Designing Intuitive Multi-touch 3D Navigation Techniques

Damien Marchal[1], Clément Moerman[2,3], Géry Casiez[3], and Nicolas Roussel[4]

[1] CNRS
[2] Idées-3Com
[3] University of Lille and
[4] Inria Lille Villeneuve d'Ascq, France
{damien.marchal,clement.moerman,gery.casiez}@lifl.fr,
nicolas.roussel@inria.fr
http://ns.inria.fr/mint/MoveAndLook

Abstract. Multi-touch displays have become commonplace over recent years. Numerous applications take advantage of this to support interactions that build on users' knowledge and correspond to daily practices within the real world. 3D applications are also becoming more common on these platforms, but the multi-touch techniques for 3D operations often lag behind 2D ones in terms of intuitiveness and ease of use. Intuitive navigation techniques are particularly needed to make multi-touch 3D applications more useful, and systematic approaches are direly needed to inform their design: existing techniques are still too often designed in ad-hoc ways. In this paper, we propose a methodology based on cognitive principles to address this problem. The methodology combines standard user-centered design practices with optical flow analysis to determine the mappings between navigation controls and multi-touch input. It was used to design the navigation technique of a specific application. This technique proved to be more efficient and preferred by users when compared to existing ones, which provides a first validation of the approach.

Keywords: 3D navigation, multi-touch, interaction technique, design rationale.

1 Introduction

Multi-touch displays have become commonplace over the recent years. Smartphones, tablets, interactive kiosks and systems of other sorts can now detect and react to the presence of two or more contact points on the screen surface. Numerous applications take advantage of this to support *reality-based interactions* [13] that build on users' knowledge and correspond to daily practices within the real world. 3D applications are also becoming more common on these platforms, including games, virtual tours, and CAD applications for both specific, e.g. interior design, and general purposes. But the multi-touch techniques for 3D operations often lag behind the 2D ones in terms of intuitiveness and ease of use. Navigation particularly seems to be the Achilles heel of multi-touch 3D applications. Existing techniques are still too often designed in ad-hoc

P. Kotzé et al. (Eds.): INTERACT 2013, Part I, LNCS 8117, pp. 19–36, 2013.
© IFIP International Federation for Information Processing 2013

ways. Intuitive techniques are direly needed to make the applications more useful, and systematic approaches direly needed to inform their design.

The Merriam-Webster Dictionary defines intuitive as "*attaining to direct knowledge or cognition without evident rational thought and inference*". Based on an extensive review of the relevant literature, Ingram et al. also identify *direct manipulation* as the most influential factor determining the intuitiveness of multi-touch systems [10]. Direct manipulation is commonly supported by 2D multi-touch applications, due to the trivial mapping between 2D tasks and the multi-touch input space. However, finding a *direct* mapping between 3D tasks and this 2D input space is much more difficult. To clarify the requirements for intuitive 3D navigation techniques, we propose to turn first to cognitive accounts of the feeling of directness.

Hutchins et al. identify two underlying phenomena that give rise to this feeling: a small cognitive distance, and direct engagement [9]. The cognitive distance is the one "*between the user's intentions and the facilities provided by the machine*". It encompasses the semantic distance, concerned with the meaning of available interactions, and the articulatory one, concerned about their form. For the semantic distance to be small, the system should provide users with adequate commands to concisely express what they want to do. For the articulatory distance to be small, the system should provide an adequate mapping between user actions and the commands. This mapping should not be arbitrary, but should rather favor similarities between user action and command meaning. Lastly, for direct engagement, the system should provide continuous representations of the objects of interest and promptly react to user actions on them. Ultimately, the degree of directness relates to the one to which the system supports skill-based rather than rule-or knowledge-based behaviors [22].

Navigation concerns viewpoint control and is the aggregate of *wayfinding* (cognitive planning of one's route), *travel* (the motor aspects) and *inspection* (for particular proximal views of objects). The importance of each sub-task depends on the considered application. We did not consider wayfinding sub-tasks in this work. We rather focused on multi-touch support for traveling and, to a lesser extent, inspection. Travel techniques support the motor aspects of 3D navigation, allowing users to control the position and orientation of their viewpoint [2]. Viewpoints are typically modeled using seven parameters: the camera's field of view, three Cartesian coordinates (its position) and three Euler angles (its orientation). Controlling these parameters requires a rich command vocabulary because of their number and the different ways to use them. Turning around is pretty straightforward, for example, as it requires the control of a single viewpoint parameter (Figure 1, R_y). Wandering around a horizontal space requires the control of three parameters at the same time (T_x, T_z & R_y). But some navigation tasks require quite a complex coordination of controls, especially when the desired motion is conceptually tied to other reference points. Orbiting around an object, for example, couples planar circular motions with a rotation around an orthogonal axis, both centered on the object (T_y, T_z & R_x for O_x; T_z, T_x & R_y for O_y). Temporary transformations can also be useful, such as adjusting the field of view (FOV) of the camera to remotely inspect a distant place or have a closer look at a nearby detail.

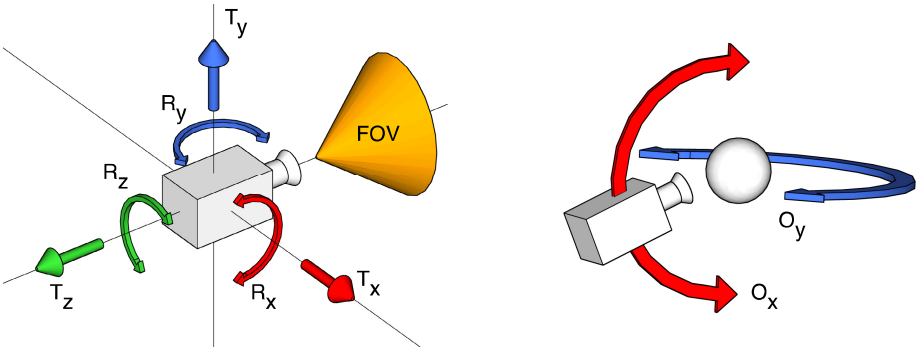

Fig. 1. Typical viewpoint controls: dolly, sidestep and fly (T_z, T_x and T_y), tilt, pan and roll (R_x, R_y and R_z), zoom (FOV) and orbit (O_x and O_y)

To reduce semantic distance, multi-touch navigation techniques should support not only elementary viewpoint controls, but also coordinated ones, including the complex coordination required by externally referenced tasks and temporary ones. To reduce articulatory distance, they should be based on a mapping favoring similarities between user gestures and commands meaning. To support direct engagement, they should provide a close and continuous visual-motor loop. In this paper, we present a methodology for designing multi-touch 3D navigation techniques that meet all these requirements. After discussing related work, we describe our methodology and explain how it was used to design the navigation technique of a particular application. We then provide some implementation details for that application. We finally report on a study that compared this technique with existing ones and provides a first validation of the approach.

