exploring 3D scenes. Contrary to the above techniques, Navidget does not directly map the input DOF to the camera DOF. It rather allows the user to define target destinations, without having in mind the camera DOF that are implied.

The POI approach has been shown to have many benefits, as discussed in the introduction. This technique has primarily been described by Mackinlay et al. (1990), where logarithmic control of speed is proposed. Some extensions have since been explored. For example, in Mackinlay et al. (1994), radial and/or lateral viewpoint motion have been discussed. Zeleznik and Forsberg (1999) describe the UniCam where one of their techniques, called *click-to-focus on silhouette edges*, is aimed at automatically choosing the endpoint of the camera trajectory according to the proximity of the edges of some object. Hachet et al. (2006) proposed the *z-goto* for mobile devices, an extension of the "go to" approach, where the endpoint is directly selected in depth by means of simple keystrokes. For all these techniques, the virtual camera is automatically positioned in the 3D space. Users do not have any direct control once the target point is selected. In our approach, we want the user to be able to control the camera destinations. Zeleznik et al. (2002) have investigated the use of controlled POI for VR setups. They proposed a set of techniques where users can adjust the viewing direction and the distance to the target by way of tracked VR devices. Navidget shares the same goal, but for standard 2D screen-based interfaces.

In the scope of computer graphics, 3D widgets such as Bier's (1987) Skitters and Jacks are commonly used. Today, the wide majority of the modeling applications are based on these visual 3D elements for the manipulation of 3D objects (e.g. Blender,). These widgets have shown many benefits for manipulation tasks (translate/rotate/scale). On the other hand, the use of 3D widgets for navigation and inspection tasks has been little explored. The IBar (Singh et al., 2004) has been proposed to control some parameters of the camera in an interactive fashion (e.g. perspective distortion). In our approach, we donot want to control the camera parameters independently. We rather provide an interface that allows the user to complete a 3D camera positioning task in a unique operation.

An alternative approach for 3D interaction tasks is the use of sketch-based techniques. The idea is to let the user control the system from simple 2D strokes. For example, *SKETCH* (Zeleznik et al., 1996) and *Teddy* (Igarashi et al., 1999) interpret the input gesture for modeling and editing tasks. A sketching philosophy has also been proposed for animation tasks in *Motion Doodles* (Thorne, 2004). Concerning camera manipulation, Unicam (Zeleznik and Forsberg, 1999) is based on a gesture vocabulary for the control of the camera parameters. Finally, the system proposed by Igarashi et al. (1998) allows 3D walkthrough from free-form path drawing on the 3D scene ground. These gesture-based techniques, where the user can control the system by way of simple 2D gestures inspired our work.

3. Navidget

3.1. General approach

The main idea of Navidget is to let the users control where they want to look at, contrary to the conventional POI techniques that try to guess where the user wants to focus. With a 2D sequence only (i.e. press, move and release), the Navidget users define their target destinations. This is done using adapted controls and a dedicated widget.

3.2. Navidget controls

3.2.1. Distance

The simplest gesture sequence that is related to Navidget consists in pointing a target destination by picking a 3D point in the scene. This is similar to the conventional POI approach. The main problem with this basic approach is that the camera can only be moved towards a surface. Moreover, the resulting view may be little convincing as the user does not control the depth where the camera has to stop (see Fig. 2). To solve these problems, we propose an initial extension consisting in circling the target area.

Circling is highly intuitive as the user directly draws what he or she would like to see, as illustrated in Fig. 2. Circling is particularly well suited to stylus-based and finger-based systems. Indeed, this gesture is usually done with a pen on a paper when something has to be highlighted. The circling metaphor has been used by Schmalstieg et al. (1999) with transparent props for selecting virtual objects. However, their work did not involve navigation tasks. Circling can be achieved using any pointing device, depending on the hardware settings. With mouse-based environments, the circling metaphor can be replaced by the conventional rectangle-based selection technique, which is commonly used in many 2D software programs or with some 3D orthographic CAD applications.

After a release event, the camera moves towards center of the circle. There are several possibilities to compute the final camera position, as presented in Fig. 3. The first and simplest one (CENTER) makes use of the 3D point given by the projection of the circle center into the
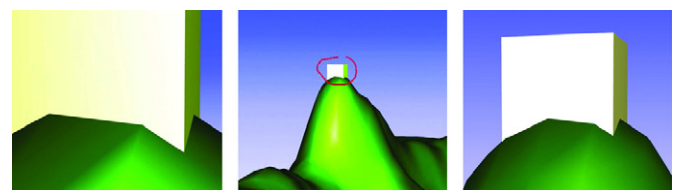

Fig. 2. Pointing a target may induce senseless views (left). Circling it moves the camera to an appropriate view (right).
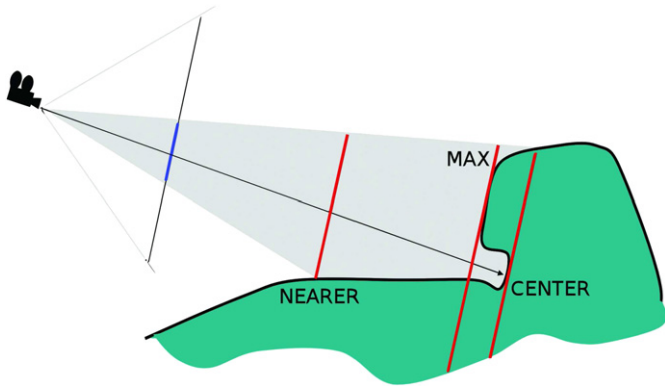
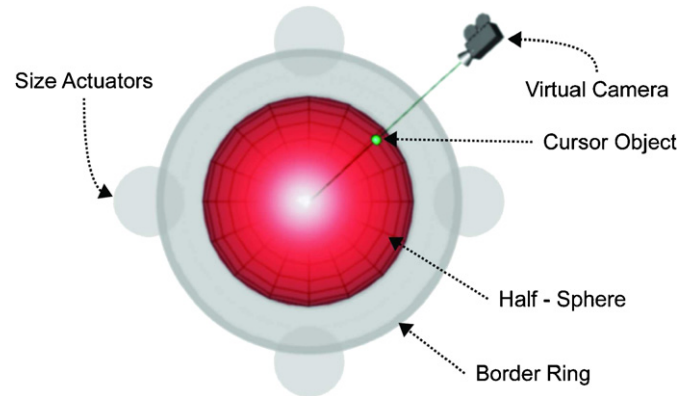Fig. 3. The three different possibilities (NEAR, MAX, CENTER) to compute the depth value.

