Improved Automatic Speed Control for 3D Navigation

Domi Papoi[1]and Wolfgang Stuerzlinger[2]

[1] York University, Toronto ON M3J 1P3, Canada
[2] SIAT, Simon Fraser University, Vancouver BC V3T 0A3, Canada
w.s@sfu.ca

Abstract. As technology progresses, it is possible to increase the size and complexity of 3D virtual environments. Thus, we need deal with multiscale virtual environments today. Ideally, the user should be able to navigate such environments efficiently and robustly, which requires control of the user speed during navigation. Manual speed control across multiple scales of magnitude suffers from issues such as overshooting behaviors and introduces additional complexity. Most previously presented methods to automatically control the speed of navigation do not generalize well to environments with varying scales. We present an improved method to automatically control the speed of the user in 3D virtual environment navigation. The main benefit of our approach is that it automatically adapts the navigation speed in a manner that enables efficient navigation with maximum freedom, while still avoiding collisions. The results of a usability test show a significant reduction in completion time for a multi-scale navigation task.

Keywords: 3D Navigation; Virtual Environments.

1 Introduction

Virtual navigation, i.e., movement within a *virtual environment* (VE), is a common interactive task in *three-dimensional* (3D) VE. During such navigation, users need to maintain their orientation and interact to move their viewpoint. Thus, 3D navigation involves two main tasks: wayfinding and travel, but we only focus on the later here.

Travel is the motor component of navigation. It can be defined as the actions that the user makes through the user interface to control the position and orientation of their viewpoint. In VEs, travel techniques enable the user to control their viewpoint and direction, and other attributes of movement, such as the speed. Here, we present a new method to control the speed of travel in *multi-scale virtual environments* (MSVEs).

Generally, the user can move in all 6 degrees of freedom (6DOF) in VEs. Yet, direct control of all 6DOFs is challenging. Compare the skills required to pilot a car or plane (which many can master) to those required to control a helicopter (which fewer possess). One can also observe this in most computer games, where navigation typically involves control over four or fewer DOF, typically rotate left/right and up/down, move forward/backward and both ways sideways, all at predefined speeds.

Many 3D *user interfaces* (UIs) ignore the aspect of changing the speed of the travel and simply use a reasonable constant velocity. This works reasonably well as long as the size and detail of the environment do not vary much. In MSVEs, a fixed speed leads to problems because a constant speed will always be too slow in some situations and too fast in others. If the speed is too slow, user frustration can set in quickly. If the speed is too fast, the user can overshoot the target, forcing the user to turn around or back up, to navigate back to the intended destination. On the other hand, allowing the user to explicitly control the speed across multiple scales adds complexity to the interface, and the user then even more easily overshoots or undershoots the target [21] and might be forced to again take corrective actions [20]. Another issue with manual control is that users can fly into objects when they do not stop in time or when backing up. This can lead to usability issues, as especially novice users do not know how to recover quickly from being inside an object [7]. A potential solution for this problem is to slow the user down when they get close to an object. Another aspect of VE immersion that needs to be considered is cybersickness. Recent work aims to reduce such symptoms [11].

There are many options for a user to control their speed, including buttons, sliders, or various attributes of the users' pose. A discrete technique for speed control might use buttons, which increases/decreases the speed by a predefined amount (say by 50%) and allows backward travel, while a slider-based control might use a linear mapping. All these controls give the user direct control over the speed of travel. The main drawback is that this choice adds complexity to the user interface, as the user has to constantly monitor their speed and adapt it to the current environment.

Ideally, a navigation control scheme should be as simple as possible, to make the user interface easy to learn. Another constraint is that each navigation functionality requires some physical control or a widget, which consumes either display space, requires dedicated buttons, or introduces modes. That means that fewer controls are typically better. This design trade-off is directly visible in touchscreen user interfaces, where space for widgets is at a premium, and multi-touch interaction possibilities are limited to, e.g., one-/two-/three-finger-based controls. There, a system-controlled speed technique might be more appropriate. Another ideal use case for automatic speed control are MSVEs where the user has to repeatedly travel between regions with radially varying spatial complexity, such as tight corners that necessitate short, slow, and precise movements, while open spaces can benefit from higher speeds. In such cases, the user can benefit much from automatic changes to the speed depending on the surrounding geometry. This idea is the main motivation for our research.