2 Related Work

Navigation in virtual 3D worlds has been extensively studied in immersive and desktop environments. Bowman et al. [2] and Christie et al. [4] provide detailed reviews of the relevant concepts and techniques in these contexts, many of which are also relevant to multi-touch environments. Navigation techniques map user actions on one or more input devices to viewpoint controls such as those of Figure 1. Theoretically, a technique could allow users to operate all controls at once. However, it is rarely the case since few input devices (or device combinations) have enough degrees of freedom, and their control properties seldom match the perceptual structure of the navigation tasks [12]. Viewpoint controls thus tend to be organized in groups, which can raise issues about consistency and mode switching. The following review focuses on input-to-control mappings for multi-touch systems, but also discusses pen or mouse-based navigation techniques that could easily be adapted to these systems.

2.1 Basic Viewpoint Control

Different techniques have been proposed to freely and precisely control the viewpoint. These are typically used when navigation is a primary task of the application,

one without which it would not be the same. They most often provide only elementary input-to-control mappings to move and orient the viewpoint.

Games using a first-person perspective are probably the most popular 3D applications supporting this kind of navigation on multi-touch platforms. A common practice is to use one or two on-screen joysticks for moving and turning (R_y, T_z or $T_x, T_z + R_y, R_x$). Virtual joysticks make control grouping explicit and make it possible to use non-linear transfer functions for a trade-off between speed and control. They can also be complemented by buttons, sliders or other widgets for discrete or continuous actions on other controls (e.g. FOV, T_y). Coordinating interactions on multiple screen locations is not necessarily easy, though, especially without haptic feedback. Compound controls such as orbiting are thus usually difficult in these settings. Instead of spreading controls across the screen, some techniques combine them using modes. In the RealMyst[1] game, for example, touching the screen and moving horizontally or vertically changes the orientation of the viewpoint while holding still moves forward. Such an approach is of course only practical for a small number of modes.

Multi-touch devices can be used to interact with a 3D scene displayed elsewhere. The ability to use a different view of the scene, a different orientation, or a different physical shape offers new interesting possibilities. The *Finger Walking in Place* (FWIP) technique allows to navigate in a CAVE by mimicking walking movements with fingers on a horizontal multi-touch device [14, 15]: repeated single-touch sliding gestures move the viewpoint forward or backward (T_z), while multi-touch turn gestures rotate it left or right (R_y). The *Follow my Finger* (FmF) technique uses a horizontal multi-touch table to navigate in a scene shown on a vertical screen [1]. The table shows a 2D bird's-eye view of the scene with a camera icon that users can move (T_z, T_x) and orient (R_y) using the 2D Rotate'N Translate technique [16]. The *CubTile* [24] takes the idea of aligning the perceptual structure of the tasks with the input device in the opposite direction. This device combines 5 multi-touch surfaces arranged as a cube so that gestures performed on multiples sides at the same time define a 3D gesture that can be used for 3D interaction. Although designed for object manipulation, the CubTile may well be suitable for navigation tasks.

2.2 Viewpoint Control Facilitation

It might well be the case that the 3D environment in which a user wants to navigate is extremely large [19]. Or the user might be willing to quickly get a glimpse of the scene from different perspectives. Or (s)he might be engaged in repeated tasks requiring frequent switches between two or more viewpoints. In these situations, navigation is just a mean and not an end. One wants to transition between viewpoints but does not necessarily care about all the details of the transition. Even with sophisticated transfer functions, basic viewpoint controls are not enough: one needs faster and integrated techniques to move and orient the viewpoint.

The *Point Of Interest* (POI) desktop technique was precisely designed for rapid controlled movement through a 3D space [17]. After selecting a POI with the mouse,

[1] `http://www.cyanworlds.com/iOS_realMyst/`

it allows to quickly move there by simply pressing a key, the system taking care of the transition with an animation adjusting the viewpoint position (using a logarithmic function of the remaining distance) and reorienting it to face the POI. *UniCam* [25] is a mouse or stylus-based system that uses simple gesture recognition to facilitate a variety of complex navigation tasks including orbiting around specific points, click-to-focus on points and edges, and region zooming. *Navidget* [8] expands on these ideas by allowing the user to not only specify a point of interest but also control the final position and orientation of the viewpoint, rather than inferring them. The system uses single-stroke symbolic gestures and animations in constant time to combine travel and inspection with the ability to go back to a previous viewpoint configuration.

A difficulty when trying to control the viewpoint without any external representation (as in FmF) is that by definition, it cannot be seen. As a consequence, it can only be indirectly manipulated. There is, however, a way to change this: by giving users the impression they can grab the whole scene and manipulate it. Instead of indirectly moving the viewpoint to a particular place, for example, they would thus manipulate the whole scene so that the place comes into the viewpoint. To support this, one needs to make sure that any object touched by a finger remains under it as long as it stays in contact with the surface. This approach has its roots in Gleicher & Witkin's early work on *through-the-lens camera control* [7] and recently received renewed attention after Reisman et al. adapted it to the interactive manipulation of 3D content on 2D multi-touch systems under the name *screen-space* [23]. The *DabR* system [5] uses it in a strict way to support the direct manipulation of basic viewpoint controls, for example. A drawback of the screen-space approach is that its output (viewpoint transformation) is not always predictable due to ambiguities in potential mappings between points in screen space and the 3D scene. To avoid these ambiguities, the *Drag'n Go* technique [20] assigns viewpoint controls to input gestures based on *kinematic correspondence*, i.e. the similarity of the input and output paths [3].