Fig. 4. The Navidget.

scene. The drawback of that naive approach is that this particular point does not necessarily exist. To overcome that issue, we developed an alternative approach (MAX) based on the most frequent depth value of the selected area. The main drawback of this method is that objects that are located between the current viewpoint and the target area might not be visible afterwards. Finally, the most conservative technique (NEARER) considers the nearest depth value of the selected area in order to insure that the whole selection is visible.

Because of perspective, the resulting view is not exactly what has been circled. It should be noticed that Zeleznik and Forsberg (1999) have proposed a concept of *region zooming* that may be related to our technique. However, with their technique, users must first select a point before modifying the size of the zooming area. Consequently, the target area is always centered on a 3D point in the scene. We think that circling is a very intuitive metaphor, as users directly show what they want to focus on using a simple and familiar gesture.

3.2.2. Viewing direction

The targeted viewing direction is automatically computed in existing POI techniques. Generally, the camera looks at the target point with a viewing direction that is aligned with the normal of the related face, or with a pre-defined oblique view (Forsberg et al., 1998). Whatever the choice made by the designers of POI techniques, the final camera angle is fixed. Consequently, users cannot control how they want to visualize the target. Worse, automatic computation of the destination views can be disturbing. For example, the viewpoint can be moved to a position which is too close to the target.

We use a 3D widget in order to let the user control the viewing direction at destination. This widget, illustrated in Fig. 4 is composed of a half-sphere facing the current viewpoint, a border ring, a set of size actuators, and a virtual camera. This widget is centered on the target. It appears if no release events occur just after a pointing operation. In other words, if the user clicks and waits for a short time, the widget appears. At this level, a default
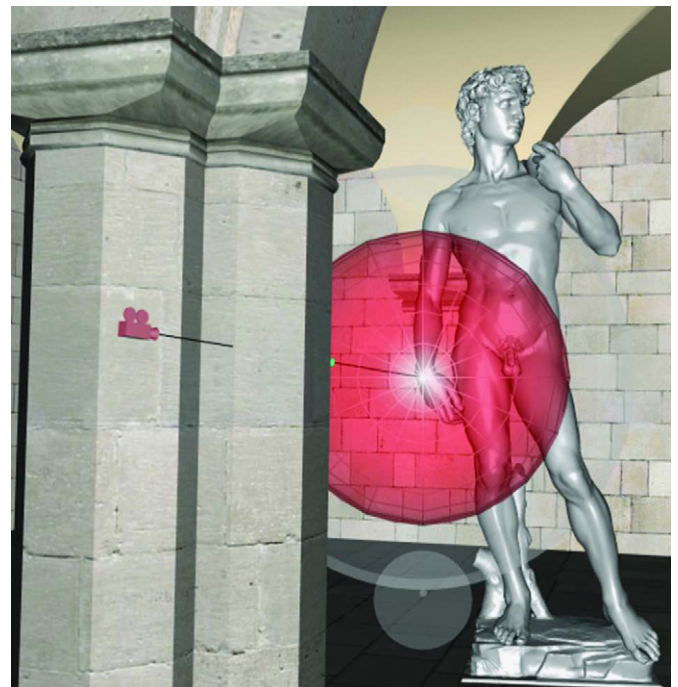

Fig. 5. The visual feedback provided by Navidget allows the user to avoid occluded views.

radius is computed in order for the widget to appear as a constant proportion of the application window. We did not base our approach on object bounding-boxes in order not to constrain the user interaction to object hierarchies. Indeed, Navidget allows the users to focus on specific parts of large objects such as terrains, or to travel towards areas where several objects are implied.

The half-sphere highlights the part of the scene that will be seen at the end of the camera trajectory. By moving a 3D cursor on this surface, users control how they want to visualize the target. A representation of a camera shows the position and angle to which the viewpoint will be moved. This intrinsically provides good feedback to the user. Indeed, the possible occlusions can be seen clearly, so they are naturally avoided (see Fig. 5). In Section 3.2.5, we present an extension, called *smart camera*, which allows

emphasizing the visual feedback that is provided by Navidget.

A pilot study that will be described in Section 4 revealed a need for efficient selection of viewpoints that are orthogonal to the current viewing direction. For this purpose, we added a border ring. Indeed, reaching the border ring is more convenient than accurately positioning the cursor on the border of the half-sphere. Moving the cursor around the border ring has similar effect than moving it along the visible limits of the half-sphere.

The pilot study also revealed that the users wanted to move the virtual camera behind the half-sphere. Consequently, we introduced a mechanism allowing switching the direction of the half-sphere, in front or behind the target depth plane. When the cursor exits and re-enters the half-sphere, the latter is 180° rotated through a quick animation (see Fig. 6). This allows moving the virtual camera all around the target area. A fast outside–inside movement allows going directly behind the target area. A similar gesture within a slower movement insures continuity in the virtual camera movements, the border ring being similar for both front and back half-spheres.

3.2.3. Distance and viewing direction

Similarly to the pointing technique, the widget appears if no release event occurs once a circling gesture has been performed. After a hold phase, the widget is displayed and the radius is directly given by the size of the drawn circle. In this way, the user can manage both the distance to the target and the final viewing direction. This sketch-based approach enhances intuitive interaction. Of course, the widget can be called after rectangle-based area selection, too.

If available, the barrel button, the non-preferred hand, pop up menus or pressure-based input could also be used to make the widget appear. However, since our technique is based on a simple and unique gesture sequence by a one handed interaction, we want to avoid extra pop up menus as well as additional interaction using the non-preferred hand. Furthermore, using the barrel button or a pressure based input is not suitable for stylus or finger-based interaction system.

