Implementing Walking in Virtual Environments

Gerd Bruder and Frank Steinicke

Abstract In the previous chapter, locomotion devices have been described, which prevent displacements in the real world while a user is walking. In this chapter we explain different strategies, which allow users to actually move through the real-world, while these physical displacements are mapped to motions of the camera in the virtual environment (VE) in order to support unlimited omnidirectional walking. Transferring a user's head movements from a physical workspace to a virtual scene is an essential component of any immersive VE. This chapter describes the pipeline of transformations from tracked real-world coordinates to coordinates of the VE. The chapter starts with an overview of different approaches for virtual walking, and gives an introduction to tracking volumes, coordinate systems and transformations required to set up a workspace for implementing virtual walking. The chapter continues with the traditional isometric mapping found in most immersive VEs, with special emphasis on combining walking in a restricted interaction volume via reference coordinates with virtual traveling metaphors (e. g., *flying*). Advanced mappings are then introduced with user-centric coordinates, which provide a basis to guide users on different paths in the physical workspace than what they experience in the virtual world.

1 Introduction

Using sophisticated hard- and software technology, immersive virtual environments (VEs) provide users with a multisensory medium for exploring and interacting with computer-generated three-dimensional environments. In particular, ego-centric perspectives and natural interaction metaphors can provide users with a compelling experience similar to interactions in the real world, which cannot be simulated using

Gerd Bruder and Frank Steinicke
University of Würzburg, Department of Human-Computer-Media, Campus Nord, Oswald-Külpe-Weg 82, D-97074 Würzburg, Germany, e-mail: {gerd.bruder|frank.steinicke}@uni-wuerzburg.de

any other existing technology. In this context, the most natural technique for exploring a virtual world is *real walking*, which provides a greater sense of presence than other virtual traveling techniques [4, 24], such as flying or walking in-place [36], and naturally stimulates human spatial wayfinding and cognitive map building [27]. As described in Part 1, walking is a form of natural locomotion, which encompasses repetitive motions of legs or body for active self-propulsion [9], such that users in immersive VEs receive proprioceptive, kinesthetic and efferent copy signals from their physical movements, supporting the perception of self-motion in the virtual world.

In order to provide users with an unimpaired sense of place and plausibility during self-motions [29], virtual reality (VR) applications have to maintain simultaneous awareness of coordinate systems and transformations in both the real and virtual world. In this chapter, we describe the basic transformations that can be used to implement real walking user interfaces in VR laboratory workspaces. In particular, we show how the sense of moving in computer graphics environments can be stimulated with sequences of frame to frame changes of the position and orientation of a user in a VR workspace. If the changes from one frame to the next are large, we talk of *teleportation*, whereas if the changes are considerable small, the feedback from the virtual world causes a sensory flow (e. g., optic flow [15] or acoustic flow [30]), which engenders the sense of continuous motion.

We distinguish between two main characteristics of real walking user interfaces:

- *Isometric* transformations describe mappings that preserve motion distances and angles when movements of a tracked user in the physical workspace are mapped to changes of a virtual representation.
- *Nonisometric* transformations, in contrast, describe different mapping approaches to introduce a discrepancy between user movements and virtual feedback.

It is generally assumed that human spatial perception and cognition in virtual worlds is optimally supported in isometric user interfaces, since sensory motion feedback from the user's physical movements (e. g., proprioceptive and vestibular motion cues) match feedback from the virtual world (e. g., optic and acoustic flow). However, isometric user interfaces have a severe practical problem: With such mappings the size of the physical workspace limits the size of the virtual scene that a user can explore by natural walking. We show how such limitations can be alleviated with multimodal interfaces that combine walking over short distances with traveling over long distances. We introduce nonisometric mapping strategies that provide a different solution to the problem of unrestricted omnidirectional walking by guiding users on a different path in the real world than experienced in the virtual scene. Nonisometric mappings for walking user interfaces are encompassed under the term *redirected walking* [23].

The remainder of this chapter is structured as follows. Section 2 gives a short introduction to workspaces and coordinate systems in VR laboratories. In Section 3 we present the basic math and algorithms necessary to implement isometric virtual

walking, and then show with reference coordinates how limitations of virtual interaction space can be alleviated with traveling techniques. In Section 4 we describe nonisometric transformations for redirected walking, give an overview of the basic algorithms with user-centric coordinates, and go into detail on linear and angular scaling transformations, as well as curvature mappings. We present a simple algorithm that allows practitioners to implement unrestricted redirected walking in VR workspaces. Section 5 concludes the chapter.