Fu et al. assembled an impressive set of viewpoint control facilitation techniques more or less inspired by the above ones for exploring large-scale 3D astrophysical simulations [6]. Yet this assemblage seems quite ad-hoc. The fact is that designers have little information to rely on when creating a new application.

2.3 A Lack of Systematic Approach

Intuitive navigation techniques are needed to make multi-touch 3D applications useful. Different techniques have been proposed to support basic viewpoint controls and facilitate more complex ones. But comparing these techniques is hard, considering the little information available on their design process and performances. Although the initial design motivation is usually clearly stated in the corresponding papers, the design rationale is largely undocumented. Which decisions were made during the design process, and why, for example? How did the authors come up with the proposed mapping between user actions and viewpoint controls? Why did they decide to group them this way? Without these explanations, one might wonder if there was actually a design process. The authors of UniCam somehow acknowledge this problem: "*Our choice of how to gesturally map the 3 DOFs of camera translation to 2D mouse movements involves some apparently arbitrary choices. (...) In lieu of an*

explanation, we note that from our experience with gestural interaction, the most reliable technique for insuring usable interactions is through empirical evaluations." [25].

We collectively need more explanations on the design of these techniques. Evaluations are also important and need to be properly reported. As illustrated by Table 1, few of the techniques we reviewed have been evaluated and even less have been compared to others. We definitely need more comparative evaluations. Without explanations of what is being done and comparisons with existing solutions, there can be no progress in the understanding of the problems and their solutions. Systematic approaches are direly needed to inform the design of new techniques.

Table 1. Summary of the characteristics of the most relevant techniques and systems discussed in this section. Rows prefixed with a star correspond to those specifically designed for multi-touch interaction.

Technique or system	Design motivation	Reported Evaluation
*FWIP [14,15]	walking metaphor	comparative (vs. joystick) & usability testing
*FmF [1]	2D directness	None
POI [17]	speed and control	none
UniCam [25]	integrated suite	empirical?
Navidget [8]	ease of use and control	comparative (vs. standard 3D viewer)
*Screen-space [23]	3D directness	none
*DabR [5]	3D directness	none
*Drag'n Go [20]	multi-scale navigation	comparative (vs. POI, DabR, keyboard+mouse)
*Fu et al [6]	large scale navigation	usability testing

3 Design Methodology

Considering a set of application-specific tasks, how can one map the associated viewpoint controls to the input handles provided by a multi-touch system? In this section, we report on the design of such a mapping for a particular application. Although the resulting technique is specific to that application, we believe our design methodology should be of general interest. The application we worked on is one for reviewing interior designs (Figure 2) that typically runs on computers equipped with a multi-touch screen. Our goal was to design an intuitive navigation technique for this application, as defined earlier, i.e. one with reduced semantic and articulatory distances and a close and continuous visual-motor loop. In the following, we explain how standard user-centered practices and optical flow analysis helped us identify application controls and input handles and define the mappings between them.

3.1 Identifying Navigation Tasks and Associated Controls

A way to reduce the semantic distance *by design* is to work with users to define the navigation commands from their perspective, rather than the application developers'.

Developers usually think of viewpoint control in terms of parametric modifications of the camera model, as these are ultimately the only controls available. But they typically have little insight into the ways these should be grouped. Users, on the other hand, typically think in terms of high-level situated tasks (i.e. context-specific) that can help structure the design space. To provide users with adequate commands to concisely express what they want to do, application designers must identify these tasks.

Fig. 2. Sample interior design. Reviewing such a scene requires the ability to quickly navigate through it (including moving from one floor to the other), to orient the viewpoint in a precise way (e.g. to check the view from the couch) and to inspect particular objects (e.g. the ones on the table).

Based on our specific application context (Figure 2), the related work we previously described and interviews of potential users, we decided to focus on the following tasks, in decreasing order of importance:

- *Move around* - Users need to be able to move around the virtual interior the way they do in the real world, i.e. mostly by moving forward or backward (T_z) while possibly turning left or right (R_y). Although commonly supported by video games, sidestepping (or *strafing*,T_x) is rarely used in real world situations and thus of lesser importance. Altitude control (T_y) is also pretty limited in the real world without assistance, and thus also of lesser importance.
- *Look around* - Adjusting the viewpoint orientation is another important task that must be supported by the considered application. Users need to be able to look left and right (R_y) as well as up and down (R_x). The third rotation of the camera (R_z) does not seem necessary as people have limited control on it in the real world and it does not change what they see but only how they see it.
- *Circle around* P - When focusing on a particular object or area, users need to be able to look at it from different sides. This is typically achieved by orbiting around a previously specified point (P) in the horizontal plane (O_y).
- *Scrutinize* P - Looking at a particular point of interest (P) from different sides might not be enough. One might want to have a closer look at it. In real-life, one can bend over or use optical tools such as a magnifying glass or binoculars to temporarily modify one's field of view (FOV).

Table 2 summarizes the viewpoint controls of Figure 1 associated with the above four tasks to be supported by our particular application. Having identified the viewpoint model's degrees of freedom we want to control and taken a few first steps into their organization, we must now turn to the input device (the multi-touch screen) and examine the handles it provides for that control.

Table 2. Relevance of viewpoint controls to high-level navigation tasks, by decreasing order of importance.●: relevant,◐: partially relevant, ○: not relevant.