We added some size actuators that allow the widget to be resized at any time, without releasing the stylus or the finger. These actuators are located on the cardinal direction of the half-sphere. Once these actuators are captured, the system is set to a RESIZE state. The user can then modify the size of the widget—and consequently the distance of the virtual camera to the target—by doing vertical (resp. horizontal) movements on the screen. The RESIZE state is left when horizontal (resp. vertical) movements are detected (see Fig. 7). This means the user wants to come back to the half-sphere or the border ring to specify the viewing direction. Hence, by controlling a direction and a distance, users are able to position the camera in the 3D space.

When a release event occurs and the cursor is located on the half-sphere or the border ring, the viewpoint is smoothly moved to the target virtual camera. If the cursor is outside, the action is canceled and the widget is hided.

To obtain a smooth camera movement we create a key-frame animation by interpolating the position and orientation of the initial viewpoint and the specified target camera. After a circling gesture we use a simple linear interpolation, which results in a straight-lined movement towards the selected target. If the half-sphere widget was used to specify the target, we perform a Bézier interpolation that results in a smooth curved trajectory. Every animation lasts 100 key-frames (ca. 3 s), which guarantees that the target is reached in the same time, independently of the distance to travel.

3.2.4. Preview window

Navidget has initially been designed to favor fast camera positioning in 3D environments. We noticed that it could also be useful for distant inspection tasks. A preview window corresponding to the virtual camera's view can be displayed in addition to the main visualization window (see Fig. 8). Thereby, users can inspect distant areas and decide whether or not to move to the corresponding targets, according to what they have seen. Concretely, if the users just want to inspect objects without moving in the scene, they will release the pen outside of the widget. This refers to a cancel action. If they want to move to the target, they will release inside the widget, once the preview visualizations correspond to their targets.

It can be noticed that this technique recall previous work such as the one by Grosjean and Coquillart (1999), where distant inspection was possible from manipulation of a 6-DOF magic mirror in immersive virtual environments.
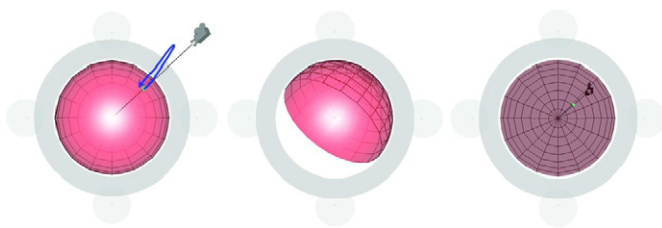

Fig. 6. An outside–inside movement (*blue stroke*) allows switching the orientation of the half-sphere (front or back), so the virtual camera can be moved behind the target area.
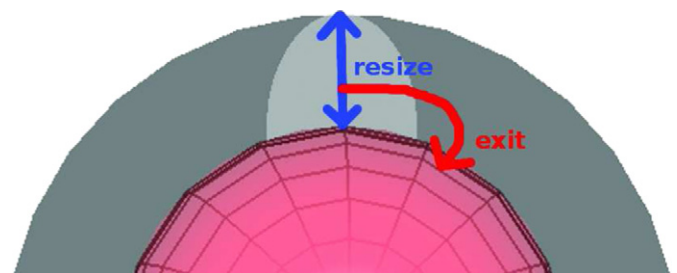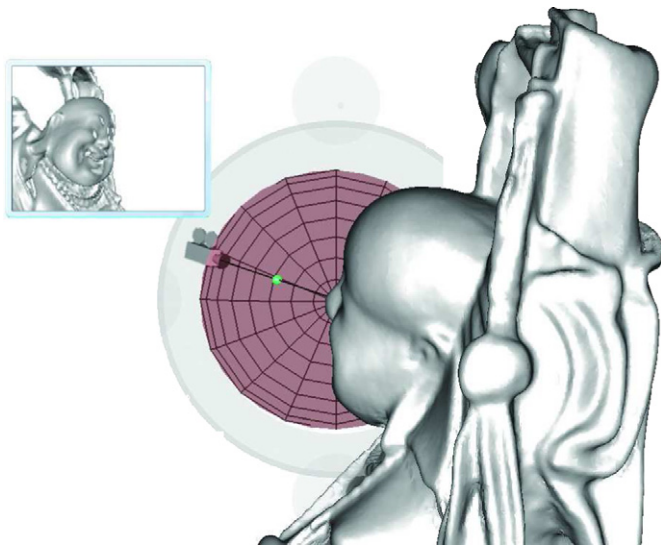

Fig. 7. The size actuators.

Fig. 8. The preview window. In this example, the half-sphere has been moved behind the target area.

With Navidget associated with a preview window, the user can look all around the target area with a 2-DOF input control.

A preview window is not adapted when small displays such as those of the mobile devices are used. Additional windows may also be disturbing with immersive stereoscopic visualization. However, when the display of an additional window is possible, the use of a preview window with Navidget can be very useful. This allows exploring a 3D scene without moving one's position. This is particularly interesting as the user can focus on specific areas while keeping the global context. This *focus + context* approach is useful for the cognitive processes that are implied in 3D navigation tasks. It has to be noticed that this technique requires additional computation. Consequently, real-time rendering can be a problem with big 3D scenes.

3.2.5. Smart camera

As we have already described in Section 3.2.2, Navidget provides users with good visual feedback. In addition to this intrinsic feedback, we propose an extension that supports the user by providing additional information as the camera is being positioned. This extension is useful as soon as an object is located between the POI and the position of the virtual camera, or when the camera is covered by an occluder. Both cases are presented in Fig. 9(a) and (c).