2 Virtual Reality Workspaces

In order to support real walking, user movements in a VR laboratory have to be tracked and mapped to motions in a three-dimensional virtual scene. In particular, movements of the user's head position in the physical workspace have to be measured and transferred to motions of camera objects in the virtual space in order to provide ego-centric visual feedback to the user's eyes from the virtual world[1].

Physical workspaces in VR laboratories incorporate tracking systems to measure the position and/or orientation of objects located in the tracking space. Such tracking systems can differ in underlying technology, accuracy and precision of tracking data, as well as how the user is instrumented. In particular, some VR laboratories incorporate separate tracking systems for position and orientation measurements, such as optical marker tracking systems that measure the head position and inertial orientation sensors that measure the head orientation. The coordinate systems in which tracking systems provide position and orientation data are not standardized, such that usually the tracking coordinates have to be transformed into the coordinate system used for the virtual scene [8]. In the following, for convenience, we assume that virtual and physical coordinate systems are calibrated and represented in right-handed OpenGL coordinates [28]. Therefore, the y-axis is oriented in inverse gravitation direction, whereas the x- and z-axis are orthogonal to the y-axis and each other, thus defining the ground plane. These coordinates can easily be derived from arbitrary tracking coordinates by reassigning the x-, y- and z-axes, or multiplying the z-coordinate with -1 for changing the handedness.

Figure 1 illustrates such a coordinate system in a tracked workspace in a VR laboratory. Position and orientation of tracked objects can be described as a transformation from a specified origin of the tracking volume to the object's local coordinate system. Tracking systems often provide position data as a translation vector $(x_r, y_r, z_r) \in \mathbb{R}^3$, and orientation data as yaw, pitch and roll angles $(\tilde{y}_r, \tilde{p}_r, \tilde{r}_r) \in [0, 360)^3$, describing three subsequently applied rotational transformations[2]. Although the axes and order of yaw, pitch and roll transformations are not standardized

[1] While most immersive VEs implement head tracking for visual feedback, some laboratories also implement tracking of other body parts to provide virtual body feedback or interaction methods.

[2] Some tracking systems use quaternions as their native reporting format, which provides an alternative representation of the transformations, and can be converted from and to the angular notation used in this chapter [21].

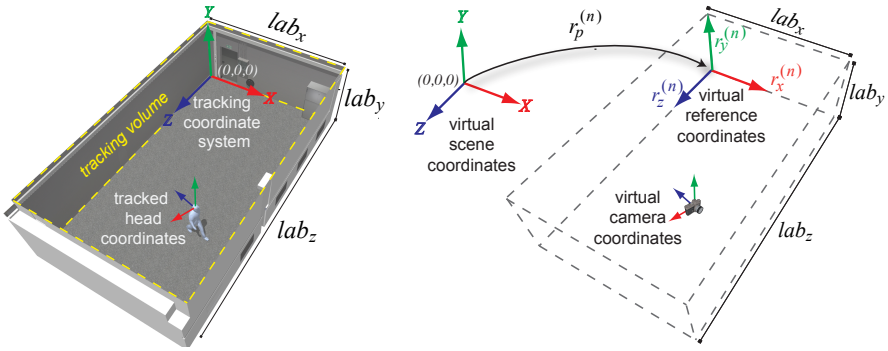


Fig. 1 Illustration of (a) three-dimensional tracking coordinates in a VR laboratory with a user's tracked local head coordinate system as defined by positional and orientational tracking data, and (b) virtual scene coordinates with interaction volume defined by reference coordinates.

among tracking systems, practitioners tend to represent orientations first with yaw rotations around the y-axis, followed by pitch rotations around the x-axis, and roll rotations around the z-axis [35]. For instance, assuming the user's head position and orientation is tracked with a local coordinate system that is defined with the z-axis for the (inverse) *look*-direction, the y-axis in *up*-direction, and the x-axis in *strafe*-direction (see Figure 1(a)), then yaw, pitch and roll rotations correspond to turning the head to the left or right, up or down, or around the view axis, respectively. In the following, we assume tracked head orientations to be provided using this representation.