	T_x	T_y	T_x	R_x	R_y	FOV	O_y
Move around	◐	◐	●	○	●	○	○
Look around	○	○	○	●	●	○	○
Circle around P	○	○	○	○	○	○	●
Scrutinize P	○	○	○	○	○	●	○

3.2 Identifying Input Handles

Although some touch sensing technologies provide rich information about contact regions, including their shape or the applied force, most multi-touch APIs only expose the 2D coordinates of their centroid. One might thus think that using n fingers, users should be able to control $2 \times n$ degrees of freedom. But in reality, it is never the case. Multi-touch systems can't distinguish between fingers, so degrees of freedom cannot be univocally associated to them. The order of appearance of contacts or hit-testing with specific on-screen areas can be used for these associations. But in the end, interaction will always be constrained by limited finger individuation: it is quite difficult to move one finger without some degree of involuntary movement at one or more of the others [11].

Instead of considering contacts individually, one can group them using different methods (e.g. hit-testing, proximity, hand identification) and extract from the collated movement information global parameters to be associated with degrees of freedom to control. A common practice is to consider multi-touch gestures on objects as *Rotate-Scale-Translate* (RST) manipulations and to determine and characterize the principal transformation involved - e.g. $R(\alpha, C)$ for a *turn* gesture, $S(\kappa, C)$ for a *pinch* or a *spread*, and $T(x, y)$ for a *swipe*[2]. Figure 3 shows a simplified view of the state machine we used, based on this approach. The machine differentiates four interaction states (shown in gray): one for single-touch interaction, and three differentiating multi-touch interactions based on the first principal transformation detected.

Having described the desirable viewpoint controls for our application (Table 2) and the different handles provided by a multi-touch screen, i.e. $T(x, y)$ for single-touch interactions and $R(\alpha, C)$, $S(\kappa, C)$ & $T(x, y)$ for multi-touch ones, we will now examine the mappings between them.

[2] The parameters associated to each transformation will be explained in the next section.

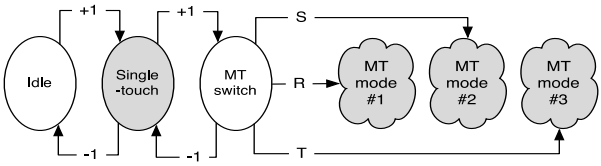

Fig. 3. Simplified view of a multi-touch state machine based on an RST classifier. +1 and -1 transitions are triggered by contact addition and removal. The R, S and T transitions are triggered based on the first principal transformation detected. Note that once in mode #1, #2 or #3, multi-touch interaction does not have to be restricted to the transformation initially detected.

3.3 Choosing the Right Mappings

A way to reduce the articulatory distance and ensure tight coupling between perception and action is to choose the mappings between viewpoint controls and multi-touch gestures so that contact trajectories match the scene transformations caused by viewpoint modifications. To achieve this, we created video clips illustrating the effect of the 7 viewpoint controls of Table 2 on a particular scene. We used OpenCV to compute and visualize the optical flow of each of these videos (Figure 4). We then compared these flows to those corresponding to multi-touch RST manipulations, i.e. $T(x, y)$ when using a single finger and $R(\alpha, C)$, $S(\kappa, C)$, $T(x, y)$ otherwise.

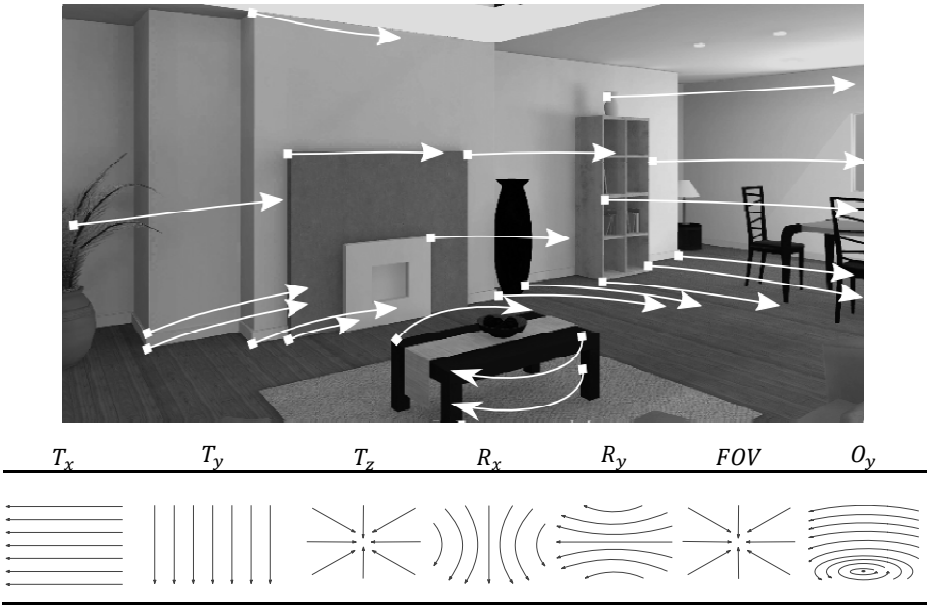

Fig. 4. Top image: optical flow computed over time with OpenCV when circling the viewpoint along the vertical axis (R_y). Bottom images: stylized renderings of the flows corresponding to the 7 viewpoint controls of Table 2 (arrows could all point in the opposite direction).

Table 3 summarizes the compatibility between gesture flows and optical flows using a three-level scale. Starting from this table, we applied the following heuristics to choose between alternative mappings:

- As MT-$R(\alpha, C)$ is the most compatible gesture with O_y, we decided to map the two.
- T_z and FOV are compatible with the same gestures, MT-$S(\kappa, C)$ and ST-$T(x, y)$. Since T_z is one of the most important controls, we wanted to keep it as simple as possible and thus preferred a single-touch gesture for it. Moving forward/backward seemed better matched with a vertical movement rather than a horizontal one, so we chose ST-$T(., y)$. For FOV, we chose MT-$S(\kappa, C)$.
- For ST-$T(x, .)$, we were left with T_x, R_y and O_y, the first two being more important than the third one. We decided to map ST-$T(x, .)$ to R_y so that single-touch interaction would support both *Move around* (with T_z and R_y) and *Look around* (with R_y).
- For MT-$T(., y)$, we were left with T_y and R_x. We chose the latter, as looking up/down was considered more important than controlling one's altitude.
- For MT-$T(x, .)$, we were left with T_x and R_y. Informal tests convinced us that the latter was preferable, considering our previous choice of R_x for MT-$T(., y)$.