As example consider a VE for a star system, where the user is on a planet's surface and looks up into the vast empty space between planets. Launching, the user into space at very high speeds seems a good choice but can lead to a loss of control if the user steers in the wrong direction. Instead, we could take the (invisible) geometry behind the user into account and start slow, but keep increasing the speed, if the user keeps moving towards free space. In contrast, when the user approaches an object with high speeds, the system will slow the user down, which avoids overshooting. This is especially important if the user aims just beside the planet to get to the other side of the planet. Yet, as noted by Trindade [21], directly using the proximity of geometry can slow the user down too much in certain scenarios. For example, if the user navigates through a tunnel the system will reduce the speed drastically based on the close proximity to the walls, which could lead to speeds that are perceived as frustratingly slow.

1.1 Previous Work

In VEs, user actions must be mapped in some more or less intuitive way to travel. Mine [17] presented an overview of motion specification interaction techniques and, similar to Robinett [19], also discussed issues relevant to their implementation of travel techniques. Several studies of immersive travel techniques have been described in the literature, for instance comparing different travel modes and metaphors for specific VE applications, e.g., [5,15]. Bowman et al. [3] discussed various ways to control travel speed. Yet, allowing the user to explicitly control the speed across multiple scales adds complexity to the interface, and the user then even more easily overshoots or undershoots the target [21] and might be forced to take corrective actions [20].

A common approach to scaling the user during navigation is to allow the user to actively control the scale of the world. One of the earliest was the 3DM immersive modeler [4], which enables the user to "grow" and "shrink". SmartScene [14] also allowed the user to control the scale of the environment to allow both rapid navigation and manipulation of objects at all scales. The scaled-world grab technique [18] scales the user in an imperceptible way when an object is selected. While active scaling enables the user to specify the scale of the world, it requires additional interface components to do so. In contrast, a 3D UI could also change the scale of the world automatically based on the user's current task or position. This automated approach obviates the need for the user to specify a scale. An example of an automated scaling approach is "Multi-scale Virtual Environments" [12]. This approach allows the user to concentrate on navigating instead of scaling while still benefitting from having the world scale up or down. However, such VEs require careful design, as the hierarchy of objects and scales need to be intuitive and usable for the user.

The speed control of a travel technique is at least weakly linked with the scale of the environment and the user's preferences. The maximum allowed speed is dictated by the scale of the environment, while the minimum sensible speed corresponds to the finest detail. Users can manually adjust the speed through a travel interface by various input commands [10] or speed mappings [1]. If the scale and level of detail of the environment is known a priori, then the maximum and minimum speed can be set accordingly. Freitag et al. [9] identify locations that maximize scene visibility, based on predefined region importance scores and real-time tracking of the exploration status of scene regions.

Mackinlay [13] first observed that the current distance to a target point is an appropriate way to control viewer speed. Ware and Fleet [22] investigated this further and found that in most situations, the minimum distance to any visible point generally works best, but noted also that average distances were competitive.

An improved version of Ware and Fleet's interface [22] is the approach proposed by McCrae et al. [16], which uses a six-sided distance map, the *cubemap*, which encodes the distance to all visible parts of the surroundings of the user through six depth maps from the camera viewpoint. These depth maps are generated by rendering six images in the six main axial directions, each one corresponding to a side of the cube. Every time the camera viewpoint changes, the cubemap is updated in real time. Based on the cubemap, McCrae et al.'s method then computes a vector that displaces the camera in a way that adjusts both speed and direction, similar to the distance-dependent speed control presented by Ware and Fleet [22]. Through the weighting by distance, the direction of the vector adjusts the travel direction to avoid collisions.