In order to provide a user with visual feedback by rendering the three-dimensional scene onto one or more VR display surfaces, we have to consider a virtual analog of the user's head in the VE[3]. We assume virtual camera coordinates are represented with the triple of orthogonal axes as used for physical head tracking coordinates [8], with transformations from the origin of the virtual scene to the camera coordinates defined by a translation vector $(x_v, y_v, z_v) \in \mathbb{R}^3$, and yaw, pitch and roll angles $(\tilde{y}_v, \tilde{p}_v, \tilde{r}_v) \in [0, 360)^3$. Figure 1(b) illustrates local camera coordinates in virtual scene coordinates.

[3] Depending on the display system (e. g., head-mounted displays or immersive projection technologies) the actual positions or orientations of computer graphics camera objects are usually specified relative to these head coordinates, such as transformations from the head center to the eye displays [25].

3 Isometric Virtual Walking

In this section we present the basic math and algorithms to implement isometric real walking transformations, i. e., mappings that preserve distances and angles of a user's movements.

3.1 One-to-one Mappings

Assuming the real and virtual workspaces are defined using the coordinate systems introduced in Section 2, basic one-to-one mappings can be implemented by using the tracked position and orientation of a user's head in the laboratory to define the position and orientation of a corresponding virtual camera object for each rendering frame. In particular, a tracked change of one unit (e. g., meter or degree) in the physical workspace is mapped to a translation or rotation of one unit in the virtual scene. Examples of such mappings are often found when displaying a virtual replica of a virtual reality laboratory to users in head-mounted display environments [12, 33], or in architectural passive haptics environments, in which real and virtual objects are registered to provide users with haptic feedback when touching virtual objects [10]. In such environments, one-to-one mappings can be implemented using the following simple pseudo code:

Algorithm 1 One-to-one mapping from tracked head to camera coordinates

for all rendering frames $n \in \mathbb{N}_0$ **do**
  // Get current head tracking state:
  $(x_r^{(n)}, y_r^{(n)}, z_r^{(n)}) \leftarrow$ tracked head position (in $\mathbb{R}^3$)
  $(\tilde{y}_r^{(n)}, \tilde{p}_r^{(n)}, \tilde{r}_r^{(n)}) \leftarrow$ tracked head orientation (in $[0,360)^3$)

  // Set virtual camera state:
  $(x_v^{(n)}, y_v^{(n)}, z_v^{(n)}) \leftarrow (x_r^{(n)}, y_r^{(n)}, z_r^{(n)})$ *// position*
  $(\tilde{y}_v^{(n)}, \tilde{p}_v^{(n)}, \tilde{r}_v^{(n)}) \leftarrow (\tilde{y}_r^{(n)}, \tilde{p}_r^{(n)}, \tilde{r}_r^{(n)})$ *// orientation*
end for

In the pseudo code, $(x_r^{(n)}, y_r^{(n)}, z_r^{(n)}) \in \mathbb{R}^3$ denotes the current three-dimensional position, and $(\tilde{y}_r^{(n)}, \tilde{p}_r^{(n)}, \tilde{r}_r^{(n)}) \in [0,360)^3$ the current yaw, pitch and roll orientation of the user's head in the physical workspace as provided by the tracking system for rendering frames $n \in \mathbb{N}_0$, as well as $(x_v^{(n)}, y_v^{(n)}, z_v^{(n)}) \in \mathbb{R}^3$ the computed new position and $(\tilde{y}_v^{(n)}, \tilde{p}_v^{(n)}, \tilde{r}_v^{(n)}) \in [0,360)^3$ the new orientation of the camera object that is used as the basis for rendering the current frame to be displayed to the user.

3.2 Reference Coordinates

A simple extension of isometric mappings is to introduce virtual *reference coordinates*. Since one-to-one mappings only allow a user to explore a volume in the virtual scene that is exactly as large as the interaction volume in the laboratory, it becomes important to map the user's movements to specific regions of interest in the virtual scene. This can be accomplished by introducing an intermediate reference coordinate system when transferring position and orientation data from tracking coordinates to virtual scene coordinates. Introducing such virtual reference coordinates, the corresponding pseudo code in Section 3.1 changes to:

Algorithm 2 Isometric mapping with reference coordinates

for all rendering frames $n \in \mathbb{N}_0$ **do**
  // Get current head tracking state:
  $(x_r^{(n)}, y_r^{(n)}, z_r^{(n)}) \leftarrow$ tracked head position (in $\mathbb{R}^3$)
  $(\tilde{y}_r^{(n)}, \tilde{p}_r^{(n)}, \tilde{r}_r^{(n)}) \leftarrow$ tracked head orientation (in $[0,360)^3$)