Table 3. Compatibility between gesture flows (rows) and optical flows (columns): ○ incompatible, ● compatible, ◐ compatible under one of the conditions below. A dot in place of x or y indicates that this component is ignored. The gray cells correspond to the chosen mapping.

		T_x	T_y	T_z	R_x	R_y	FOV	O_y
ST	$T(x, .)$	●	○	●	○	◐(1)	●	◐(2)
	$T(., y)$	○	●	●	◐(3)	○	●	○
MT	$R(\alpha, C)$	○	○	○	◐(4)	◐(5)	○	●
MT	$S(\kappa, C)$	○	○	●	○	○	●	○
MT	$T(x, .)$	●	○	◐(6)	○	◐(1)	◐(6)	◐(2)
	$T(., y)$	○	●	◐(6)	◐(3)	○	◐(6)	○

(1) compatibility is inversely proportional to the vertical distance to the center of the screen
(2) the point of interest (P) must have been previously specified
(3) compatibility is inversely proportional to the horizontal distance to the center of the screen
(4) C must be in the middle of a vertical border of the screen, i.e. left or right
(5) C must be in the middle of a horizontal border of the screen, i.e. top or bottom
(6) contacts must be "close enough", i.e. within a certain radius, so they can be reduced to single-touch interaction

4 Implementation: The Move and Look Technique

A close look at Table 2 and the chosen mapping in Table 3 shows that ST-$T(x, y)$ corresponds to *Move around* while MT-$R(\alpha, C)$ corresponds to *Circle around*, MT-$S(\kappa, C)$ to *Scrutinize* and MT-$T(x, y)$ to *Look around*. As illustrated by Figure 5, each of our 4 high-level navigation tasks can thus be associated to an interaction state of the machine shown in Figure 3. This section details the implementation of the resulting navigation technique, which we called *Move&Look*.

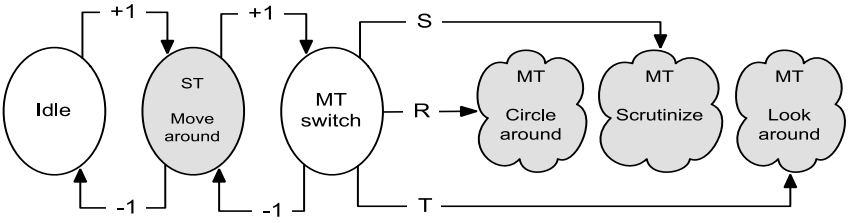

Fig. 5. The Move&Look technique, instanciated from Figure 3

4.1 Single-Touch Interaction: *Move Around*

When a single contact is detected, subsequent $T(x,y)$ finger movements are mapped to R_y, T_z camera movements to support the *move around* task. When touch is detected, a ray is casted in the 3D scene from the camera center through the contact point in the camera plane. The intersection with the scene (P) defines the point of interest, and the ray a path towards it. Progression along the path is controlled through finger movements in the following way:

- Lateral movements ($T(x,.)$) do not affect the camera position. Proximal finger movements translate the camera towards P and distal movements translate it backwards along the path ($T(.,y)$).
- The distance between the initial contact point and the bottom of the display is mapped to the entire path length: the destination is reached when the finger reaches the bottom of the display.
- Distal finger movements past the initial contact point (i.e. above it) continue moving the camera backwards along the path. For consistency, the scale factor remains the same as when closing in on P.

Users can turn the camera left and right (R_y) through lateral finger movements. The camera orientation is computed so as to always keep the projection of P under the finger. R_y is computed either analytically [7] or numerically [23] to minimize the distance between the previous projection of P and the current finger position (we used the minimizer from ALGLIB to solve the different minimization problems).

4.2 Multi-touch Interaction Switch: RST Classifier

When multiple contacts are detected, their movement is analyzed to determine whether the state machine should switch to *circle around*, *scrutinize* or *look around*. The movement of the n contact points is interpreted as a rigid transformation combining a rotation $R(\alpha, C)$, an homogeneous scaling $S(\kappa, C)$, and a translation $T(x,y)$. The initial position of the contact points is noted r_i and their current position c_i. The R, S and T transformations correspond to the minimization of the cost function F defined by Equation 1. $T(x,y)$ is first computed from the centroids of the initial set of contact points (C_i) and the current one (C_c) according to equations 2, 3 and 4. The rotation angle α is the one that minimizes the cost function of Equation 5 and is computed

using Equation 6 where $co = c_i - C_i$ and $ro = r_i - C_c$. The scale factor κ is similarly the one that minimizes the cost function of Equation 7 and is computed using Equation 8 where $cr = R^{-1}(\alpha).c$:

$$F(x, y, \alpha, \kappa) = \sum_{0 \le i \le n} \|T(x, y)R(\alpha)S(\kappa)r_i - c_i\| \quad (1)$$

$$C_i = \frac{1}{n} \sum_{0 \le i \le n} r_i \quad (2)$$

$$C_c = \frac{1}{n} \sum_{0 \le i \le n} \mathrm{c_i} \quad (3)$$

$$T(x, y) = C_c - C_i \quad (4)$$

$$G(\alpha) = \sum_{0 \le i \le n} \|R(\alpha)(r_i - C_i) - (c_i - C_c)\| \quad (5)$$

$$\alpha = \operatorname{atan} 2\left(- \sum_{0 \le i \le n} co_x ro_y - co_y ro_x \, , \sum_{0 \le i \le n} co_y ro_y + co_x ro_x \right) \quad (6)$$

$$H(\kappa) = \sum_{0 \le i \le n} \|S(\kappa)ro - R^{-1}(\alpha)co\| \quad (7)$$

$$\kappa = \frac{cr_x^2 + cr_y^2}{cr_x ro_x + cr_y ro_y} \quad (8)$$

The only rotations considered in these equations are those centered on the centroid of the contact points. Although we perceive it as an elementary rotation, moving one's index finger around one's thumb while keeping this one steady will thus be interpreted as a combination of a centroid-centered rotation and a translation. To tackle this problem, we weight all contact positions by the inverse of their traveled distance when computing C, the center of both the rotation $R(\alpha, C)$ and the homogeneous scaling $S(\kappa, C).T(x, y)$ is also adjusted by removing the displacement possibly introduced by $R(\alpha, C)$.