Trindade et al. [21] improved McCrae's et al.'s approach to facilitate travel in a MSVEs. In their flying technique, they also include collision avoidance and automatic navigation speed adjustment with respect to the scale of the environment. They identified that when flying close to geometry, speed control via the global minimum can unnecessarily slow the user down. For example, when the user is flying through a tunnel that has no geometry straight ahead, the nearby walls reduce the speed (too) much, and therefore the user would fly very slowly. Their solution is to use the distance along a ray in the view direction to detect situations where the viewer could speed up. Using an exponentially weighted average between the distance along the view direction and the global minimum distance, they smooth out the resulting rough speed changes. Despite this weighted average approach, a speed computed for a distance of infinity or equivalent will overwhelm any other terms. This can cause the user to move at huge speeds very close to geometry, which is undesirable. Moreover, the discrete nature of using a sampling ray can cause abrupt speed changes, if said ray falls on/off geometry. Another variation of the cubemap was used by Duan et al. [6] to control a flying vehicle model.

Argelaguet [2] proposes a new method of speed control that aims to keep optical flow constant. Yet, they found that there is no strong difference between distance-based speed control and a method that keeps optical flow constant. In contrast, Freitag et al. [8] adjust the travel speed automatically based on the informative quality of the viewpoint. When the viewpoint has a high visual quality, the navigation speed is decreased and vice versa.

1.2 Contributions

Our contributions are:

- A new, efficient, and robust way to automatically adapt the user's speed depending on the camera's direction and the surrounding environment by smoothly attenuating the effect of geometry in the view direction different from the surround.
- A user study evaluating our new automatic speed control method relative to two other methods from previous work.

2 Automatic Speed Control for 3D Travel

In this chapter, we first explain the technical details of our automatic speed control technique. Our speed control method applies to all 3D travel interfaces where the user controls their motion through specifying a direction and then flying or traveling in (or reversing based on) said direction. Similar to McCrae et al. [16], we compute the distances to all objects around the viewer by generating a cube map of all objects. Instead of using a world space aligned cube map we use a view aligned cube map, as suggested by Trindade et al. [21]. After all, a world space aligned cube does not encode directly where geometry is relative to the viewer and their view direction. This makes it (a bit) harder to tell where an object is relative to the viewer.

We propose an improvement to the equation for computing the displacement vector proposed by McCrae et al. [16], by scaling it with a smoothing function. We first compute the average displacement vector from the cubemap over all its pixels:

$$\overrightarrow{disp} = \frac{1}{6N_xN_y}\sum_{x,y,i} w\big(dist(x,y,i)\big)\cdot norm(\overrightarrow{pos}(x,y,i) - \overrightarrow{eye})$$

In the above equation i is an integer value between 1 and 6 and represents one side of the cube map. The horizontal and vertical resolutions are represented by N_x and N_y. While the sum appears to involve only 2 dimensions, there are 3D vectors involved and the final result is also a vector. To give a larger weight to geometry closer to the viewer, McCrae et al. used an exponential soft penalty function. To reduce computational effort, we present here a simpler option for the weighting function that uses a smooth-step or an improved version of the smooth-step function with zero 1st and 2nd order derivatives at *t=0* and *t=1* to determine how nearby geometry influences the viewer:

$$smoothstep(t) = 6t^5 - 15t^4 + 10t^3$$

$$w_1(dist) = \begin{cases} 1, & if\left(\frac{\min\,(dist,\delta)}{\delta} < \alpha\right), else \\ 1 - smoothstep\left(2\,\frac{\min\,(dist,\delta)}{\delta} - 1\right) & \end{cases}$$

Where δ represents the bound radius within which objects should affect the user and α is a dynamic penalty control variable within [0, 1]. As δ is constant across samples, the viewer's collision boundary is then a sphere with radius δ. The bound radius δ can be modulated by a scale estimate, which is the minimum distance from the cubemap. In our work, we choose 0.5 for the dynamic penalty control variable α. McCrae et al.'s technique then uses the minimum distance across the cubemap to control the speed and applies the displacement vector to the viewer position to avoid collisions.

