Multimed Tools Appl (2014) 73:939–959
DOI 10.1007/s11042-012-1127-8

Improving 3D navigation techniques in multiscale environments: a cubemap-based approach

Daniel Ribeiro Trindade · Alberto Barbosa Raposo

Published online: 22 May 2012
© Springer Science+Business Media, LLC 2012

Abstract Navigation in virtual 3D environments, especially those with multiscale features, is still a problem for many users. In this regard, a good design of the navigation interfaces is critical to ensure that the users navigate with the best possible efficiency and comfort. In this paper, we present improvements for two well-known interfaces: *fly*, including support to collision treatment and automatic navigation speed adjustment in relation to scale, and *examine*, with automatic pivot point. Such techniques are based on the *cubemap* structure, which provides information about the surrounding environment at each instant. Usability tests with both beginner and advanced users revealed that the proposed techniques created a significant improvement in the execution of navigation tasks and a reduction in navigation errors.

Keywords Navigation · Multiscale environments · Cubemap · Interaction techniques

1 Introduction

With the development of new technologies and the increasing processing power of computers, larger and more detailed 3D virtual environments are becoming more common. Accordingly, several navigation tools were created to help users to explore these environments. However, despite the efforts of researchers, navigation in virtual environments is still problematic for many people.

D. R. Trindade · A. B. Raposo (✉)
Tecgraf/Department of Informatics, Pontifical Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil
e-mail: abraposo@tecgraf.puc-rio.br

D. R. Trindade
e-mail: danielrt@tecgraf.puc-rio.br

There are a number of reasons for this. For example, when using a navigation tool, difficulty can arise from either a users' lack of knowledge about the tools' usage or function, or poor design of the tool itself [6]. Another frequent problem is related to the type of environment being explored. Multiscale virtual environments [15], for instance, often require methods to allow the users to navigate through the different scales of the environment. Multiscale environments provide information in different levels of detail, ranging from a single screw to an oil field spanning dozens of miles.

Our goal is to propose techniques that help users, especially novice ones, navigate through multiscale environments. The solutions proposed should satisfy two main requirements: a) maximum automation, therefore demanding minimal intervention by the user, and b) independence from the type of model being viewed, so that new types of objects can be viewed in the future without the need to develop new solutions. Moreover, the solutions are intended to work on desktop setups, using mouse and keyboard, as this setup is more common to most users [16].

In this paper, we improve on two well known navigation techniques. In the case of the *fly* technique, we added collision support and automatic speed adjustment in relation to the scale; in the *examine* technique we added a way to automatically determine the pivot point.

This paper is organized as follows. In Section 2 we present related work. The cubemap concept is presented in Section 3. In Section 4, the navigation techniques are described, and in Section 5 the results of usability tests are presented. Section 6 concludes this article.

2 Related work

Navigation in virtual environments has been studied by many researchers. The main goal of most of these studies is to propose and analyze techniques that facilitate the task of exploring such environments. As an example, Ware and Osborne [19] proposed three different navigation tools: *eyeball in hand*, *scene in hand*, and *flying vehicle control*. They concluded that none of these three techniques by thenselves are able to fulfill all user needs. The same conclusion was reached by Fitzmaurice et al. [6], who analyzed the behavior of some users of 3D applications and introduced techniques to improve the interface of some tools. Also seeking to propose new interaction solutions, Mackinlay et al. [13] introduced the *point of interest (POI)* technique, in which the user must first choose a destination point and then the camera starts to move toward the chosen point, changing its speed in relation to the remaining distance. Tan et al. [17] proposed a hybrid solution in which the *fly* and *examine* tools are combined into a single interaction technique.

Multiscale environments were first introduced in the literature by Perlin and Fox [15] and Bederson et al. [3], through the creation of the *Pad* and *Pad++* interfaces. These developments have been the focus of many researchers [7, 9, 22]. To give users a sense of scale of the environment, Glueck et al. [8] provides a grid reference. This is modified consistently with the camera position: as the camera gets closer, the grid is subdivided in more pieces. As the scale of the grid is known, it is possible to identify the scale of the objects by comparing these with the grid. Besides the grid reference, they also provides position pegs, which consist in projecting the object on the grid. Through the size of this projection and a connection between this

and the grid, it is possible to identify the relative distance of objects. Although they provide information that helps in identifying the scale, that is not used to modify parameters related to navigation. Kopper et al. [12] presented a system that allows the user to navigate through the different scales of an environment. The hierarchy of the scene models is used to define the *levels of scale*. However, with this approach the user is forced to issue a command to change between scales. Moreover, the adjustment of navigation parameters is only made at time of the command. Thus, one can say that the method proposed by Kopper et al. [12] is discrete in the sense that the different scale levels are well defined regarding their form and their location in the hierarchy of the scene. By contrast, the works presented by Ware and Fleet [20] and McCrae et al. [14] make continuous adjustments to the navigation parameters. Therefore the virtual environments do not require a well defined hierarchy of levels of scale. Ware and Fleet [20] proposed adjusting the navigation speed of the *fly* tool by using the depth information present in the *Z buffer*. To this effect, they use the smallest value resulting from scanning 15 lines of the *Z buffer*. McCrae et al. [14] constructed a representation of the environment called *cubemap*, which provides information that allows the adjustment of parameters such as speed and clipping planes, as well as offering collision support.

To deal with collisions, Baciu et al. [1, 2] presented image space based techniques which consist of projecting the geometry onto an image using graphics hardware and then checking for interference, usually by analyzing the *Z buffer*. Calomeni and Celes [4] proposed the construction of a connectivity graph in preprocessing time. Through this graph, they are able to move the camera in order to avoid collisions with the environment. This concept was also explored by Xiao and Hubbold [21], who used force fields rather than a connectivity graph. Nonetheless, a preprocessing stage is still required to compute the fields. McCrae et al. [14] proposed using a new structure, which they call a *cubemap*, to help construct the force map with higher resolution while eliminating the preprocessing stage altogether.

Some of the problems related to the pivot point were identified by Fitzmaurice et al. [6] . In their work, the pivot point is chosen manually by the user via a menu option. The pivot point is also drawn on the screen, which allows, according to the authors, the users to have a better understanding of the behavior of the *examine* tool. To inspect objects, they use a widget called *ViewCube* [11]. This is a cube drawn on the corner of the screen and the object is rotated by means of this cube. Khan et al. [10] developed a new tool for object inspection called *hovercam*. When far from the object, this tool works like an *examine*. When close, it has the behavior of a pan operation, making the camera to move parallel to the faces of the object. This is done by continuous updating the pivot point to the nearest point of the scene. Unlike Fitzmaurice et al. [6], the choice of the pivot is automatic on *hovercam*. When the use of hovercam begins, the camera is reoriented toward the location of the nearest point. However, this may conflict with the user intent of inspecting a specific object. For example, in a scene containing several objects, the camera could be reoriented to an object wich is not being visualized (because it is the closest one), when in fact the user's intent was to inspect another object that, although further, was in its angle of vision. Moreover, their first implemented of hovercam required the pre-computation of a sphere-tree structure in order to determine the closest point. This was resolved in a second version, when McCrae et al. [14] reimplemented the tool using the cubemap determine the closest point.