The x, y, C, α and κ parameters resulting from the above computations are used to determine the prominent gesture among *Rotate*, *Swipe* and *Pinch*. Our classifier considers one model $M(t)$ for each gesture type and returns the one that better fits the observations (highest R^2 value). The models map the initial configuration r_i to an estimated state $\tilde{c_i}$ (Equation 9). The estimated error (the residual sum of squares) and the coefficient of determination R^2 are then computed using Equations 10, 11 and 12.

$$\tilde{c_i} = M(t).r_i \quad (9) \qquad R^2 = 1 - \frac{SS_{res}}{SS_{tot}} \quad (10)$$

$$SS_{res} = \sum_{0 \le i \le n} \|c_i - \tilde{c_i}\|^2 \quad (11) \qquad SS_{tot} = \sum_{0 \le i \le n} \|r_i - c_i\|^2 \quad (12)$$

Our classifying method is similar to the *GestureMatching* method used by Nacenta et al. [21], but instead of classifying the combined contribution of *Rotate*, *Swipe* and *Pinch* gestures, ours allows to classify the contribution of individual gesture types. Our method requires enough information to properly work. The classifier is thus enabled only when the summed distance covered by the contact points is beyond a given threshold. Based on preliminary tests, we found that a value of 10 pixels on a 90PPI display (around 2.8 mm) provides a good trade-off between latency and success rate, which is in agreement with other thresholds reported in the literature for similar applications [25].

4.3 Multi-touch Gestures: Circle Around, Look Around and Scrutinize

When the classifier detects a prominent *Rotate* gesture, the technique enters the *circle around* state of Figure 5 until all contacts are lost. $R(\alpha, C)$ provides the pivot point to rotate the scene and the angle of rotation (the center of the 3D rotation is computed from the projection of C in the 3D scene).

When a *Swipe* gesture is detected, the technique enters the *look-around* state until all contacts are lost. $T(x, y)$ is then used to rotate the camera (R_x, R_y) in a way similar to *move around*, but with two degrees of freedom instead of one.

When a *Pinch* gesture is detected, the technique enters the *scrutinize* state until all contacts are lost. The scale factor of $S(\kappa, C)$ is used to adjust the FOV of the camera. To ensure smooth camera movements, its look-at point remains fixed while contacts are moving. The FOV is restored to its initial value when all contacts are lost. This state further supports remote inspection by using $T(x, y)$ to rotate the camera (R_x, R_y), as in the *look-around* state.

5 Experiment

Our main motivation in this experiment was to assess our design choices by comparing Move&Look to other techniques from the literature (Screen-space [23], DabR [5] and Drag'n Go [20]) or available in commercial products (Virtual joysticks and the RealMyst technique described above), most of which have never been evaluated nor compared.

5.1 Task

Informal user testing with Move&Look suggested the technique was particularly efficient for interior designs mainly consisting of flat orthogonal surfaces. Encouraged by this, we wanted to assess the effectiveness of the technique in a more demanding environment. The task we chose consisted in collecting spheres placed inside boxes in an outdoor environment, and dropping them in a fountain at the center of the scene (Figure 6). To provide a fair comparison between multi-scale navigation techniques (Drag'n Go and Move&Look) and the others and focus on the evaluation of camera movements, the boxes were not positioned far away from each other but close to the central drop zone.

Participants had to find the boxes in the scene. For each box, they had to position the camera in front of its only open face to pick up the sphere it contained. A sphere

would turn from red to green when the camera was close enough to indicate one could touch it to pick it up. Participants could carry only one sphere at a time, and it was automatically dropped once in the drop zone. Collision detection prevented participants from moving through objects, and a trial was considered as fully completed after all the spheres had been dropped. Participants were instructed to perform this as quickly as possible. They could ask the experimenter to reset the camera to its initial position or withdraw a trial if they felt unable to complete it. They were not encouraged to do so, however. The experimenter rather encouraged them to finish a trial if he felt they could succeed.

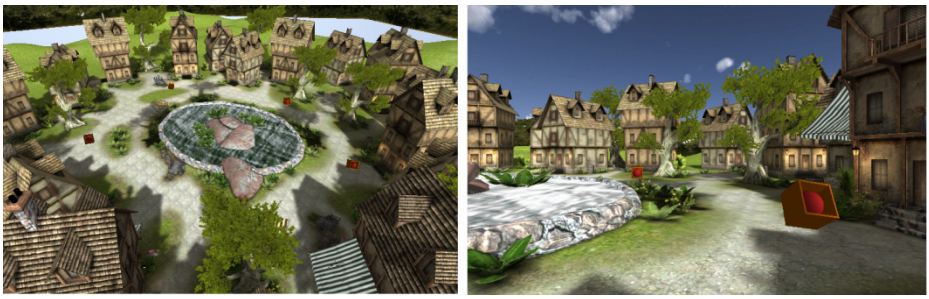

Fig. 6. Left: overview of the 3D environment used in the experiment. Right: detailed view showing a box containing a sphere to pick up and drop in the fountain.

5.2 Participants

Twelve unpaid male participants with a mean age of 35 (SD=14) served in the experiment. Five of them used a computer on a daily basis, played 3D video games and were familiar with touch-screens through mobile phones or tablets. Seven of them had a low experience with video games and were novice with touch interfaces.

5.3 Apparatus, Design and Procedure

Participants were seated in front of a 22" 3M multi-touch screen orientated at an angle of about 70° from a horizontal desk. The experiment was implemented using Unity 3.5[3]. A repeated measures within-subjects design was used. The independent variable was the interaction technique (TECH) with six levels: DabR, Screen-space, Drag'n Go, RealMyst (a custom implementation of the RealMyst technique), Virtual joysticks (a standard combination of two Unity virtual joysticks displayed at fixed positions) and Move&Look. A trial consisted in collecting 4 spheres and each technique was evaluated with 3 successive trials (TRIAL). In summary the experimental design was: 12 participants × 6 TECH × 3 TRIAL = 212 total trials.