As mentioned above in the review of previous work, this computed speed may be too low in long narrow passages [21]. To address this issue in a better way than Trindade et al.'s ray-based solution [21], we propose to *add* a second weighting term *$w_2(dist)$* to the sum, which increases the weight of the contribution of geometry close to the view direction with a smooth fall off for geometry orthogonal or behind the viewer. Based on pilot experiments, we use the 16th power of the cosine of the angle relative to the view direction and redefine the weighting function accordingly.

$$w_2(dist) = \max(\cos^{16}(\theta), 0)$$
$$w(dist) = w_1(dist)w_2(dist)$$

Without the weighing term, all directions have equal influence. As shown in Fig. 1, applying the second weighting term *$w_2(dist)$* reduces the influence of geometry that is not in front of the viewer on the final result. To compute the final speed, we scale the length of the final displacement vector so that the user can never collide with objects.

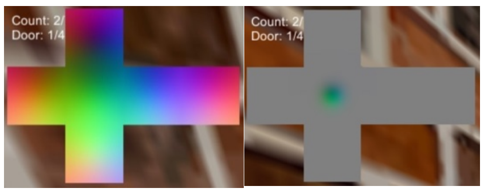

Fig. 1. Illustration of depth buffer without and with a second weighting term.

3 Evaluation

To evaluate our proposed automatic speed control and to compare it with the speed control via the global minimum described by McCrae et al. [16] as well as the automatic speed adjustment developed by Trindade et al. [21] we performed a user study.

We chose to evaluate our new technique with the mouse as input device, as we wanted to investigate one-handed operation (which frees the second hand for other operations). This keeps our navigation method open for other devices, such as touch screens and VR-style controllers. Similar to many games, any mouse movement without a button held down controls the user's view direction in our control scheme, while holding the left mouse button down will move the viewer forward, with mouse movements mapped to steering. The right mouse button is mapped to backward movement. We used only two buttons, to keep the user interface as simple as possible and to leave other buttons free for other purposes, such as object selection and manipulation.

3.1 Participants

We recruited 14 participants (11 male, 3 female) for this study, aged from 23 to 45 (mean age 31.8 years, *SD* 8.35). In the practice session, one participant found the task too difficult and declined to continue and another seemed to experience strong motion sickness symptoms and needed to be excluded. All remaining 12 participants had used VEs before, played FPS games, or 3D race car games.

3.2 Setup

The experiments were conducted on a generic PC with a nVidia GeForce GTX 970 on a 24" wide screen monitor (HP ZR24w) at 1920x1200, with a Microsoft IntelliMouse mouse as the input device. We did not use stereoscopic display.

To evaluate our new navigation technique, we were inspired by Argelaguet's experimental design and the VE they used [2]. Thus, our VE also includes a maze-like and a geometry-filled section, repeated across three different scales (1:1, 1:2 and 1:10).

To guide users we painted arrows on the maze walls to show the direction of the path to be followed. This helps if a user gets disoriented. We used textured walls (brick, stone) to enhance depth cues within the environment. To encourage all users to follow the same path we added rotating red cubes that users had to pick-up (see Fig. 3). Once the user collided with such a cube it was removed from the environment and the pick-up event was convoyed by a positive acoustic sound. Within the second geometry section of the VE these pick-up cubes were connected through thin rays, so that participants could always easily tell where to go next. The (previous) thin ray was always removed whenever the user reached a cube (see Fig. 3).

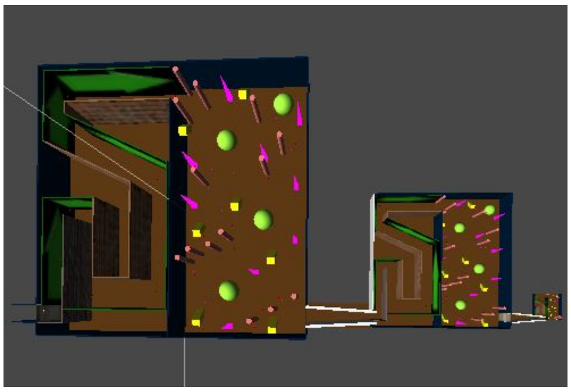

Fig. 2. VE used for the travel task. The VE is composed of 2 different sections replicated across three different levels of scale (1:1, 1:2 and 1:10).