Since we know the position of the POI and the current position of the camera, we are able to detect any geometry between these two positions that could disturb the user. Therefore, we can visualize the occluding part in high contrast (see Fig. 9(b)) so it is clearly visible to the user. Furthermore, we can prevent the camera from vanishing behind an object by making any such objects transparent, as illustrated in Fig. 9(d).
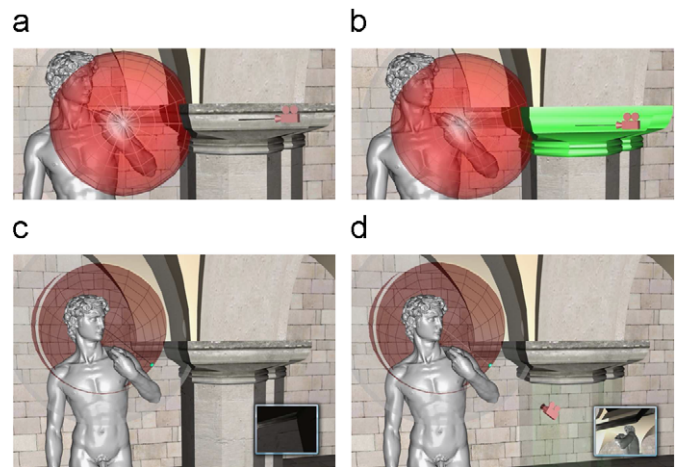

Fig. 9. Smart camera: (a, c) without smart camera and (b, d) with smart camera.

It should be noted that the computational cost to detect occlusions increases with the complexity of the scene. Indeed, precise occlusion computation implies numerous intersection tests. In order to preserve real-time, we currently use a simple method based on two intersection tests. The first one tests the visibility from the virtual camera to the center of the half-sphere (Fig. 9(b)). The second one tests the visibility between the current viewpoint and the virtual camera (Fig. 9(d)).

Beyond these tests, we could also move the camera to a more convenient position where no occlusions occur. However, this would imply taking away control from the user, which may cause disturbances.

4. User study

4.1. General comments

In this section, we describe a pilot study we carried out to assess Navidget. We have designed a set of experiments in order to evaluate the general usability of the technique. The feedback we obtained from the subjects who participated in this study helped us for the final design of Navidget. For all the experiments, a TabletPC was used. Our study focused on tasks where Navidget can be useful, i.e. when the user wants to visualize 3D areas from specific viewpoints.

Our motivation for this study was to evaluate the usability of each Navidget control. Moreover, we wanted to have an idea about the user preference between Navidget and the standard 3D viewer controls.[1] We set up experiments where the subjects had to complete camera positioning tasks. After the experiments, we asked the subjects to answer a questionnaire.

In our preliminary experiment, we did not use time as a metric to evaluate Navidget in comparison with the standard controls. Indeed, completion times largely depend on the speed that is chosen for Navidget animations. It also

---

[1]Standard controls were *fly*, *pan*, *look around*, *go to* and *orbiting*.

depends on the sensibility levels that are set for the direct controls. Consequently, the completion times obtained would have been strongly linked to the initial settings.

4.2. Procedure

Thirty subjects divided into two groups took part in this first study (21 males, nine females, average age = 26). For the first group, 15 3D expert users were recruited. The second group included 15 novice users (i.e. having less than one experience using videogames or 3D applications a month). For each group, half of the subjects completed all the experiments with Navidget, then with the standard controls. The other half began with the standard controls.

The experimental scenes consisted in cubes for which one face was highlighted. On this face, some letters (red "A" and black "B") were written. The tasks entails counting the number of red "A". To complete the task, the subjects had to move to appropriate viewpoints (see Fig. 10). A trial was completed when the subject selects a number as an answer of the task. This was done using an answer panel.

Before the experiments, we presented both interfaces to the subjects. They tried each control at least once inside a test environment to get familiar with them.

During the experiment with Navidget, the subjects were explicitly asked to use one of the Navidget controls (i.e. *point only*, *circle only*, *point + widget*, *circle + widget*, or *widget + size actuators*). Each control has been used three times with different targets' size and location.

4.3. Results

The analysis of the questionnaires have shown that the general usability of Navidget is good. Both expert and novice subjects did not have difficulties when using the Navidget controls (see Table 1). The circling metaphor has been appreciated. Indeed, the link between circling and focusing is very strong. Consequently, the circling metaphor was used extensively by the subjects, even if a simple pointing technique would have been necessary. Similarly, sketching the widget by way of a circle gesture works well. The link between the drawn circle and the corresponding 3D widget was easily understood by the subjects. The subjects managed to define the targeted viewing direction easily. They have understood the role of the half-sphere

Fig. 10. Example of an experimental scene for the pilot study. From the initial view (*left*), the subjects had to move closer to the target face (*right*) in order to count the number red "A".

directly. Similarly, the subjects have controlled the size actuators efficiently, from the first use.

Among the comments we had, some subjects complained not to be able to move the virtual camera behind the half-sphere. For this reason, we have developed the mechanism that enables to switch the half-sphere front and back. Some additional subjects have tested this mechanism in an informal way. They appreciated the technique and they have used it without difficulties.

Similarly, the border ring has been developed from subjects' comments. The addition of this functionality has improved the interface. Indeed, side views can be reached very fast. We also have good feedbacks from the first users concerning the Navidget preview window.

The pilot study has shown that the wide majority of the subjects preferred using Navidget rather than the conventional controls for the proposed tasks, as shown in Fig. 11.

Wilcoxon signed rank tests have shown that this preference was significant for each question we asked[2] to both experts and novices, except for the ease of understanding where both interfaces were easily understood. In particular, both expert and novice subjects found that Navidget was easier to use than the standard controls (Expert *Wilcoxon Z test*: $-3.52, p < 0.05$; Novice *Wilcoxon Z test*: $-2.99, p < 0.05$). They also found Navidget faster (Expert *Wilcoxon Z test*: $-3.45, p < 0.05$; Novice *Wilcoxon Z test*: $-3.34, p < 0.05$).

With Navidget, a simple action is used to complete the task. The same task requires switching between different techniques when using the standard controls, which is time consuming. Following Buxton (1995), each control switch divides the action in separate chunks, because the applied motor tension changes and the motion continuity is interrupted. This leads to higher separation into subtasks and therefore to higher cognitive load.

4.4. Discussion

Novice users had some difficulties using conventional controls they were not familiar with, particularly since they had to remember the specific nature of each of them (translation in the xy plane means nothing for a novice user). Generally, they were trying a control without knowing what was going to happen in the 3D scene. They often got lost. Some of them also reported some motion sickness.