The presentation order for TECH was counter-balanced across participants using a balanced Latin Square design. To favor expert usage and a fair comparison between techniques, each was first introduced by the experimenter with a demo and then a

[3] http://unity3d.com/unity/whats-new/unity-3.5

training session. Participants could also use a cheat sheet throughout the experiment. After each technique, participants filled a questionnaire inspired by the Nasa TLX test and at the end of the experiment, they were asked to rank the techniques and give additional feedback.

5.4 Results

The dependent variables were the completion time, the number of give-ups and camera resets, and the qualitative results.

Numbers of Give-Ups and Camera Resets — 25% of trials were aborted for Screen-space, 17% for RealMyst, 8% for Virtual joysticks and 0% for Drag'n Go, DabR and Move&Look. The camera was reseted in 66% of all trials for Screen-space, 14% for Virtual joysticks, 11% for RealMyst, 3% for DabR and Move&Look, and 0% for Drag'n Go.

Task Completion Time — Task completion time is defined as the time needed to successfully collect the four spheres and drop them in the fountain. Trials where participants gave up were removed for the analysis. Trials at least three standard deviations away from the mean for each TECH condition were considered as outliers and also removed. A repeated measures ANOVA showed a significant effect of TECH ($F_{5,55} = 9.9, p < 0.001$). Subsequent pairwise comparison showed significant differences ($p < 0.005$) between Drag'n Go and Screen-space, Drag'n Go and RealMyst, Move&Look and Screen-space, and Move&Look and RealMyst. No significant difference was found between Drag'n Go and Move&Look. Completion times were $97s$ for DabR, $184s$ for Screen-space, $60s$ for Drag'n Go, $117s$ for RealMyst, $111s$ for Virtual joysticks and $72s$ for Move&Look.

User Ranking and Questionnaire — The participants ranked the techniques in decreasing order of preference. Overall, Move&Look came first (10 participants ranked it first and 2 ranked it second) followed by Drag'n Go, DabR, Virtual joysticks, RealMyst and Screen-space. The participants who ranked Move&Look first explained it nicely complements Drag'n Go as it allows to control more degrees of freedom while keeping the navigation intuitive: it does not require focusing on the gestures to execute nor does it require planning a trajectory in the scene to reach a target. Screen-space was ranked last considering its lack of intuitiveness: in spite of the frequent use of the cheat sheet the participants did not understand how to effectively use the technique to navigate the way they wanted (we believe the important semantic distance explains this gap between users' intentions and the system's behaviours). These subjective results are in agreement with the quantitative results found for completion time and the numbers of give-ups up and camera resets.

After each technique, the participants answered questions related to the following six criteria on a 5 point Likert scale: mental demand, physical demand, performance, effort, frustration and satisfaction. The questions asked were similar to the ones available in the Nasa TLX test. We ran a Friedman analysis with Bonferroni-corrected Wilcoxon post-hoc analyses. This analysis shows significant differences between the techniques for all criteria, especially for the techniques at the bottom of the participants' ranking. Table 5 summarizes the significant differences that were found.

Table 4. Details of the post-hoc analysis for cases where one or more significant differences were found (●: significant difference, ○: non significant difference)

	Mental demand	Physical demand	Perfor-mance	Effort	Frustra-tion	Satisfac-tion
Move&Look-Screen-space	●	●	●	●	●	●
Move&Look-RealMyst	●	●	●	○	●	●
Drag'n Go-Screen-space	●	●	●	●	○	●
Drag'n Go-RealMyst	●	●	●	○	○	○
Drag'n Go-DabR	○	●	○	○	○	○
RealMyst-Virtual joysticks	○	●	○	○	○	○
RealMyst-DabR	○	○	○	○	●	○
Screen-space-DabR	●	○	○	○	●	●
Screen-space-Virtual joystick	●	●	○	○	○	●

User Feedback and Observations — During the experiment we encouraged the participants to "think aloud" and freely comment on the interaction techniques. Comments were overall in agreement with the user ranking.

Screen-space received the most negative critics. All participants repeatedly reported their frustration with this technique. They felt out of control and found the mappings between fingers and camera movements inconsistent. The finger movements corresponding to different screen-space controls can indeed be quite similar, as illustrated by the optical flows of Figure 4 (e.g. T_x and R_y). The output of the screen-space solver is also strongly influenced by the picked point in the 3D scene, and thus by the geometrical shape of the underlying objects. Lastly, the movements to execute in order to move forward (T_z) and to zoom (FOV) depend on whether the initial contact point is above or below the invisible horizon (in the former case, one has to move up, in the latter, one has to move down). All these reasons probably contribute to the fact that users were not able to anticipate camera motions. The comparison of Screen-space to other interaction techniques in 3D manipulation tasks corroborates these observations [18].

Participants found DabR, Virtual joysticks and RealMyst either too slow or too fast. We hypothesize this was caused by the use of transfer functions not specifically tuned for the particular 3D environment we used: long distances took too much time to travel while participants traveled too fast on short distances. Participants found the Virtual joysticks to be less fatiguing. We hypothesize this was due to the use of rate control, which reduces physical movements. Participants reported an important fatigue when using DabR and complained they had to pay attention to the number of fingers they used. They complained about the delay introduced by the time-based mode switch used by RealMyst and the fact that the traveling direction is not towards the selected point but along the T_z axis of the camera. Drag'n Go was particularly appreciated for its ability to quickly reach distant targets, but moving to a box while orienting the viewpoint in order to pick the sphere was found more difficult and required some planning. This was not reported as a problem with Move&Look.