The techniques introduced in the present paper are also based on the cubemap. However, the approach taken to construct the cubemap and the navigation techniques based on it had to be modified in order to fulfill the requirements of our applications, as will be described in the following sections.

3 The cubemap

The purpose of the cubemap, as proposed by McCrae et al. [14], is to provide information about the virtual environment at a given moment. Given a camera position, this structure is constructed from 6 rendering passes, each in a different direction in order to cover the whole environment. The FOV of the camera is 90°, therefore the combination of the 6 resulting frustums yields a cube. At each pass, a shader is used to calculate the distance from the fragment generated to the camera. The computed distance values are normalized in relation to the *near* and *far* values, and stored in the *alpha* channel of the positions related to the fragments. Rendering is made in 32-bit float images. This procedure is performed at each frame, or each time the camera position changes. The image resolution used for rendering does not have to be high, since only an estimate needs to be obtained.

The cubemap construction process we implemented differs from the one described by McCrae et al. [14] in two aspects: the orientation of the cubemap is the same as that of the camera (McCrae et al. always used canonical directions); and the *RGB* channels of the images in the cubemap store a unit vector pointing from the position of the generated fragment to the camera. The first change was necessary to simplify the process of obtaining the distance from the viewer to the center of the screen, which will be important for the automatic speed adjustment techniques of *fly* (Section 4.1) and for the automatic definition of the pivot point in *examine* (Section 4.3). The second change simplifies the construction of the force map to detect collisions (Section 4.2).

4 Navigation techniques

In this section we will present the three improved navigation techniques.

4.1 Fly with automatic speed adjustment

Navigation speed is related to the scale of the environments to be explored. Larger environments require faster speeds, while the opposite is more convenient on smaller scales.

In several applications, the scale of the virtual world does not change much and is well known, allowing a fixed navigation speed to be used. This is the case for many games, for instance. Multiscale environments, however, require a way to estimate the current scale in order to adjust the navigation speed accordingly.

McCrae et al. [14] use *minDist*, the minimum distance in the cubemap, as an estimation to determine the current scale the camera is at. Based on that, first we developed a *fly* tool which could be controlled by the user by pressing the arrow keys of the keyboard to move the camera while guiding the direction of the motion

with the mouse movements. Navigation speed was adjusted automatically according to this equation:

$$V_{\text{nav}} = k\, minDist \tag{1}$$

where V_{nav} is the adjusted navigation speed and k is a parameter of unit 1/s that causes an increase or reduction in the acceleration applied to the camera. We noticed that two situations caused discomfort and disorientation for some users. The first situation is when k is too high. In this situation, when users move away from the geometry, the camera accelerates too quickly, producing an effect similar to teleporting and making users lose their location. The second situation happens for low k values, which can considerably increase the time required to reach the intended destination, making the navigation tedious. Thus, the value of k should be selected in a way that it balances these extremes.

In face of the difficulty in determining a value for k that suited all users, we decided to let the users define it manually, using the scroll button of the mouse. However, this led to other problems. When navigating very close to an object, some users increased the k value to move faster but forgot to readjust it when the camera was distant from the geometry, falling into a situation where k is too high.

These examples revealed the disadvantage of using only *minDist* as an estimation to adjust the speed; when *minDist* is too low, it works as a break even when the user wishes to move faster. For instance, when navigating through a corridor, the user may experience slow navigation due to the closeness of the walls. Thus, it is convenient to use some other type of estimation.

With that goal in mind, we attempted to use the distance to the central point of the screen. This seemed reasonable since this point represents momentarily the location the user wants to reach. Therefore, navigation speed started to be adjusted using *centerDist*, the distance from the camera to the center of the screen, rather than *minDist*.

However, the use of *centerDist* resulted in a strange behavior. The motion of the camera started to present peaks of speed, giving the impression that it stopped or accelerated instantly. The reason for this can be understood by observing the graph in Fig. 1. This graph shows the behavior of the curves of the *minDist* and *centerDist* estimations for one path followed by the camera in a time interval of around 6 seconds.

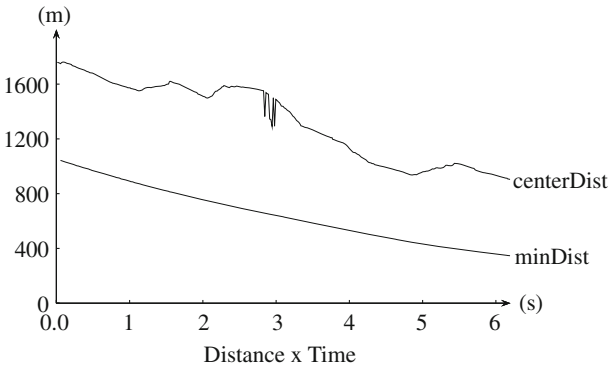


Fig. 1 Graph representing the behavior of the *minDist* and *centerDist* curves

It can be seen that the *minDist* curve is smooth while the *centerDist* curve is noisier, displaying some peak values that are completely inconsistent with the general behavior of the curve. This is related to the freedom the *fly* tool gives to the user, who can turn the camera to any direction at any time. At a given moment, the central point of the screen may fall on an object that is distant from the camera. The user then can turn the camera almost instantly toward a close-by object, leading the value of *centerDist* to drop abruptly. This is reflected in the speed adjustment, creating a steep deceleration. The opposite is also a problem. If, for example, the camera is inside an object and the user unwillingly points the camera toward an external point, the camera will be subject to a fast acceleration and will be thrown outside the object. These effects do not happen when *minDist* is used as estimation because it is independent from the orientation of the camera.

To avoid the situations caused by the peak values of the *centerDist* curve, we applied an *exponential moving average (EMA)*:

$$EMA_i = EMA_{i-1} + A\left(centerDist_i - EMA_{i-1}\right) \tag{2}$$

In (2), EMA_i is the smoothed value of $centerDist_i$ at instant i, EMA_{i-1} is the value smoothed at instant $i-1$, and A is a constant that influences how smooth the new curve will be and how quickly this curve will converge to the *centerDist* values. The smaller A is, the smoother the curve will be and the higher the time needed for such convergence to take place. The maximum value of $A = 1$ is equivalent to the original curve. Figure 2 shows the results of smoothing an interval of the *centerDist* curve (represented by the darker line) for three different values of A.