  // Set virtual camera state:
  $(x_v^{(n)}, y_v^{(n)}, z_v^{(n)}, 1)^{\mathrm{T}} \leftarrow \begin{pmatrix} r_{x_x}^{(n)} & r_{y_x}^{(n)} & r_{z_x}^{(n)} & r_{p_x}^{(n)} \\ r_{x_y}^{(n)} & r_{y_y}^{(n)} & r_{z_y}^{(n)} & r_{p_y}^{(n)} \\ r_{x_z}^{(n)} & r_{y_z}^{(n)} & r_{z_z}^{(n)} & r_{p_z}^{(n)} \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot (x_r^{(n)}, y_r^{(n)}, z_r^{(n)}, 1)^{\mathrm{T}}$ *// position*
  $(\tilde{y}_v^{(n)}, \tilde{p}_v^{(n)}, \tilde{r}_v^{(n)}) \leftarrow (\tilde{y}_r^{(n)}, \tilde{p}_r^{(n)}, \tilde{r}_r^{(n)})$ *// orientation*
end for

In the pseudo code, the 4×4 transformation matrix for homogenous coordinates defines a reference position $r_p^{(n)} = (r_{p_x}^{(n)}, r_{p_y}^{(n)}, r_{p_z}^{(n)}) \in \mathbb{R}^3$ in the virtual scene, as well as coordinate axes with the direction vectors $r_x^{(n)} = (r_{x_x}^{(n)}, r_{x_y}^{(n)}, r_{x_z}^{(n)}) \in \mathbb{R}^3$, $r_y^{(n)} = (r_{y_x}^{(n)}, r_{y_y}^{(n)}, r_{y_z}^{(n)}) \in \mathbb{R}^3$, and $r_z^{(n)} = (r_{z_x}^{(n)}, r_{z_y}^{(n)}, r_{z_z}^{(n)}) \in \mathbb{R}^3$ along the transformed x-, y- and z-axes of the reference coordinates. The virtual yaw, pitch and roll transformations are applied to the reference coordinate axes. Figure 1 illustrates reference coordinates that are used to move the virtual interaction volume (limited by the size of the laboratory workspace $(lab_x, lab_y, lab_z) \in \mathbb{R}^3$) to regions of interest.

To account for changing regions of interest in the virtual scene, reference coordinates can be changed at run time. In particular, teleportation of the user's virtual viewpoint can be implemented by abrupt changes of the reference coordinates, whereas continuous traveling can be implemented by iterative changes in reference positions and orientations [3, 18, 24].

3.3 Virtual Traveling

Instead of just implementing natural walking, many immersive VEs make use of a hybrid walking-and-flying metaphor, in which the user's head is tracked in a limited interaction space, whereas the user can change the reference position or orientation in the virtual environment by using a hand-held controller. This interaction technique can easily be grasped by users when introduced as a *flying carpet* [40], i. e., the user can naturally walk over a limited carpet region, while the carpet itself can be flown through the virtual world. In contrast to real walking, which is identified by natural locomotion in the physical workspace, flying and steering techniques are denoted as virtual *traveling* [3]. A traditional implementation of virtual traveling is view-directed flying [3, 18, 24], which refers to user-initiated changes of reference coordinates relative to the user's virtual view, i. e., the coordinates of the virtual camera object. With view-directed flying in immersive virtual environments usually only the reference position is changed, whereas the orientation of reference coordinates is not affected, such that the virtual interaction volume remains level to the real world [18].

A basic virtual flying controller can be implemented using the following simple approach. For each rendering frame $n \in \mathbb{N}$ we compute the current view-direction $(v_{v_x}^{(n)}, v_{v_y}^{(n)}, v_{v_z}^{(n)}) \in \mathbb{R}^3$, the strafe-direction $(s_{v_x}^{(n)}, s_{v_y}^{(n)}, s_{v_z}^{(n)}) \in \mathbb{R}^3$, and the up-direction $(u_{v_x}^{(n)}, u_{v_y}^{(n)}, u_{v_z}^{(n)}) \in \mathbb{R}^3$ of the camera object in the VE (see Section 2). Providing commodity input hardware to the user, such as a keyboard, the user can initiate changes in reference coordinates by pressing different keys. For instance, if we detect that the user has pressed the up- or down-key on a keyboard, we compute the reference position for the next rendering frame as