On the other hand, with Navidget, the subjects enjoyed dealing with a technique where several parameters can been controlled by a simple gesture. Unlike the standard controls where the 2 input DOF are directly mapped to 2 of the camera viewpoint DOF (e.g. *z-translation + y-rotation* for *fly*), the 2D gestures of Navidget make it possible to perform basic camera positioning tasks, where multi-DOF are jointly implied. The rich visual feedback of Navidget favors efficient interaction. During
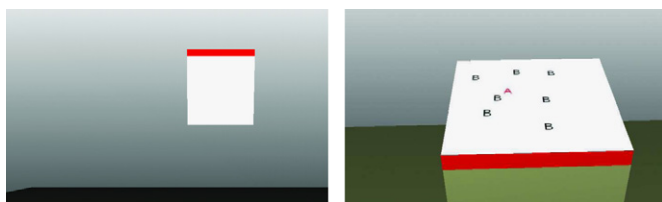
---

[2]See Fig. 11 for the questions.

Table 1
Satisfaction for each of the controls of Navidget [Novices, Experts (*mean*)]

|	Circle	Half-sphere	Resize actuators	Circle + Half − sphere
Was the interaction technique easy to understand?	3.0, 2.9 (3.0)	2.9, 2.8 (2.8)	2.6, 2.6 (2.6)	3.0, 2.9 (3.0)
Was the interaction technique easy to use?	2.9, 2.9 (2.9)	2.7, 2.7 (2.7)	2.2, 2.4 (2.3)	2.7, 2.8 (2.8)
Were you able to interact freely?	2.9, 2.9 (2.9)	2.5, 2.9 (2.7)	2.3, 2.8 (2.5)	2.9, 2.9 (2.9)
Did you feel like interacting precisely?	2.9, 2.6 (2.7)	2.6, 2.7 (2.6)	2.3, 2.8 (2.5)	2.6, 2.8 (2.7)
Did you feel like interacting fast?	2.8, 2.9 (2.8)	2.8, 2.9 (2.8)	2.4, 2.6 (2.5)	2.7, 2.9 (2.8)

0 = Fully disagree, 3 = fully agree.
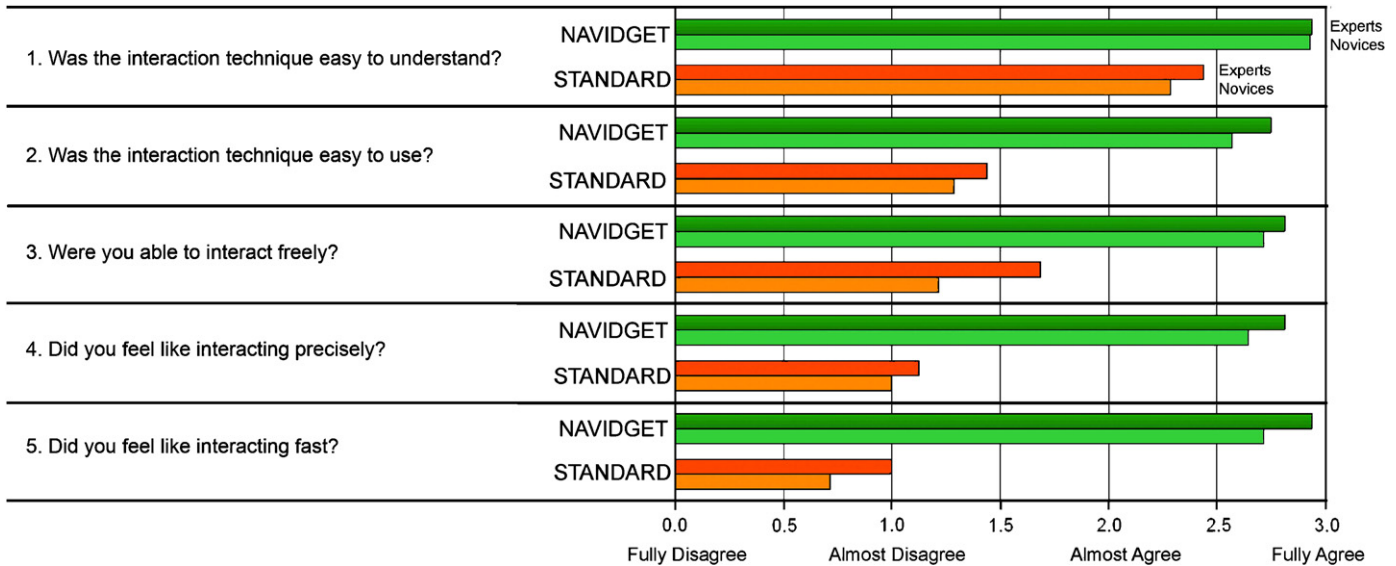
Fig. 11. General satisfaction for Navidget and standard controls.

the experiments, the subjects were exactly knowing what was going to happen. They were able to easily control where they wanted to move. Thanks to smooth camera movements, no motion sickness has occurred.

Zooming + orbiting is the standard technique that relates the most to Navidget. We have noticed that this technique may perturb the user when large scenes were used. Indeed, the whole scenes move around the user, which can be disturbing. Moreover, many occlusions can occur. This makes this technique little adapted to dense environments. Of course, *Zooming+orbiting* is well suited for exocentric tasks where single objects are observed. On the other hand, Navidget seems better suited when users navigate large 3D scenes with egocentric viewpoints. Navidget seems better suited when true real-time cannot be insured, too. Further investigations have to be conducted to confirm this.

Of course, this study does not mean that Navidget is better than the standard controls in general. It simply shows that, for camera positioning tasks, Navidget can be more appropriate. The current version of Navidget is adapted to target-based navigation, where the destination is visible from the current viewpoint; it does not fit well with naive exploration of 3D scenes. During the experiments we conducted, some participants complained that they were not able to return to previous camera positions.

We advised them to use another standard control (e.g. fly and look-around) as an alternative. But they did not want to take back the direct control of the camera movements. Consequently, we have extended the initial Navidget as described below.