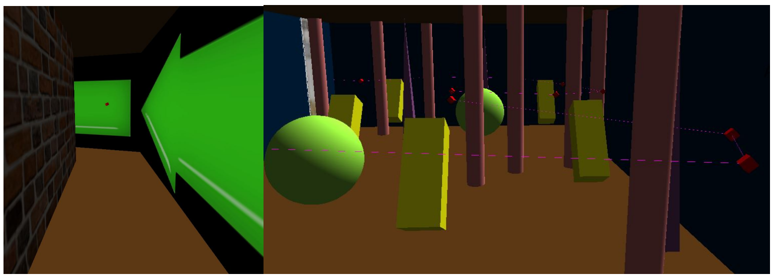

Fig. 3. (Left) First section of the environment, the maze, with directional arrows pointing towards pick-up objects and textured walls. (Right) Second section of the environment with geometry objects showing pick-up objects connected through thin rays.

3.3 Procedure

First, each participant was given a brief questionnaire about their background. The questionnaire recorded gender, age, and previous experience with 3D VEs. Then, the participant was instructed to use the UI and encouraged to practice until they felt comfortable. The mouse was the only means to navigate the environment. The left mouse button was mapped to forward movement, while the right mouse button initiated backward movement. With no button pressed the users could orient their view in the VE.

The order of the speed control techniques was counterbalanced with a Latin square design across all participants to minimize learning effects. The order of the sections (Maze, Geometry) and the scale factors was fixed due to the design of the VE (see Fig. 1). We did not counterbalance the sections, as the maze was easier to navigate.

And, the smaller mazes might be more difficult to navigate. All three techniques shared settings for the collision radius, smoothing term, and near and far plane distances.

Once the participants were comfortable with the VE, they were instructed to traverse the VE following the path marked by the pick-up cubes. Each participant was instructed to hit these target cubes as quickly and accurately as possible but not to be overly concerned if any given pick-up was not successful. To encourage participants to follow the path, we introduced a somewhat unpleasant audio cue for any missed pick-up.

At the end of the experiment, we gave participants a short questionnaire on their perceptions on the ease of use and navigation smoothness for all three techniques using 5-point Likert scales. Overall, the study took about half an hour per participant.

3.4 Results

Data was first filtered for clear participant errors, such as deviating from the sequence of cubes to pick up or pausing in the middle of the navigation sequence. We removed such errors by eliminating results with more than three standard deviations from the mean as outliers. This amounted to less than 3% of the total data.

3.5 Task Completion Time

The data for task completion time was not normally distributed. Levene's test for homogeneity revealed that the data did not have equal variances. We used the Aligned Rank Transform for nonparametric factorial data analysis [23] and then performed a repeated measures parametric ANOVA on the transformed data.

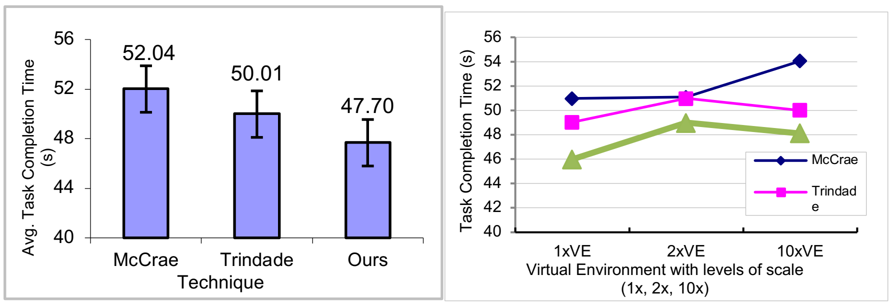

Fig. 4. (Left) Graph depicting the average task completion time for each technique, with standard deviations. (Right) Graph depicting the task completion time (s) for each VE scale.

There was a significant effect of completion time on technique ($F_{2,22} = 8.14$, $p < .01$). See Fig. 4 for task completion times for across all scales. A Tukey-Kramer post-hoc test revealed that our technique had a smaller completion time than both other techniques. Fig. 4 also shows task completion times for each scale of the environment.