$$(r_{p_x}^{(n)}, r_{p_y}^{(n)}, r_{p_z}^{(n)}) = (r_{p_x}^{(n-1)}, r_{p_y}^{(n-1)}, r_{p_z}^{(n-1)}) + (v_{v_x}^{(n)}, v_{v_y}^{(n)}, v_{v_z}^{(n)}) \cdot g_v^{(n)},$$

with $g_v^{(n)} \in \mathbb{R}$ defining a speed factor for virtual traveling in view direction, e. g., with $g_v^{(n)} > 0$ for forward motions if the user pressed the up-key, and $g_v^{(n)} < 0$ for backward motions if the user pressed the down-key on the keyboard. In particular, this means that the user can turn the head towards a target in the virtual scene, and fly towards the virtual target by pressing the up-key. Using corresponding keys and speed factors, we can allow the user to change the reference position in the virtual scene not only in view-direction, but also in strafe-direction $(s_{v_x}^{(n)}, s_{v_y}^{(n)}, s_{v_z}^{(n)})$, and up-direction $(u_{v_x}^{(n)}, u_{v_y}^{(n)}, u_{v_z}^{(n)})$. The speed factors may be as simple as a constant or a more sophisticated function based on sensor inputs.

4 Nonisometric Virtual Walking

Isometric mappings enable users to explore a virtual region by real walking. However, since with isometric mappings the virtual interaction space is limited, virtual traveling techniques have to be used to cover larger distances in the VE, which impair the user's sense of being able to explore a VE like the real world, and can significantly degrade spatial perception and task performance [27, 36]. To alleviate such problems, researchers proposed nonisometric mappings, which have the potential to enable unrestricted omnidirectional walking. In order to describe such mappings, we introduce relative user-centric coordinates.

4.1 User-Centric Coordinates

Instead of mapping position and orientation data from the tracking volume in the laboratory for each rendering frame to their respective absolute position and orientation in a fixed virtual interaction volume, redirection techniques are based on relative mappings, in which each change in position or orientation from one rendering frame to the next is addressed separately [32]. Using user-centric relative coordinates requires more transformations and may introduce numerical error propagation, but allows more sophisticated mapping strategies.

From Absolute Position and Orientation to Relative Changes

For each rendering frame the change in position and orientation of the user's head is measured in coordinates of the tracking volume. Changes can be computed as the difference of the current tracking data at rendering frame $n \in \mathbb{N}$ from the previous state at rendering frame $n-1$, defined by tuples consisting of the previous position $(x_r^{(n-1)}, y_r^{(n-1)}, z_r^{(n-1)}) \in \mathbb{R}^3$ and orientation $(\tilde{y}_r^{(n-1)}, \tilde{p}_r^{(n-1)}, \tilde{r}_r^{(n-1)}) \in [0,360)^3$, and the current position $(x_r^{(n)}, y_r^{(n)}, z_r^{(n)}) \in \mathbb{R}^3$ and orientation $(\tilde{y}_r^{(n)}, \tilde{p}_r^{(n)}, \tilde{r}_r^{(n)}) \in [0,360)^3$ in the real-world tracking volume. The three-dimensional head position change $(\Delta x_r^{(n)}, \Delta y_r^{(n)}, \Delta z_r^{(n)}) \in \mathbb{R}^3$, as well as the changes in yaw, pitch and roll head orientation angles $(\Delta \tilde{y}_r^{(n)}, \Delta \tilde{p}_r^{(n)}, \Delta \tilde{r}_r^{(n)}) \in [-180,180)^3$ result as:

$$\begin{cases} \Delta x_r^{(n)} = x_r^{(n)} - x_r^{(n-1)}, \\ \Delta y_r^{(n)} = y_r^{(n)} - y_r^{(n-1)}, \\ \Delta z_r^{(n)} = z_r^{(n)} - z_r^{(n-1)}, \end{cases}$$

$$\begin{cases} \Delta\tilde{y}_r^{(n)} &= atan2(\ sin(\tilde{y}_r^{(n)} - \tilde{y}_r^{(n-1)}),\ cos(\tilde{y}_r^{(n)} - \tilde{y}_r^{(n-1)})\ ), \\ \Delta\tilde{p}_r^{(n)} &= atan2(\ sin(\tilde{p}_r^{(n)} - \tilde{p}_r^{(n-1)}), cos(\tilde{p}_r^{(n)} - \tilde{p}_r^{(n-1)})\ ), \\ \Delta\tilde{r}_r^{(n)} &= atan2(\ sin(\tilde{r}_r^{(n)} - \tilde{r}_r^{(n-1)}),\ cos(\tilde{r}_r^{(n)} - \tilde{r}_r^{(n-1)})\ ). \end{cases}$$