5. Extended Navidget

In order to allow the users to navigate using only Navidget, we introduced two new sets of gestures. The first one allows them to come back to previous states while the second one makes it possible to turn the camera in any direction.

5.1. Back-gesture

A back-gesture (see Table 2) entails repositioning the camera in its previous state. In other words, the camera follows the last trajectory path that has been played, but in the reverse direction. From the user's perspective, this action can be seen as an ''undo'' action. Technically, all the new positions and orientations of the camera are pushed in a stack. An undo gesture entails popping the stack to move the camera back to its previous state.

Table 2
Extended Navidget gestures

Gesture	Resulting action from gestures
Point and encircle	Focus on an area
Back-gesture	Go back to previous location
Horizontal/vertical strokes	Turn the view

Navidget, associated with this new functionality, allows the users to explore 3D environments in a new manner. This type of 3D exploration can be related to website navigation, where users explore new areas of interest by way of hyperlinks, and return to previous states using "back" actions. We think that such simple web-based navigation can help the users in their navigation tasks since they benefit from a well-established mental model.

For example, let's consider a user who wants to observe several objects that are simultaneously visible in a 3D scene. With a standard approach, the user would have to reach the first target, then would have to search for the second one from the new location, and so on. This can be difficult, particularly because wayfinding in 3D environments is a difficult task. With the extended version of Navidget, the user can focus on the first object, then can come back to the initial location by way of a simple stroke on the screen, and then focus on a new target, and so on. This is similar to what users do when navigating through web pages.

5.2. Turn-gestures

Another feature we have implemented to extend the initial version of Navidget is the possibility for users to turn the view around their current location. This is done by way of vertical and horizontal strokes (see Table 2). Following the Navidget philosophy, the users do not control directly the movements of the camera. By drawing a vertical (resp. horizontal) stroke, users indicate the system that they want to look up/down (resp. left/right). At the end of the stroke, the camera smoothly turns in the sketched direction.

Technically, a 45° horizontal/vertical rotation is applied to the camera. With a standard 90° camera *field-of-view*, this entails moving the viewing vector towards one of the edges of the screen. Hence, the users can look everywhere around them using simple strokes.

5.3. Summary

The gestures for the extended Navidget are summarized in Table 2. The number of gestures that the users have to remember is small. Consequently, this extended version of Navidget does not require an additional learning period.

We have not conducted a formal study to evaluate these new controls. However, according to our first user feedback, it appears that this extended version of Navidget works well. The users have not experienced any difficulties in assimilating the new Navidget functionality, which provide more control for navigation in 3D environments.

6. Further uses of the Navidget approach

In the previous sections, we have described Navidget in terms of camera positioning tasks. In the sections that follow we will present several implementations that show how the Navidget approach can be useful in many different situations.

6.1. Defining a 3D vector

Navidget makes it possible to orient a 3D vector around a target area, from 2D input interaction. For camera positioning tasks, this vector refers to the viewing direction at which the user wants to visualize a target. This 3D vector can also be used for other tasks. We present two implementations in the following sections.

6.1.1. 3D pointing

With current 3D applications, pointing generally refers to the selection of a 3D point using a picking operation from the standard 2-DOF pointing devices. By using the Navidget approach, users are no longer limited to 3D positions; they can also control orientations as well as distances to targets. This can be beneficial in numerous situations. For example, an adapted design of the Navidget widget can be useful for the definition of force vectors in interactive physical simulation. could be envisioned, such as spot light positioning, virtual drilling, clipping plane positioning, and so on. Fig. 12 illustrates an example of 3D pointing.

6.1.2. Rotation of objects

The Navidget approach can also be useful for manipulation tasks. It allows users to define 3D axes of rotation, around which the object can spin. Current interfaces for object rotations are either based on a virtual trackball or canonical decomposition of the rotation space. In the first case, many rotations are difficult or even impossible to achieve (e.g. rotation around the viewing direction axis). In the second case, the users have to compose the rotation they want to achieve using iterative adjustments. On the other hand, with the Navidget approach, a rotation axis can be defined very quickly and easily. The benefits of this approach for such manipulation tasks are similar to those
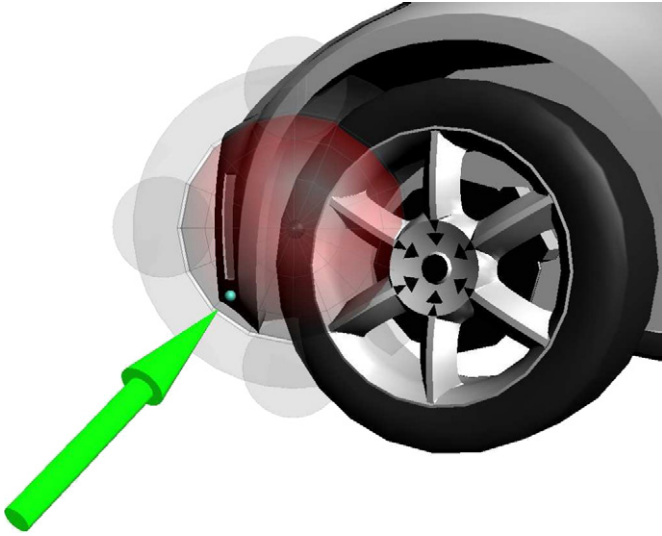
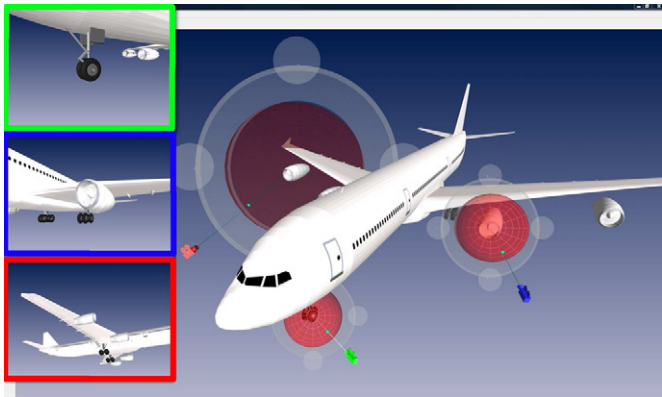Fig. 12. The Navidget approach for 3D pointing.