An ANOVA test on group effect was not significant ($F_{5,6} = 0.652$), which confirmed that the counterbalancing cancelled out any potential learning effects. Further, we detected no significant learning effects across participants.

3.6 Average Speed

To analyze the average speed across all three scales, we multiplied the speeds from the half-scale environment by two and the speeds from the 1:10 scale by ten. After this adjustment the data was normally distributed.

The one-way ANOVA of technique versus speed showed a main effect on technique ($F_{2,22} = 3.39$, $p < .05$). See Fig. 5 for the average speeds.

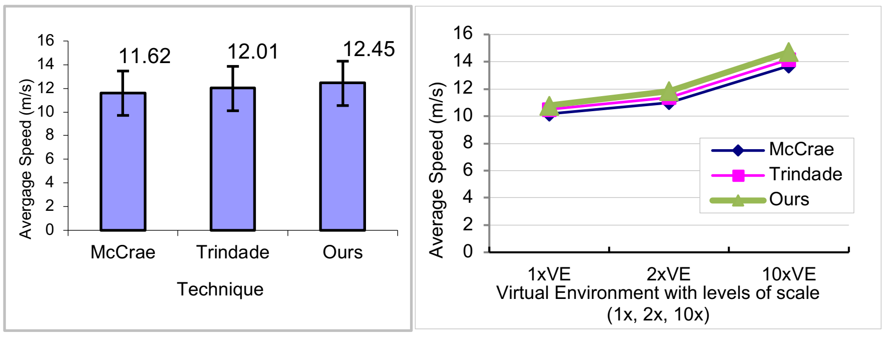

Fig. 5. (Left) Mean speed across all scales, with standard deviations. (Right) Average speed in m/s for each scale and technique.

A Tukey-Kramer test showed that the mean speed for our new technique was significantly higher than for Trindade's version, which in turn was higher than for McCrae's. Fig. 5 also shows that the ordering was consistent across techniques and scales.

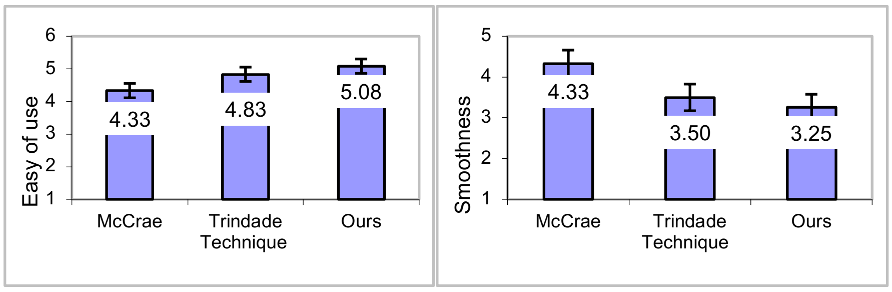

Fig. 6. (Left) Graph showing average user feedback for ease-of-use for each technique (higher is better), with standard deviations. (Right) Graph showing user feedback regarding the smoothness of speed changes for each technique (lower is better), with standard deviations.

From our observations of the participants and their comments during the experiment, we were able to identify that our new technique was perceived to be the best option. The data from the user questionnaire on the ease of use and the smoothness of the speed changes corroborates this insights. The outcomes confirm our observations (see Fig. 6). None of our 12 main participants reported discomfort or simulator sickness symptoms.

4 Discussion

The overall conclusion from this study is that our implementation allowed for smooth navigation with a performance improvement over the state-of-the-art travel speed control approaches. In comparison to two previously presented methods that are directly comparable, our method demonstrated a reduced completion time and improved speed, allowing the users to achieve the navigation goal in less time.