It should be noted that computing the angular difference from one frame to the next is not trivial. In this computation we assume that the user's head rotation between the previous and current frame did not exceed 180 degrees in each dimension, which is a reasonable assumption in real-time simulations. Therefore, we compute the smaller of the two angles in each direction that can lead from the previous orientation angle to the current angle (i. e., rotating clockwise or counterclockwise). The computed $\Delta\tilde{y}_r^{(n)}$, $\Delta\tilde{p}_r^{(n)}$ and $\Delta\tilde{r}_r^{(n)}$ angles are in the interval $[-180, 180)$.

Mapping Relative Changes

Linear and angular changes of the user's head pose in tracking coordinates have to be mapped to the virtual environment for each rendering frame, i. e., the virtual position and orientation result from accumulation of relative differences measured in the physical tracking volume. We can describe a one-to-one relative mapping for all linear and angular movements from the tracking volume to virtual coordinates in pseudo code as follows:

Algorithm 3 Relative mapping from tracked head to camera coordinates

for all rendering frames $n \in \mathbb{N}$ **do**
  // Get current head tracking state:
  $(x_r^{(n)}, y_r^{(n)}, z_r^{(n)}) \leftarrow$ tracked head position (in $\mathbb{R}^3$)
  $(\tilde{y}_r^{(n)}, \tilde{p}_r^{(n)}, \tilde{r}_r^{(n)}) \leftarrow$ tracked head orientation (in $[0, 360)^3$)

  // Compute relative changes:
  $(\Delta x_r^{(n)}, \Delta y_r^{(n)}, \Delta z_r^{(n)}) \leftarrow$ head position change (in $\mathbb{R}^3$)
  $(\Delta\tilde{y}_r^{(n)}, \Delta\tilde{p}_r^{(n)}, \Delta\tilde{r}_r^{(n)}) \leftarrow$ head orientation change (in $[-180, 180)^3$)

  // Set virtual camera state:
  $(x_v^{(n)}, y_v^{(n)}, z_v^{(n)}) \leftarrow (x_v^{(n-1)}, y_v^{(n-1)}, z_v^{(n-1)}) + (\Delta x_r^{(n)}, \Delta y_r^{(n)}, \Delta z_r^{(n)})$ *// position*
  $(\tilde{y}_v^{(n)}, \tilde{p}_v^{(n)}, \tilde{r}_v^{(n)}) \leftarrow (\tilde{y}_v^{(n-1)}, \tilde{p}_v^{(n-1)}, \tilde{r}_v^{(n-1)}) + (\Delta\tilde{y}_r^{(n)}, \Delta\tilde{p}_r^{(n)}, \Delta\tilde{r}_r^{(n)})$ *// orientation*
end for

This approach describes relative transformations from one rendering frame to the next, i. e., it is reasonable to initialize the virtual position and orientation of the user at the beginning of the VR experience with the identical state as in the tracking volume, or to define an initial offset using reference coordinates as described for isometric mappings (see Section 3.2).

Local Frames of Reference

In contrast to the absolute position and orientation of the user in tracking coordinates, such relative changes are independent of a specific origin defined in the tracking volume. However, the relative changes are not independent of the specific axes defined in the tracking space. In particular, a movement of a user's head is described as a position change for frame $n \in \mathbb{N}$ as $(\Delta x_r^{(n)}, \Delta y_r^{(n)}, \Delta z_r^{(n)}) \in \mathbb{R}^3$ along the x-, y- and z-axes of the tracking volume. Most advanced redirection techniques, however, manipulate position or orientation changes relative to specific coordinates (e. g., determined from the user's head or body state) in the real and virtual world. We can account for such frames of reference by introducing local coordinate transforms, for which the virtual camera transformation at frame $n \in \mathbb{N}$ changes to:

$$\begin{pmatrix} x_v^{(n)} \\ y_v^{(n)} \\ z_v^{(n)} \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & x_v^{(n-1)} \\ 0 & 1 & 0 & y_v^{(n-1)} \\ 0 & 0 & 1 & z_v^{(n-1)} \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot M_v \cdot M_r \cdot \begin{pmatrix} \Delta x_r^{(n)} \\ \Delta y_r^{(n)} \\ \Delta z_r^{(n)} \\ 1 \end{pmatrix},$$

with homogeneous coordinates and 4×4 matrices M_r and M_v defining local coordinate transformations in real and virtual coordinates, respectively. An example of such transformations is discussed in the following section.