By smoothing the *centerDist* curve, we can use it to adjust the speed.

But one last problem remained: when the camera is near to an object but points to a distant place, the navigation speed tends to quickly increase, since *centerDist* is far. This can lead to undesirable situations, such as when the camera is inside an object and the user wants to view its exterior. In this case, the user needs to go outside of the object and then point the camera toward the object again. When leaving the object, the camera might point to a distant location for a period of time long enough to increase the speed too much. As a result, the camera could move away from the object before the user is able to adjust the movement. This is because the scale

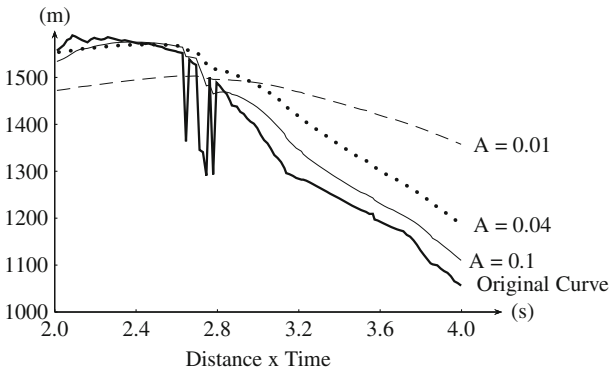


Fig. 2 Effect of applying the EMA to a curve

perception provided by *minDist* is no longer present. In such cases, *minDist* acts as a break, preventing the camera from moving far away too quickly.

To address this final issue, values of *centerDist* higher than $n \times minDist$ were discarded. This way, *minDist* serves as a criterion to decide when a *centerDist* value should or not be considered inconsistent. We found that $n = 10$ worked well on ours tests.

This solution reflects a hybrid use of the *minDist* and *centerDist* estimations: while *centerDist* ensures that speed adjustment is closer to what the user intends, *minDist* acts as a break in cases where the speed would be too high. The result is increased comfort for the user when using this tool, as demonstrated by the usability tests.

4.2 Collision detection and treatment

Preventing the camera from cutting through objects in a virtual environment can be crucial in some situations [5]. In immersive environments, for example, colliding with an object can halt the immersion and leave the user disoriented. Another problem occurs when a visualization involves a stereoscopy effect; in this case, a collision with an object in the scene can cause physical discomfort to the user's eyes.

McCrae et al. [14] used information from the distance cube to obtain a collision factor that causes the camera to smoothly dodge the closest objects. The idea is that each point in the cubemap located at a distance smaller than a given radius r produces a repulsion factor given by:

$$F(x, y, i) = w(dist(x, y, i))\, norm\big(pos(x, y, i) - camPos\big) \tag{3}$$

$$w(d) = e^{\frac{(r-d)^2}{\sigma^2}} \tag{4}$$

where $F(x, y, i)$ is the repulsion factor produced by point p referring to position (x, y) of image i of the cubemap. Value $dist(x, y, i)$ is the distance from p to the camera. The term $pos(x, y, i)$ is the position of p in world space, and *camPos* is the camera position. Function $norm(v)$ indicates the normalized vector of v. In (4), σ is a parameter that indicates the smoothness of the collision factor. The higher σ is, the smoother the calculated factor. Considering a spherical region with radius r and centered on the camera position, (4) results in determining a collision penalty that grows exponentially from the moment when point p enters this region.

The repulsion factors referring to (3) are computed for each position in the cubemap where $d < r$ and then are combined into a single factor:

$$F_{\text{collision}} = \frac{1}{6\, cubeRes^2} \sum_{x,y,i} F(x, y, i) \tag{5}$$

where *cubeRes* is the resolution of the distance cube.

When we applied the factor given by (5) to the camera, the *fly* tool behaved as described in the previous section: as the camera moves, $F_{\text{collision}}$ ensured that it smoothly avoided the objects in its path. The behavior obtained is similar to the assisted navigation described previously [4, 21], whereby the user can navigate through the environment without worrying about choosing a collision-free path as the system is in charge of this task. It is a different approach from that used by

McCrae et al. [14], who employed $F_{\text{collision}}$ combined with the POI technique, a more restrictive solution which does not give the user total control over the camera when navigation is being performed.

4.3 Examine with automatic pivot point

The *examine* tool allows any object or location in the scene to be inspected. Basically, its function corresponds to rotating the camera around a point, called the *pivot point*. In our test application, drag movements using the left mouse button made the camera rotate around the pivot point, which can be chosen by the user. Dragging vertically or horizontally with the right mouse button caused the camera to zoom in or zoom out, respectively. When the scroll mouse button is present, it can also be used for zoom operations. In our implementation collision detection and treatment is not taken into account when using the *examine* tool, since it would prevent certain forms of inspection to be performed. For example, zoom out operations can be used to change the level of scale in which the camera is. If collision treatment is present, the camera can get stuck due to a surface located behind it.

The location of the pivot point is crucial for the proper functioning of the *examine* tool. If it is not specified correctly, the camera can display behaviors which, from the user's point of view, would seem confusing. Figure 3 illustrates two of these cases.

In case (i), the pivot point (referred to as *pivot* in the image) is located outside the user's field of vision, too distant from the object to be inspected. When the *examine* tool is used, a mathematically correct rotation (from camera position 1 to 2) is made around the pivot. For the user, however, this operation leads to a completely unexpected motion. This problem is worse when the pivot is located at a great distance from the model: the greater the distance, the higher the angular speed of the camera and, as a consequence, the bigger the error as perceived by the user. Informal observations showed that this situation occurred rather often in our test application. In the case of (ii), the pivot is within the viewing angle of the camera but is located outside the object of interest. As seen in the image, *pivot* is beyond the model, and the rotation around it has the effect of a pan operation.

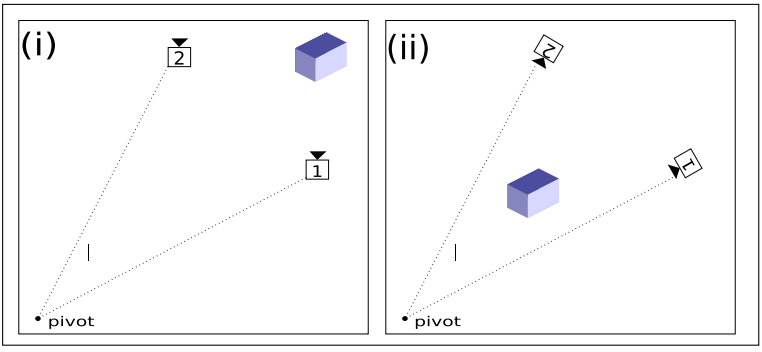


Fig. 3 Problems related to the pivot point: (*i*) pivot point located outside the field of vision; (*ii*) pivot point mapped beyond the object

Analyzing these situations, it can be concluded that they are basically caused by one reason: the pivot point is not located at a point that adequately corresponds to the object to be examined.