In general, our new solution seems to better address the issue that excessive slowdowns can occur with previous methods for automatic speed control, e.g., when the viewer slides along walls or navigates through tunnels. Our automatic speed control method takes the geometry that is in front of the viewer more strongly into account, instead of the whole surroundings (such as McCrae et al. [16]). In contrast to Trindade et al.'s work [21], we use an average over an area in front of the viewer, which eliminates the downsides of using only a single point. Our approach weights the influence of geometry behind or beside the viewer less than that of geometry directly ahead, with a smooth interpolation to guarantee smooth transitions. In essence, this allows the user to navigate parallel to an existing plane at a higher speed than would be achievable if navigating perpendicular towards the plane. By combining the influence of the surround geometry in a more robust way with the forward one, the navigation speed is adequate when navigating away from geometry, but still results in a smooth "takeoff" behavior.

The system presented here also generalizes to navigation on touchscreen systems used with a single hand, where a single finger touch/drag controls the look direction, while a 2-finger touch-and-hold will move the user forward based on the current user's view for the duration of the hold. Steering is then achieved by dragging both fingers in the desired direction. Backward movement can be mapped to a 3-finger touch event. Because the navigation techniques require only 2D input, our new method can also be easily used on other input devices, such as with a pen or a VR style controller.

We did not record sufficient data to analyze the participant's motion trails for signs of temporary disorientation. Yet, the lack of corresponding observations by the experimenter make it unlikely that this was a notable issue. Moreover, we observed only very few episodes with overshooting and subsequent backwards movements, or participants turning around. We believe that the root cause of this lack of overshoot is that the automatic speed control methods already reduce the speed of the participants sufficiently far in advance to enable them to adjust their path before they run into problems.

4.1 Limitations

As mentioned above, the computations result in a single displacement vector that pushes the user away from the nearest geometry. As mentioned by McCrae et al. [16] this will move the viewer towards the center of any cavity and can push the user out of rooms with openings. As the environment scale decreases, the magnitude of this vector increases and will start moving the user stronger away from nearby geometry, which can lead to surprising results and frustrating situations, e.g., when the viewer is extremely close to a surface while looking parallel to it. Then McCrae's algorithm will create a local "drift", even though the path in front of the user is clear, which could be addressed through a threshold that depends on the scale or another dynamically calculated value adapted to the desired use case.

Considering the size of the world relative to the smallest detail and the three different scales, we only explored up to a range of 5,000:1 in our experiment, with the speed decreasing at most by a factor of ten compared with the 1:1 scene. Initial experiments with larger scenes at approximately 1,000,000:1, revealed implementation-specific issues. The vector computations of McCrae et al.'s main equation [16] suffered quickly from lack of floating-point precision, which caused undesirable "jitter" effects. If we scale the environment up even further, i.e., explore scale differences of (say) 1 billion:1, a speed based on the minimum distance would in theory still adjust itself, but will likely also run into depth value precision issues.

The overall computing overhead of our (not fully optimized) shader-based implementation of our new automatic speed control method per frame was small: 2.88 ms compared to 2.52 ms for McCrae's and Trindade's versions, an increase of 12%.

5 Conclusion

In summary, the main contribution of this work is a method for automatic speed control for 3D travel in multi-scale virtual environments. We proposed a new and efficient way to automatically adapt a user's speed. This new method derives its benefit by taking the geometry that is in in front of the user better into account. By using shaders, this technique has also low overhead relative to CPU-based techniques. Future work will focus on applying the ideas behind our technique to navigation in vast empty spaces, such as a star system. Specifically, we will look at situations where the distance between objects is (far) beyond what can be represented on graphics hardware with floating-point number precision. We will also explore optimizations through clipping planes at the bounding radius.

References

1. Anthes, Christoph, Paul Heinzlreiter, Gerhard Kurka, Jens Volkert. "Navigation models for a flexible, multi-mode VR navigation framework." Virtual Reality continuum and its applications in industry, pp. 476-479. 2004.
2. Argelaguet-Sanz, Ferran. "Adaptive Navigation for Virtual Environments." Symposium on 3D User Interfaces 2014, pp. 91-94.
3. Bowman, Doug A., David Koller, Larry Hodges (1998). A Methodology for the Evaluation of Travel Techniques for Immersive Virtual Environments. Virtual Reality: Research, Development, and Applications 3: 120–131.
4. Butterworth, Jeff, Andrew Davidson, Stephen Hench, Marc T. Olano. "3DM: A three dimensional modeler using a head-mounted display." Symposium on Interactive 3D graphics, pp. 135-138. 1992.
5. Chung, James C. "A comparison of head-tracked and non-head-tracked steering modes in the targeting of radiotherapy treatment beams." Symposium on Interactive 3D Graphics 1992, pp. 193-196.