4.2 Scaling Self-Motions

The most often found redirection techniques are based on nonisometric mappings of user-centric translations or rotations to virtual camera motions. Such mappings can be described by *self-motion gains*, which define ratios between real and virtual self-motions. Two types of self-motion gains are distinguished in immersive virtual environments, i. e., *rotation gains* and *translation gains*.

Rotation Gains

Rotation gains define the ratio between physical head turns and virtual camera rotations [32, 16]. Assuming a relative change in the orientation of the user's head has been determined for frame $n \in \mathbb{N}$ as $(\Delta\tilde{y}_r^{(n)}, \Delta\tilde{p}_r^{(n)}, \Delta\tilde{r}_r^{(n)}) \in \mathbb{R}^3$, rotation gains $g_R = (g_{R[\tilde{y}]}, g_{R[\tilde{p}]}, g_{R[\tilde{r}]}) \in \mathbb{R}^3$ define the resulting virtual camera rotation, which changes to:

$$\begin{pmatrix} \tilde{y}_v^{(n)} \\ \tilde{p}_v^{(n)} \\ \tilde{r}_v^{(n)} \end{pmatrix} = \begin{pmatrix} \tilde{y}_v^{(n-1)} \\ \tilde{p}_v^{(n-1)} \\ \tilde{r}_v^{(n-1)} \end{pmatrix} + \begin{pmatrix} g_{R[\tilde{y}]} & 0 & 0 \\ 0 & g_{R[\tilde{p}]} & 0 \\ 0 & 0 & g_{R[\tilde{r}]} \end{pmatrix} \cdot \begin{pmatrix} \Delta\tilde{y}_r^{(n)} \\ \Delta\tilde{p}_r^{(n)} \\ \Delta\tilde{r}_r^{(n)} \end{pmatrix}$$

Most redirection techniques focus on scaling yaw rotations [13, 22, 23], for which an applied rotation gain $g_{R[\tilde{y}]} \in \mathbb{R}$ causes a tracked real-world head rotation $\Delta\tilde{y}_r^{(n)}$ to cause a virtual camera rotation of $g_{R[\tilde{y}]} \cdot \Delta\tilde{y}_r^{(n)}$, instead of $\Delta\tilde{y}_r^{(n)}$. This means that if $g_{R[\tilde{y}]}$=1 the virtual scene remains stable considering a user's head orientation change. In case of $g_{R[\tilde{y}]}$>1 the virtual scene appears to rotate against the direction of the head turn, whereas a gain $g_{R[\tilde{y}]}$<1 causes the scene to rotate with the direction of the head turn [13]. For instance, if a user rotates the head by a yaw angle of 90 degrees, a gain $g_{R[\tilde{y}]}$=1 maps this motion one-to-one to a 90 degrees rotation of the virtual camera in the VE. Applying a gain of $g_{R[\tilde{y}]}$=0.5 results in the user having to rotate the head by 180 degrees physically in order to achieve a 90 degrees virtual rotation. A gain of $g_{R[\tilde{y}]}$=2 results in the user having to rotate the head by only 45 degrees physically in order to achieve a 90 degrees virtual rotation.