Usually, 3D visualization applications that make use of the *examine* tool include a button (or any other interface item) that, when selected, allows a new pivot point to be chosen. We observed that some people, even more experienced users, at some point forgot to select an adequate pivot point before starting to rotate around the object of interest.

This occurred especially frequently when some users switched from the *fly* to the *examine* tool, as they attempted to rotate the camera around the object located in front of them before readjusting the pivot point. And, even when the users did not forget to perform this last operation, they reported feeling upset with the fact that they had to do it explicitly.

The solution we found for this was to automatically determine a pivot point at the moment the *examine* tool is activated. As mentioned before, the solution proposed by Khan et al. of doing this by setting the pivot point to $minDist$ and reorienting the camera could conflict with the user intent. We propose to use the point corresponding to the center of the screen as the new $pivot$. Doing this, it is possible to establish a behavior that seems natural from the user's point of view. There is no reorientation of the camera and all the user needs to do is point the camera to the object of interest and then select the *examine* tool. It is reasonable to expect this feature, as in most cases the users only decide to inspect an object once it is in front of them. However, the object might not necessarily be located exactly in the direction of the center of the screen; in fact, the pivot point could be mapped to an object behind the one the user wishes to examine. In this case, the user would experience the effect of a pan operation, as shown in (ii) in Fig. 3. Another possibility is that the central point of the screen does not correspond to any valid point in the geometry, and thus determining $pivot$ is impossible.

To avoid these problems, the smallest distance present on the front face of the distance cube, $minFront$, is used. When the central point is not valid, $pivot$ is adjusted to the point that is $minFront$ away from the camera. This way, the angular speed of the rotation of the *examine* tool will be coherent with the scale in which the camera is located, preventing it from making excessively quick movements.

The $minFront$ estimation is also used when the pivot point is automatic mapped to a distant point located behind the object. This may occurs when the object to be inspected is not located exactly in the center, but this contains valid geometry that is far away from the object of interesting. If the pivot point is mapped to this point, the situation presented on (ii) in Fig. 3 should happen. To avoid this, we do not allow $pivot$ to be adjusted to a point whose distance is greater than $k \times minFront$. We found that $k = 5$ worked well on our tests. This solution does not solve the problem definitively, but reduces its occurence adequately.

5 User tests

The techniques presented in the previous section attempt to assist the users in the task of exploring virtual environments. From the user's point of view, this should result in a more comfortable navigation experience and fewer errors.

To verify this, usability tests were carried out with groups of users in order to gather information about the solutions developed. These tests were planned and conducted based on the orientations provided by Tullis and Albert [18], and were divided in two types, according with the procedures adopted and the information gathered:

- *Qualitative tests*: to collect feedback from users, such as how they felt while using the software, what bothered them, and what could be improved. The time was not taken into account and users had more freedom to explore the environment and make certain decisions.
- *Quantitative tests*: to collect quantitative information on how users interact with the scene, like how often a collision occurs or how many times the user got disoriented. The time was tracked and users had to follow a pre-determined set of steps, without the freedom of changing them.

5.1 Test environment

The tests were performed using the SiVIEP viewer, a project under development by Tecgraf/PUC-Rio in cooperation with Petrobras (Brazilian Oil & Gas Company). SiVIEP supports a comprehensive visualization of several types of models comprising an oil exploration and production enterprise. For example, it is possible to load environments ranging from oil platforms to wells and reservoirs in a single scene (Fig. 4). The main characteristic of the virtual environments resulting from this integration is that they are multiscale.

Two different versions of SiVIEP were used in the tests:

- *Automated*: this version supports the solutions discussed in the preceding sections. The user does not have to worry about speed adjustment, collisions are prevented automatically, and the explicit use of the pivot-point tool is not necessary.
- *Manual*: this version does not include any of the improved techniques previously mentioned. The speed in the *fly* tool must be adjusted manually with the mouse scroll button, the user must be careful to not collide with the models, and the pivot-point tool has to be used always before beginning to inspect an object with the *examine* tool.

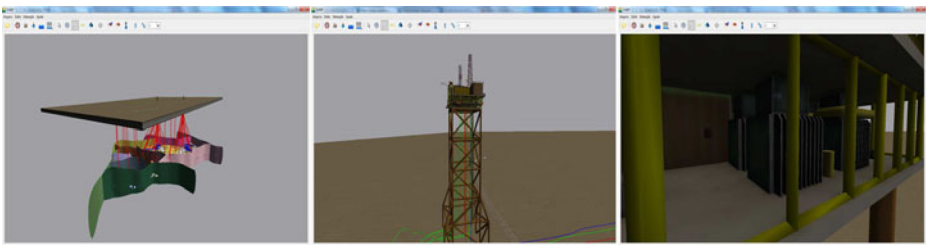


Fig. 4 SiVIEP: visualization of an oil enterprise. In the first image, a complete oil field can be seen. Then the camera reaches the scale of a platform. Finally, the interior of the platform can be navigated

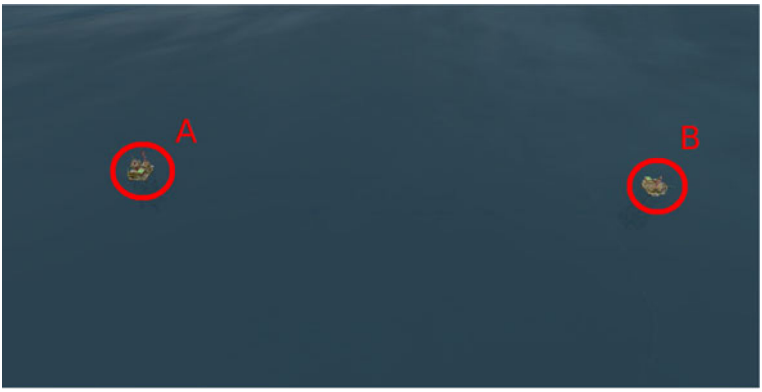


Fig. 5 Scenario used in the tests

The scenario used consisted of two oil extraction platforms, A and B, located at a certain distance apart from each other. The camera was initially placed in a position where both platforms could be seen (Fig. 5).

5.2 User profiles

Nineteen people were selected to carry out the tests. They were divided in two groups: *advanced users*, with experience in the use of 3D visualization and 3D modeling applications who use this type of software at least once in a month; and *non-advanced users*, with little experience in 3D visualization applications, except for some electronic games, and who do not use 3D visualization applications frequently.