6. Duan, Qishen, Jianhua Gong, Wenhang Li, Shen Shen, and Rong Li. "Improved Cubemap model for 3D navigation in geo-virtual reality." *International Journal of Digital Earth* 8, no. 11 (2015): 877-900.
7. Fitzmaurice, George, Justin Matejka, Igor Mordatch, Azam Khan, and Gordon Kurtenbach. "Safe 3D navigation." Symposium on Interactive 3D Graphics, 2008, pp. 7-15.
8. Freitag, Sebastian, Benjamin Weyers, and Torsten W. Kuhlen. "Automatic speed adjustment for travel through immersive virtual environments based on viewpoint quality." Symposium on 3D User Interfaces (3DUI), pp. 67-70. 2016.
9. Freitag, Sebastian, Benjamin Weyers, and Torsten W. Kuhlen. "Interactive Exploration Assistance for Immersive Virtual Environments Based on Object Visibility and Viewpoint Quality." IEEE Virtual Reality Conference, pp. 355-362. IEEE, 2018.
10. Galyean, Tinsley A. "Guided navigation of virtual environments." Symposium on Interactive 3D Graphics, pp. 103-104. 1995.
11. Kemeny, Andras, Paul George, Frédéric Merienne, and Florent Colombet. "New VR Navigation Techniques to Reduce Cybersickness." In *The Engineering Reality of Virtual Reality*, pp. 48-53. 2017.
12. Kopper, Regis, Tao Ni, Doug A. Bowman, Marcio Pinho. "Design and evaluation of navigation techniques for multiscale virtual environments." IEEE Virtual Reality Conference, pp. 175-182. 2006.
13. Mackinlay, Jock D, Stuart K Card, George G Robertson. "Rapid controlled movement through a virtual 3D workspace." SIGGRAPH 1990. pp. 171-176.
14. Mapes, Daniel P., J. Moshell. "A two-handed interface for object manipulation in virtual environments." Presence: Teleoperators & Virtual Environments 4(4), 1995, pp. 403-416.
15. Mercurio, Philip J., Thomas Erickson, D. Diaper, D. Gilmore, G. Cockton, B. Shackel. "Interactive scientific visualization: An assessment of a virtual reality system." INTERACT, pp. 741-745. 1990.
16. McCrae, James, Igor Mordatch, Michael Glueck, Azam Khan. "Multiscale 3D navigation." Symposium on Interactive 3D Graphics, pp. 7-14. 2009.
17. Mine, Mark. "Virtual environment interaction techniques." *UNC Chapel Hill computer science technical report, TR95-018* (1995).
18. Mine, Mark R., Frederick P. Brooks Jr, Carlo H. Sequin. "Moving objects in space: exploiting proprioception in virtual-environment interaction." SIGGRAPH 1997, pp. 19-26.
19. Robinett, Warren, Richard Holloway. "Implementation of flying, scaling and grabbing in virtual worlds." Symposium on Interactive 3D Graphics. 1992, pp. 189-192.
20. Stuerzlinger, Wolfgang, Chadwick A. Wingrave. The value of constraints for 3D user interfaces. Virtual Realities: Dagstuhl Seminar 2008, Springer, 2011, pp. 203-224.
21. Trindade, Daniel R, Alberto B Raposo. "Improving 3D navigation in multiscale environments using cubemap-based techniques." Symposium on Applied Computing 2011, pp. 1215-1221.
22. Ware, Colin Daniel Fleet. "Context sensitive flying interface." Symposium on Interactive 3D Graphics, 1997, pp. 127-130.
23. Wobbrock, Jacob O., Leah Findlater, Darren Gergle, James J. Higgins. "The aligned rank transform for nonparametric factorial analyses using only anova procedures." ACM CHI Conference, pp. 143-146. 2011.