Fig. 13. Multiple Navidget for simultaneous inspection.

for camera positioning tasks: users can concentrate on what they want to do, rather than concentrating on how they can do it.

6.2. Multiple cameras

Until now, we have considered the use of one Navidget at a time. Below we shall discuss some implementations where it could be beneficial to use several Navidget interfaces at the same time. In these examples, the user's viewpoint does not move automatically towards the targets. The widgets stay visible in the scene and the user can access each of them.

6.2.1. Defining several control viewports

It can be beneficial to visualize a 3D scene from several viewpoints, at the same time. If we consider the example of 3D physical simulation, it may be necessary to see several parts of the scene simultaneously in order to control the evolution of physical phenomena. To this end, we developed a small prototype, using several widgets associated with corresponding preview windows that can

be positioned in the scene, as illustrated in Fig. 13. During the simulation, if required, the user can easily adjust the virtual cameras.

6.2.2. Animation

Similarly, the use of multiple widgets can be useful for animation purposes. Indeed, animation paths are specified as a sequence of key (view) points with orientation defined by positioning several widgets into the scene. Next, the system associates time values and automatically calculates continuous intermediate values using an interpolation method.

Navidget naturally fits well with the definition of the key viewpoints, as the interface has been designed to allow fast and easy camera positioning around focus areas. In the test application we developed, the user can define key viewpoints by using several widgets and then play a camera motion along an interpolated path. We use uniform cubic B-Splines to obtain a smooth animation by interpolating the acquired key viewpoints. The positions and orientations of the virtual key cameras can been adjusted at any time.

6.3. Collaboration

We think that collaborative applications can benefit from Navidget, too. Two examples are illustrated below, which we implemented as prototypes.

6.3.1. Shared displays

In front of large screens, several participants can visualize the same image. This is of great benefit for collaborative work. On the other hand, the viewpoint is shared by all participants, so if one of them wants to focus on a specific area, the whole group is affected by the applied camera motions. We propose a new approach where several participants can share global visualization of a scene (e.g. a car) while simultaneously focusing on different specific parts of it (e.g. the tyres, the lights, and so on). To achieve this, the participants must be equipped with personal pointing devices (e.g. lasers, Wiimotes, and PDAs). Then, all of them control their own Navidget on the large screen. This allows them to reach private views in the scene. These private views can been displayed on the side of the main window as illustrated in Fig. 14. They can also be sent to the participants' personal devices (e.g. PDA and TabletPCs).

6.3.2. Distant collaboration

We have previously described the use of the Navidget approach for 3D pointing. This functionality is particularly beneficial for distant collaborative work. Indeed, distant collaborators have to understand one another despite the fact they cannot see each other. Consequently, fast and easy 3D pointing from 2D inputs is very beneficial for this type of applications. Indeed, the collaborators can highlight precisely what they are talking about.
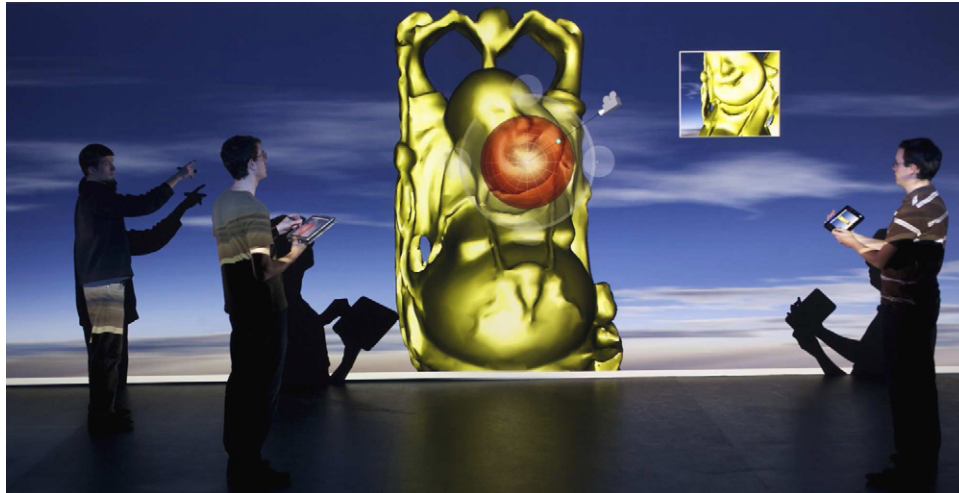
Fig. 14. The shared view is displayed on the large screen while Navidget allows the participants to get personal views on their TabletPCs. ©*CNRS Phototheque/C. Lebedinsky*.
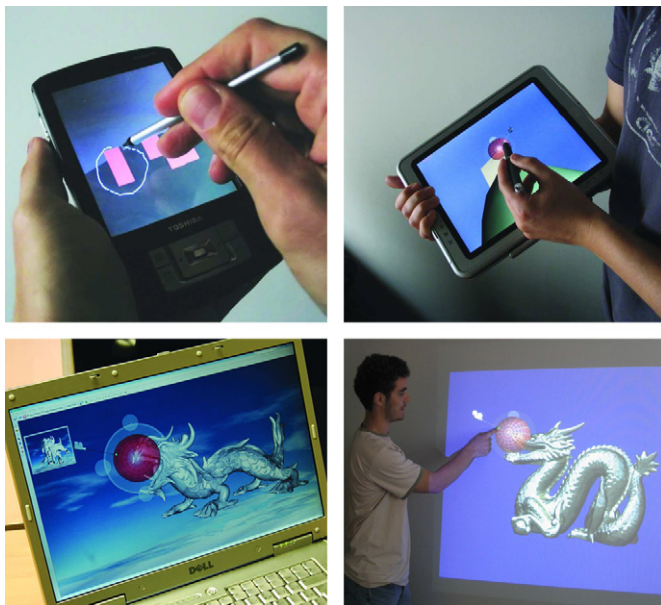

Fig. 15. Navidget on various systems.