From the 19 individuals, 7 were allocated in the first group (advanced users) and 12 in the second group (non-advanced users). All of them had the following characteristics in common: they were between 20 and 35 years old and did not have any previous contact with the application used in the tests. The individuals in the advanced group of users are herein called PA1 to PA7, while the test users in the non-advanced group are identified as PN1 to PN12. Also, advanced users and non-advanced users from PN1 to PN5 performed only the qualitative tests, while users PN6 to PN12 performed the quantitative tests. All users were male, except one of the non-advanced group that performed the quantitative tests.

5.3 Qualitative tests

5.3.1 Procedures adopted

Each person was first asked to read and sign a consent agreement to confirm their commitment to taking the test. They were then given an overview of SiVIEP to introduce them to the application and its functions. The test consisted of asking the subjects to use the two versions of SiVIEP.

Before each person started to use one of the versions, some instructions were given about the functioning of the navigation tools in that version. In the manual

version, for instance, the users were asked to avoid crossing through the models in the scene and instructed on how to make manual speed adjustments and to operate the pivot-point tool. In the case of the automated version, the users were informed that they did not have to worry about these aspects.

The users were asked to navigate to platform A using the *fly* tool. Once there, they had to explore the internal area of the platform to select three objects inspection with the *examine* tool. Finally, the users were asked to navigate from platform A to platform B.

After using each version, the users were asked to answer a questionnaire aimed at gathering their impressions about the tools used. This questionary consisted of the following statements, here identified as S1, S2, S3, S4 and S5:

- S1: *I did not have any difficulty with the speed adjustment of the fly tool.*
- S2: *I was able to perform the tasks without colliding with the environment.*
- S3: *I did not have any difficulty with the pivot-point tool.*
- S4: *I did not feel disoriented at any moment when navigating in the virtual environment.*
- S5: *I felt comfortable using the navigation tools.*

Below each of these statements there was a scale of ten numbers, from 1 to 10, 1 meaning that the user disagrees completely with the statement and 10 meaning that the user fully agrees with it. At the end of the form there was a blank space where the users could describe their general impressions and justify the grades given.

After both versions had been tested, the users were asked to fill out a final survey consisting of two written questions, identified as Q1 and Q2:

- Q1: *Which of the two approaches did you prefer: the automated navigation techniques or the manual techniques? Why?*
- Q2: *With regard to the approach you preferred, in your opinion could something be improved? If affirmative, what is it and why would it need to be improved?*

Lastly, the order in which the versions were presented to each user was different from test to test. The first person to take the test used the manual version first, and then the automated version second. The second took the test in the opposite order. This pattern was followed until the last user. This measure was taken with the purpose of minimizing the learning effect of using the first version over the second.

5.3.2 Non-advanced group results

Tables 1 and 2 show the results obtained after testing the group of non-advanced users. They include the responses to the 5 statements presented in the previous

Table 1 Results of the usability test for the manual version (group of non-advanced users)

	PN1	PN2	PN3	PN4	PN5	Avg
S1	7	8	10	9	5	7.8
S2	6	5	7	7	2	5.4
S3	6	9	8	7	3	6.6
S4	4	8	7	9	1	5.8
S5	5	8	9	10	4	7.2

Table 2 Results of the usability test for the automated version (group of non-advanced users)

	PN1	PN2	PN3	PN4	PN5	Avg
S1	9	10	10	10	9	9.6
S2	10	10	10	10	10	10.0
S3	10	8	10	10	10	9.6
S4	10	10	9	10	10	9.8
S5	10	9	10	10	10	9.8

section. Table 1 provides the grades referring to the use of the manual version, while Table 2 contains the grades given to the automated version. The final column in each table shows the average grade for each statement. For a better view of the general results, these averages are presented side by side in Fig. 6. The confidence interval used in the generation of this graph was 90%.

We applied a two-tailed paired t-test to compare the average of the results for both versions. We obtained $t(4) = -6.43$, $p = 0.001$, which affirms that this group preferred the automated version of the navigation. When using the manual version, users complained especially about the difficulty of controlling camera speed in order not to collide with the environment. One of the users, for instance, reported that "the program is somewhat abrupt, which makes its use difficult for someone who is not used to or does not have enough dexterity for 3D games and software". The lack of collision treatment created situations in which some people felt lost. When these situations occurred, 2 of the 5 users mentioned that they would like to quit the task.

The written answers confirm the statistic results. For question $Q1$, all users in this group replied that they preferred the automated version rather than the manual one. In the justifications, most users mentioned that the techniques provided by the automated version made navigation simpler, less prone to errors, and easier to control. As for question $Q2$, none of these users considered that any improvement was necessary to their preferred version.

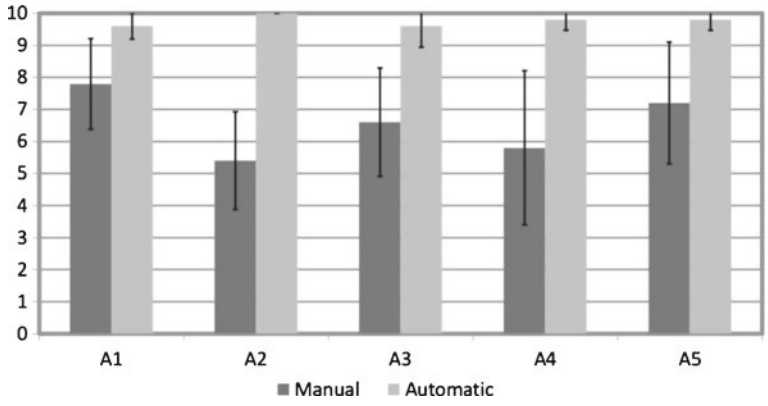


Fig. 6 Comparative results between the manual and the automated version (group of non-advanced users)

Table 3 Results of the usability test for the manual version (group of advanced users)

	PA1	PA2	PA3	PA4	PA5	PA6	PA7	Avg
S1	10	9	7	7	10	9	9	8.7
S2	3	7	7	7	10	8	7	7.0
S3	7	8	10	3	5	5	7	6.4
S4	3	9	6	5	10	6	8	6.7
S5	6	8	8	8	10	8	7	7.8

5.3.3 Advanced group results

Tables 3 and 4 show the results obtained after the application of the tests to the group of advanced users. The results are presented in the same format as those in the previous section. Figure 7 also shows a comparison between the grades given to the manual version and those given to the automated version. The confidence interval was 90%.

We applied a two-tailed paired t-test to compare the average of the results for both versions. We obtained $t(4) = -2.1$, $p = 0.09$, which does not allow us to affirm that this group preferred the automated version of the navigation. Nevertheless, as can be observed, the automated version received higher grades in all statements with the exception of S1, which sought to evaluate the automatic speed adjustment of the *fly* tool. Analyzing the justifications for the grades and the general comments made by the users revealed some interesting points.