6 Conclusion and Future Work

In this paper, we proposed an original methodology based on user-centered practices and optical flow analysis to address the problem of designing intuitive multi-touch navigation techniques for 3D environments. User-centered practices allow to define the navigation commands from the user's perspective while the optical flow analysis provides guidelines for defining intuitive multi-touch gestures to perform these commands. We instantiated this methodology for tasks articulated around the review of interior designs, which led to the design of a new interaction technique, Move&Look. The comparison of this technique to state of the art ones in a controlled experiment showed its overall superiority and revealed usability problems with the others. These results provide a first validation of the proposed design methodology. The methodology should be applied in other navigation contexts in order to further assess its effectiveness. The robustness of the proposed RST classifier should be formally evaluated, and it can certainly be improved. Even if participants did not complain about it, we observed them flattening their rotation gestures for the *circle around* command, probably because they unconsciously followed the corresponding optical flow. Our classifier could be modified to better take into account this oval shape, for example.

References

1. Ajaj, R., Vernier, F., Jacquemin, C.: Follow my finger navigation. In: Gross, T., Gulliksen, J., Kotzé, P., Oestreicher, L., Palanque, P., Prates, R.O., Winckler, M. (eds.) INTERACT 2009. LNCS, vol. 5727, pp. 228–231. Springer, Heidelberg (2009)
2. Bowman, D.A., Kruijff, E., LaViola, J.J., Poupyrev, I.: 3D user interfaces: theory and practice. Addison-Wesley/Pearson Education (2003)
3. Britton, E.G., Lipscomb, J.S., Pique, M.E.: Making nested rotations convenient for the user. In: Proc. of SIGGRAPH 1978, pp. 222–227. ACM (1978)
4. Christie, M., Olivier, P.: Camera control in computer graphics: models, techniques and applications. In: ACM SIGGRAPH ASIA 2009 Courses, pp. 3:1–3:197. ACM (2009)
5. Edelmann, J., Schilling, A., Fleck, S.: The DabR - a multitouch system for intuitive 3D scene navigation. In: Proc. of 3DTV-CON, pp. 1–4 (2009)
6. Fu, C.W., Goh, W.B., Ng, J.A.: Multi-touch techniques for exploring large-scale 3D astrophysical simulations. In: Proc. of CHI 2010. pp. 2213–2222. ACM (2010)
7. Gleicher, M., Witkin, A.: Through-the-lens camera control. In: Proc. of SIGGRAPH 1992, vol. 26, pp. 331–340 (1992)
8. Hachet, M., Decle, F., Knodel, S., Guitton, P.: Navidget for easy 3D camera positioning from 2D inputs. In: Proc. of 3DUI 2008, pp. 83–89. IEEE (2008)
9. Hutchins, E.L., Hollan, J.D., Norman, D.A.: Direct manipulation interfaces. Human-Computer Interaction 1(4), 311–338 (1985)
10. Ingram, A., Wang, X., Ribarsky, W.: Towards the establishment of a framework for intuitive multi-touch interaction design. In: Proc. of AVI 2012, pp. 66–73. ACM (2012)
11. Ingram, J.N., Körding, K.P., Howard, I.S., Wolpert, D.M.: The statistics of natural hand movements. Experimental Brain Research 188(2), 223–236 (2008)
12. Jacob, R.J.K., Sibert, L.E., McFarlane, D.C., Mullen Jr., M.P.: Integrality and separability of input devices. ACM ToCHI 1(1), 3–26 (1994)

13. Jacob, R.J., Girouard, A., Hirshfield, L.M., Horn, M.S., Shaer, O., Solovey, E.T., Zigelbaum, J.: Reality-based interaction: A framework for post-wimp interfaces. In: Proc. of CHI 2008, pp. 201–210. ACM (2008)
14. Kim, J.S., Gračanin, D., Matković, K., Quek, F.: iPhone/iPod Touch as input devices for navigation in immersive virtual environments. In: Proc. of VR 2009, pp. 261–262. IEEE (2009)
15. Kim, J.-S., Gračanin, D., Matković, K., Quek, F.: Finger walking in place (FWIP): A traveling technique in virtual environments. In: Butz, A., Fisher, B., Krüger, A., Olivier, P., Christie, M. (eds.) SG 2008. LNCS, vol. 5166, pp. 58–69. Springer, Heidelberg (2008)
16. Kruger, R., Carpendale, S., Scott, S.D., Tang, A.: Fluid integration of rotation and translation. In: Proc. of CHI 2005, pp. 601–610. ACM (2005)
17. Mackinlay, J.D., Card, S.K., Robertson, G.G.: Rapid controlled movement through a virtual 3D workspace. In: Proc. of SIGGRAPH 1990, pp. 171–176. ACM (1990)
18. Martinet, A., Casiez, G., Grisoni, L.: Integrality and separability of multitouch interaction techniques in 3D manipulation tasks. IEEE TVCG 18(3), 369–380 (2012)
19. McCrae, J., Mordatch, I., Glueck, M., Khan, A.: Multiscale 3D navigation. In: Proc. of I3D 2009, pp. 7–14. ACM (2009)
20. Moerman, C., Marchal, D., Grisoni, L.: Drag'n Go: Simple and fast navigation in virtual environment. In: Proc. of 3DUI 2012, pp. 15–18. IEEE (2012)
21. Nacenta, M.A., Baudisch, P., Benko, H., Wilson, A.: Separability of spatial manipulations in multi-touch interfaces. In: Proc. of GI 2009. pp. 175–182. Canadian Information Processing Society (2009)
22. Rasmussen, J.: Skills, rules, and knowledge; signals, signs, and symbols, and other distinctions in human performance models. IEEE Transactions on Systems, Man and Cybernetics 13(3), 257–266 (1983)
23. Reisman, J.L., Davidson, P.L., Han, J.Y.: A screen-space formulation for 2D and 3D direct manipulation. In: Proc. of UIST 2009, pp. 69–78. ACM (2009)
24. de la Rivière, J.B., Kervégant, C., Orvain, E., Dittlo, N.: CubTile: a multi-touch cubic interface. In: Proc. of VRST 2008, pp. 69–72. ACM (2008)
25. Zeleznik, R., Forsberg, A.: UniCam - 2D gestural camera controls for 3D environments. In: Proc. of I3D 1999, pp. 169–173. ACM (1999)