7. Conclusion

In this paper, we have proposed a new approach for camera positioning tasks. The key idea of this technique is to provide good controls and feedbacks to the user, in order to allow them to easily position the camera around focus areas.

We have implemented Navidget with OpenGL, OpenSG, and OpenGL|ES. The technique is based on mouse events. It can be used with a wide variety of platforms: PDAs, TabletPCs, standard workstations, large touch screens, and so on, since the entire technique can be controlled by means of a simple 2D gesture sequence. Fig. 15 shows some examples. Moreover, the same technique can be used with immersive virtual environments, by means of a virtual ray controlled by some trackers for example. This portability is

one of the strengths of Navidget. Users do not have to learn new context-dependent techniques when dealing with new interactive systems. Instead, they benefit from a unified technique. This enhances user mobility.

The user study has shown that the usability of Navidget was good. It has also shown that, Navidget can be a good alternative to the standard controls that have been available for years on many 3D viewers. With these controls, many basic navigation tasks are difficult to perform. Navidget can make the completion of these tasks easier.

In addition to camera positioning tasks, we have shown some examples where the Navidget approach can help the user in many interaction tasks. This new interface changes the way we interact with 3D data.

References

Bier, E.A., 1987. Skitters and jacks: interactive 3d positioning tools. In: SI3D '86: Proceedings of the 1986 Workshop on Interactive 3D Graphics. ACM, New York, NY, USA, pp. 183–196.

Blender. ⟨http://www.blender3d.com⟩.

Burtnyk, N., Khan, A., Fitzmaurice, G., Balakrishnan, R., Kurtenbach, G., 2002. Stylecam: interactive stylized 3D navigation using integrated spatial & temporal controls. In: Proceedings of UIST '02. ACM Press, New York, pp. 101–110.

Buxton. W.A.S., 1995. Chunking and phrasing and the design of human–computer dialogues. pp. 494–499. Available at ⟨http://portal.acm.org/citation.cfm?id=212970#⟩.

Chen, M., Mountford, S.J., Sellen, A., 1988. A study in interactive 3-d rotation using 2-d control devices. In: Proceedings of SIGGRAPH '88. ACM Press, NY, pp. 121–129.

Forsberg, A.S., Dieterich, M., Zeleznik, R.C., 1998. The music notepad. In: Proceedings of UIST'98, pp. 203–210.

Grosjean, J., Coquillart, S., 1999. The magic mirror: metaphor for assisting the exploration of virtual worlds. In: Proceedings of 15th Spring Conference on Computer Graphics, pp. 125–129.

Hachet, M., Dècle, F., Guitton, P., 2006. Z-goto for efficient navigation in 3D environments from discrete inputs. In: Proceedings of VRST, pp. 236–239.

Hachet, M., Dècle, F., Knödel, S., Guitton, P., 2008. Navidget for easy 3D camera positioning from 2D inputs. In: Proceedings of IEEE 3DUI—Symposium on 3D User Interfaces, pp. 83–89.

Hanson, A.J., Wernert, E.A., 1997. Constrained 3D navigation with 2d controllers. In: Proceedings of VIS '97. IEEE Computer Society Press, Silver Spring, MD, pp. 175–182.

Igarashi, T., Kadobayashi, R., Mase, K., Tanaka, H., 1998. Path drawing for 3D walkthrough. In: Proceedings of UIST'98, pp. 173–174.

Igarashi, T., Matsuoka, S., Tanaka, H., 1999. Teddy: a sketching interface for 3D freeform design. In: Proceedings of SIGGRAPH '99. ACM Press, Addison-Wesley, New York, Reading, MA, pp. 409–416.

Khan, A., Komalo, B., Stam, J., Fitzmaurice, G., Kurtenbach, G., 2005. Hovercam: interactive 3D navigation for proximal object inspection. In: Proceedings of SI3D '05. ACM Press, New York, pp. 73–80.

Mackinlay, J.D., Card, S.K., Robertson, G.G., 1990. Rapid controlled movement through a virtual 3D workspace. In: Proceedings of SIGGRAPH '90. ACM Press, New York, pp. 171–176.

Mackinlay, J.D., Robertson, G.G., Card, S.K., 1994. Moving viewpoint with respect to a target in three-dimensional workspace. US Patent 5,276,785.

Schmalstieg, D., Encarnação, L.M., Szalavári, Z., 1999. Using transparent props for interaction with the virtual table. In: Proceedings of SI3D '99. ACM Press, New York, pp. 147–153.

Singh, K., Grimm, C., Sudarsanam, N., 2004. The ibar: a perspective-based camera widget. In: Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology. ACM Press, New York, NY, USA, pp. 95–98.

Strauss, P.S., Carey, R., 1992. An object-oriented 3d graphics toolkit. In: SIGGRAPH '92: Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques. ACM, New York, NY, USA, pp. 341–349.

Tan, D.S., Robertson, G.G., Czerwinski, M., 2001. Exploring 3D navigation: combining speed-coupled flying with orbiting. In: Proceedings of CHI '01. ACM Press, New York, pp. 418–425.

Thorne, M., Burke, D., van de Panne, M., 2004. Motion doodles: an interface for sketching character motion. ACM Transactions on Graphics 23 (3), 424–431.

Ware, C., Osborne, S., 1990. Exploration and virtual camera control in virtual three dimensional environments. In: Proceedings of SI3D '90. ACM Press, New York, pp. 175–183.

Zeleznik, R.C., Forsberg, A.S., 1999. Unicam—2d gestural camera controls for 3D environments. In: Proceedings of SI3D'99, pp. 169–173.

Zeleznik, R.C., Herndon, K.P., Hughes, J.F., 1996. Sketch: An interface for sketching 3d scenes. In: Proceedings of SIGGRAPH 96, Computer Graphics Proceedings, Annual Conference Series, pp. 163–170.

Zeleznik, R.C., LaViola, Jr. J.J., Acevedo, Feliz, D., Keefe, D.F., 2002. Pop through button devices for VE navigation and interaction. In: VR '02: Proceedings of the IEEE Virtual Reality Conference. p. 127.