Almost all advanced users who gave a lower grade to the automatic speed adjustment reported that, when getting very close to an object, the camera would become too slow and it would take a while until they were able to move away from the object again. These users felt impatient, and this feeling was made worse by the fact that no control option was provided to allow them to *momentarily* increase the speed. Nonetheless, the same users noted that the automatic adjustment was good because it allowed them to be less concerned with the controls and helped them avoid some errors, which is in agreement with the general comments made by the non-advanced users. In summary, the advanced users wished they were offered some sort of control which allowed them to make a more "customized" adjustment at certain moments, while at the same time including automatic speed adjustment.

Another factor that contributed to the preference for the manual version in S1 was the manner in which the tests were conducted. Before the users began the manual version test, they were asked to avoid cutting through (colliding with) objects in the scene. This led them to be very cautious in relation to speed adjustment, thus avoiding possible difficulties they could face when using the manual version. This caution may have contributed to the higher grades that resulted. However, this had a negative impact on the comfort experienced by the users: in order to avoid colliding with the models, they were forced to stop often and readjust the navigation speed.

Table 4 Results of the usability test for the automated version (group of advanced users)

	PA1	PA2	PA3	PA4	PA5	PA6	PA7	Avg
S1	7	5	10	5	9	9	9	7.7
S2	10	9	9	10	10	9	10	9.6
S3	10	6	6	10	10	9	10	8.7
S4	6	9	7	10	10	8	9	8.4
S5	10	6	9	9	10	9	9	8.9

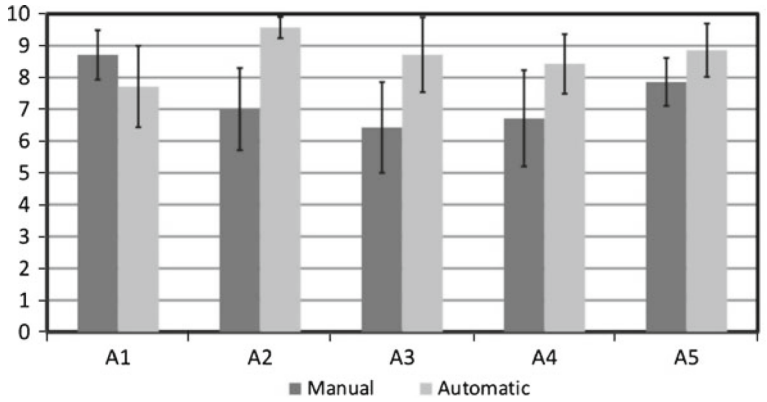


Fig. 7 Comparative results between the manual and the automated version (group of advanced users)

The greatest contributions of the automated version were related to statements S2 and S3, which aimed at evaluating the efficacy of collision treatment and automatic pivot point, respectively. The users could easily notice the effects provided by these techniques and were very satisfied with their results. Some users expressly stated that, thanks to these techniques, they did not make certain errors and were able to focus less on interface issues. Finally, the grades given by the users in the advanced group to statements S4 and S5 demonstrated that they felt more comfortable and experienced fewer moments of disorientation while using the automated version. This was also true for the non-advanced users.

Regarding the final survey question, 6 out of the 7 advanced users stated in Q1 that they preferred the automated version rather than the manual one. Only user PA2 preferred the latter. In the justification, this user mentioned the problem with the automatic speed adjustment and the lack of feedback to the user about the location of the pivot point when using automatic adjustment. As a result of PA2 suggestions, we now draw the pivot point in the screen on recent versions of ours systens. Like Fitzmaurice et al. [6], we found that this also caused a improvement on the usage of the tools related to the pivot point.

Finally, in question Q2, which asks for suggestions about what could be improved in the preferred version, all of them asked for some kind of control that allows them to momentarily increase the speed. This leads us to assume that advanced users have a greater tendency to prefer solutions that make the tools simple to use but that are not completely automated. The opposite might be said of non-advanced users: due to their lack of experience, they prefer approaches that minimize the need to adjust the parameters of the navigation tools.

5.4 Quantitative tests

5.4.1 Procedures adopted

As in the qualitative tests described above, each user was asked to sign an agreement to confirm their commitment to taking the test. Then, the features and controls of SiVIEP were presented.

Each user had to follow a pre-determined set of steps, consisting of navigating to the platform A using the *fly* tool, finding three different objects to inspect using the *examine* tool and travelling to platform B using again the *fly* tool. A key diference between this quantitative test and the previously described qualitative test is that in the quantitative test users were not given the freedom to choose which objects they should examine, rather, the objects were assigned. To minimize learning effects, the objects to be examined in the manual version were different from those of the automatic version. Thus, the difficulty in finding objects using the software for the second time should be the same. For the same reason, the procedure for changing the order in which the versions were presented to a user was also used. The users were also asked to avoid collisions when using the *fly* tool.

There was no written questionnaire issued after the test of each version. The tests were filmed, and the videos obtained were analyzed in order to determine the number of times that each of the following events occurred:

- A - Collision.
- B - Incorrect adjustment of the pivot point.
- C - Disorientation.
- D - Usage of the reset button.

An occurrence of these events indicates an error or a bad user experience.

The reset button sends the camera back to its original position. It was given to users as an option to orient themselves again when a event of disorientation ocurred. When a user clicks this button, brought the user directly to the place where the camera was before the disorientation event occurred.

The duration of the tests were also recorded. This was done as the users no longer had the freedom to choose which objects should be examined. Also, users were encouraged to relay their impressions verbally while performing the tests. This way, we could obtain extra information that could help explain the occurrence of events listed above.

5.4.2 Results

Tables 5 and 6 show the results of the quantitative tests for the occurrence of events A, B, C and D. Each row represents a different user, while the columns labeled A, B, C and D represent the number of times that the respective event happened. The last column contains the total number of events for a given user. Table 5 refers to results of the manual version. Table 6 refers to the automatic version. Figure 8 shows the total number of errors for manual and automatic versions for each user and facilitates the comparison between then.

Table 5 Number of events A, B, C and D for the manual version

	A	B	C	D	Total
PN6	5	1	3	0	9
PN7	1	0	1	0	2
PN8	2	0	0	0	2
PN9	3	4	4	3	14
PN10	3	5	6	0	14
PN11	7	2	4	1	15
PN12	5	1	1	0	7

Table 6 Number of events A, B, C and D for the automatic version

	A	B	C	D	Total
PN6	0	3	1	0	4
PN7	0	0	1	1	2
PN8	0	0	0	0	0
PN9	0	0	1	0	1
PN10	0	0	1	0	1
PN11	0	0	1	0	1
PN12	0	0	0	0	0

The number of events in the manual version was greater than in the automatic version for most users. This indicates that the automatic version was less prone to errors and caused fewer uncomfortable situations. The exceptions are users PN8 and PN7, which will be discussed later.

As expected, there were no collisions (event A) for the automatic version since the system prevents them from occurring. In the manual version, it was observed that the collisions were related to the difficulty of controlling the camera speed. These collisions often led to disorientation.

The incorrect adjustment of the pivot point was another factor responsible for disorientation. Again, the highest incidence of this error occurred in the manual version. The exception was user PN6: he had difficulty in orienting the camera so that the object to be examined could be located in the center of the screen. Because of this, the automatic adjustment of the pivot point did not work the way expected. The result is that this user reported that he preferred to use the tool to manually set the pivot point.

Table 7 contains the duration in seconds of the tests for each user. The last column indicates the relationship between the times spent on manual and automatic versions. For most users, the time required to complete the test was higher in the manual version. The main reason of this was that users spent a significant amount of time trying to correctly adjust the navigation speed. Moreover, the disorientation situations also contributed to the increase in time.

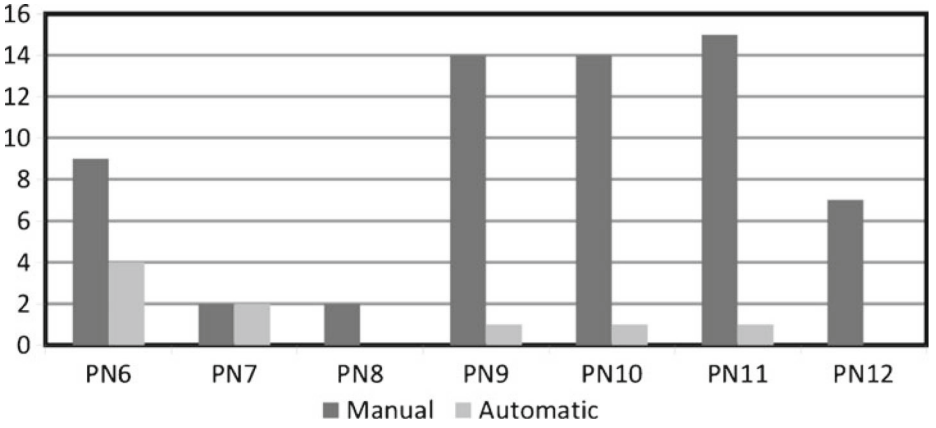


Fig. 8 Comparative results between the manual and the automated version

Table 7 Time spent to perform the tests

	Manual	Automatic	Manual/Automatic
PN6	540	240	2.25
PN7	215	380	0.56
PN8	180	213	0.84
PN9	535	230	2.32
PN10	895	400	2.23
PN11	720	480	1.5
PN12	320	250	1.28

The exceptions again were users PN7 and PN8 (Fig. 9). During the tests, it was found that these users had previous experiences that affected their navigation performances. User PN8 reported that he plays 3D games very often. Hence, he had no difficulty in manually controlling the camera on both versions and collided lightly with the objects only twice. The time spent in the automatic version was slightly greater than the time spent on the manual version, since the ability of user PN8 allowed him to move faster through the scene using the manual adjustment of speed. Similarly to some members of the advanced group of users, user PN8 complained of not being able to adjust the speed in the automatic version.

With user PN7, it was discovered during the test that he had already used other scientific visualization software in the past. The automatic version was tested first, and he spent a significant amount of time trying to use the navigation tools in SiVIEP the same way he used tools on the other software. Only after some time did the user become better acquainted with the tools of SiVIEP. This contributed to the result of the user requiring almost twice as much time in the automatic version versus the manual version. This event suggests another way to improve usability on our system: the investigation of the user's experience with other software. With such information, we could identify possible improvements to be incorporated in our navigation tools.

Finally, at the end of the tests users were asked about which version they had preferred. Only users PN7 and PN8 chose the manual version. Again, the factor that motivated this choice was the lack of speed control in the automatic version. This is consistent with the results reported in qualitative tests with the group of advanced users.

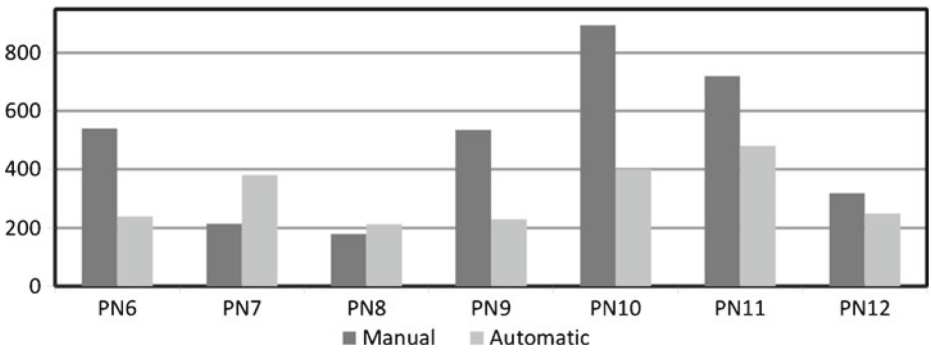


Fig. 9 Comparative results between the times spent on manual and automatic versions

6 Conclusions

This work presented techniques to assist and facilitate navigation in 3D virtual environments. They were based on the construction and maintenance of a data structure called *cubemap* [14]. Improvements were proposed for the *examine* tool, with the development of a system to automatically determine the pivot point, and for the *fly* tool, in which collision support and automatic speed adjustment in relation to the scale were implemented.

To verify the efficacy of the solutions proposed, usability tests were performed. The results allow us to conclude that the techniques presented here improve the navigation experience of the users. From the 19 subjects of the usability tests, 16 preferred the version of the application that included the techniques proposed. In particular, the automatic adjustment of the pivot point in the *examine* tool and the collision support implemented in the *fly* tool had a significant positive impact in user experience.

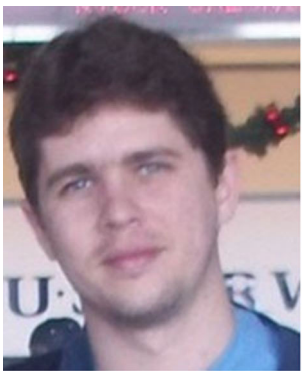
Navigation in 3D environments still presents issues and provides plenty of challenges. Multiscale environments, which are becoming more common, identify several further navigation-related problems to be solved in the future. We believe the techniques demonstrated in this work contribute to providing solutions to some of the problems identified regarding navigation in these environments.

Acknowledgements The authors thank Petrobras for this research support and for the software used in this research (SiVIEP). D. Trindade thanks CAPES and A. Raposo, FAPERJ and CNPq for the individual support granted to this research.

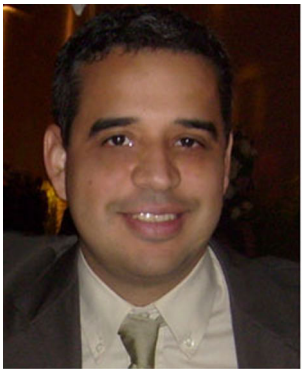
References

1. Baciu G, Wong WSK (1997) Rendering in object interference detection on conventional graphics workstations. In: Proceedings of the 5th Pacific conference on computer graphics and applications. IEEE Computer Society, Washington, DC, pp 51–58
2. Baciu G, Wong WSK, Sun H (1998) Recode: an image-based collision detection algorithm. In: Proceedings of the 6th Pacific conference on computer graphics and applications, pp 125–133. doi:10.1109/PCCGA.1998.732079
3. Bederson BB, Stead L, Hollan JD (1994) Pad++: advances in multiscale interfaces. In: CHI '94: conference companion on human factors in computing systems. ACM, New York, pp 315–316. doi:10.1145/259963.260379
4. Calomeni A, Celes W (2006) Assisted and automatic navigation in black oil reservoir models based on probabilistic roadmaps. In: I3D '06: proceedings of the 2006 symposium on interactive 3D graphics and games. ACM, New York, pp 175–182. doi:10.1145/1111411.1111443
5. de Sousa Rocha R, Rodrigues MAF (2008) An evaluation of a collision handling system using sphere-trees for plausible rigid body animation. In: SAC '08: proceedings of the 2008 ACM symposium on applied computing. ACM, New York, pp 1241–1245. doi:10.1145/1363686.1363972
6. Fitzmaurice G, Matejka J, Mordatch I, Khan A, Kurtenbach G (2008) Safe 3d navigation. In: I3D '08: proceedings of the 2008 symposium on interactive 3D graphics and games. ACM, New York, pp 7–15. doi:10.1145/1342250.1342252
7. Furnas GW, Bederson BB (1995) Space-scale diagrams: understanding multiscale interfaces. In: CHI '95: proceedings of the SIGCHI conference on human factors in computing systems. ACM Press/Addison-Wesley, New York, pp 234–241. doi:10.1145/223904.223934
8. Glueck M, Crane K, Anderson S, Rutnik A, Khan A (2009) Multiscale 3d reference visualization. In: Proceedings of the 2009 symposium on interactive 3D graphics and games. ACM, New York, pp 225–232. doi:10.1145/1507149.1507186

9. Jul S, Furnas GW (1998) Critical zones in desert fog: aids to multiscale navigation. In: UIST '98: proceedings of the 11th annual ACM symposium on user interface software and technology. ACM, New York, pp 97–106. doi:10.1145/288392.288578
10. Khan A, Komalo B, Stam J, Fitzmaurice G, Kurtenbach G (2005) Hovercam: interactive 3d navigation for proximal object inspection. In: Proceedings of the 2005 symposium on interactive 3D graphics and games. ACM, New York, pp 73–80. doi:10.1145/1053427.1053439
11. Khan A, Mordatch I, Fitzmaurice G, Matejka J, Kurtenbach G (2008) Viewcube: a 3d orientation indicator and controller. In: Proceedings of the 2008 symposium on interactive 3D graphics and games. ACM, New York, pp 17–25. doi:10.1145/1342250.1342253
12. Kopper R, Ni T, Bowman DA, Pinho M (2006) Design and evaluation of navigation techniques for multiscale virtual environments. In: VR '06: proceedings of the IEEE conference on virtual reality. IEEE Computer Society, Washington, DC, pp 175–182. doi:10.1109/VR.2006.47
13. Mackinlay JD, Card SK, Robertson GG (1990) Rapid controlled movement through a virtual 3d workspace. SIGGRAPH Comput Graph 24(4):171–176. doi:10.1145/97880.97898
14. McCrae J, Mordatch I, Glueck M, Khan A (2009) Multiscale 3D navigation. In: I3D '09: proceedings of the 2009 symposium on interactive 3D graphics and games. ACM, New York, pp 7–14. doi:10.1145/1507149.1507151
15. Perlin K, Fox D (1993) Pad: an alternative approach to the computer interface. In: SIGGRAPH '93: proceedings of the 20th annual conference on computer graphics and interactive techniques. ACM, New York, pp 57–64. doi:10.1145/166117.166125
16. Sousa Santos B, Dias P, Pimentel A, Baggerman JW, Ferreira C, Silva S, Madeira J (2009) Head-mounted display versus desktop for 3d navigation in virtual reality: a user study. Multimedia Tools Appl 41(1):161–181. doi:10.1007/s11042-008-0223-2
17. Tan DS, Robertson GG, Czerwinski M (2001) Exploring 3d navigation: combining speed-coupled flying with orbiting. In: CHI '01: proceedings of the SIGCHI conference on human factors in computing systems. ACM, New York, pp 418–425. doi:10.1145/365024.365307
18. Tullis T, Albert W (2008) Measuring the user experience: collecting, analyzing, and presenting usability metrics. Elsevier, Amsterdam
19. Ware C, Osborne S (1990) Exploration and virtual camera control in virtual three dimensional environments. SIGGRAPH Comput Graph 24(2):175–183. doi:10.1145/91394.91442
20. Ware C, Fleet D (1997) Context sensitive flying interface. In: SI3D '97: proceedings of the 1997 symposium on interactive 3D graphics. ACM, New York, pp 127–130. doi:10.1145/253284.253319
21. Xiao D, Hubbold R (1998) Navigation guided by artificial force fields. In: CHI '98: proceedings of the SIGCHI conference on human factors in computing systems. ACM Press/Addison-Wesley, New York, pp 179–186. doi:10.1145/274644.274671
22. Zhang XL (2009) Multiscale traveling: crossing the boundary between space and scale. Virtual Real 13(2):101–115. doi:10.1007/s10055-009-0114-5



Daniel Ribeiro Trindade is a researcher at Computer Graphics Technology Group - Tecgraf/PUC-Rio. He received his MSc in computer science at the Pontifical Catholic University of Rio de Janeiro - PUC-Rio. He graduated in computer engineering at the Federal University of Espírito Santo, Brazil. His current interests include Virtual Reality, 3D interaction and computer graphics.



Alberto Barbosa Raposo is an assistant professor at the Dept. of Informatics/PUC-Rio, project coordinator at Tecgraf/PUC-Rio and FAPERJ researcher. He received his BSc in electrical/computer engineering at the State University of Campinas, Brazil. His current interests include Virtual Reality, 3D interaction, groupware, HCI, and computer graphics, with more than 120 publications in these areas. His projects were supported by: Petrobras, CNPq, FINEP, FAPERJ and RNP. He is a distinguished young scholar of PUC-Rio, and also a NVIDIA academic partner.