In case such rotation gains cause differences between a user's head orientation in tracking coordinates, and a camera orientation in virtual scene coordinates, this requires us to adapt the direction of subsequent translational movements to the offset between the real and virtual head orientation. We can account for such offsets by introducing user-centric reference coordinates for translational movements in the real and virtual environment (see Section 4.1). For instance, in the example above, we can account for offsets between real and virtual yaw orientation angles by defining local coordinate transforms for position changes:

$$\begin{pmatrix} x_v^{(n)} \\ y_v^{(n)} \\ z_v^{(n)} \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & x_v^{(n-1)} \\ 0 & 1 & 0 & y_v^{(n-1)} \\ 0 & 0 & 1 & z_v^{(n-1)} \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot$$

$$\begin{pmatrix} cos(\tilde{y}_v^{(n-1)}) & 0 & sin(\tilde{y}_v^{(n-1)}) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(\tilde{y}_v^{(n-1)}) & 0 & cos(\tilde{y}_v^{(n-1)}) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} cos(-\tilde{y}_r^{(n-1)}) & 0 & sin(-\tilde{y}_r^{(n-1)}) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(-\tilde{y}_r^{(n-1)}) & 0 & cos(-\tilde{y}_r^{(n-1)}) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \Delta x_r^{(n)} \\ \Delta y_r^{(n)} \\ \Delta z_r^{(n)} \\ 1 \end{pmatrix},$$

in which head position changes in tracking coordinates are first transformed into a local coordinate system relative to the yaw orientation angle of the user's head in the previous rendering frame, and then transformed into the local coordinate system relative to the yaw orientation angle of the camera object in virtual coordinates at the previous rendering frame. Using this simple coordinate transformation, we can apply yaw rotation gains without changing the mapping of head translations relative to the user's head orientation. At this point it should be noted that similar transformations can be applied for pitch and roll transformations, e. g., to simulate virtual slopes [17]. However, since pitch and roll angles are usually applied sequentially relative to the virtual camera yaw angle (see Section 2), in most cases it is not necessary to introduce such coordinate transformations to account for applied pitch or roll gains (cf. [2]).

Translation Gains

Translation gains define the ratio between real and virtual head translations [32]. Similar to rotation gains, scaled translations can be described with translation gains $g_T = (g_{T[x]}, g_{T[y]}, g_{T[z]}) \in \mathbb{R}^3$, which are applied to relative changes in the position of the user's head $(\Delta x_r^{(n)}, \Delta y_r^{(n)}, \Delta z_r^{(n)}) \in \mathbb{R}^3$ for frame $n \in \mathbb{N}$:

$$\begin{pmatrix} x_v^{(n)} \\ y_v^{(n)} \\ z_v^{(n)} \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & x_v^{(n-1)} \\ 0 & 1 & 0 & y_v^{(n-1)} \\ 0 & 0 & 1 & z_v^{(n-1)} \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} g_{T[x]} & 0 & 0 & 0 \\ 0 & g_{T[y]} & 0 & 0 \\ 0 & 0 & g_{T[z]} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \Delta x_r^{(n)} \\ \Delta y_r^{(n)} \\ \Delta z_r^{(n)} \\ 1 \end{pmatrix}$$

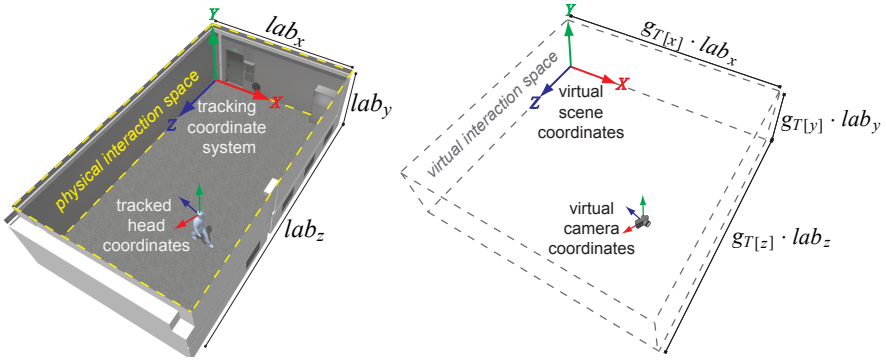
For instance, uniform scalings in horizontal walking directions are often applied in immersive virtual environments allowing users to cover a larger distance in the VE when walking in the physical workspace [37], which can be described with translations gains $g_{T[x]}=g_{T[z]}>1$, and $g_{T[y]}=1$ (see Figure 2). This causes a position change of the user's head in the real world $(\Delta x_r^{(n)}, \Delta y_r^{(n)}, \Delta z_r^{(n)}) \in \mathbb{R}^3$ to be transferred to the VE as $(g_{T[x]} \cdot \Delta x_r^{(n)}, \Delta y_r^{(n)}, g_{T[x]} \cdot \Delta z_r^{(n)})$, i. e., horizontal movements along the x- and z-axes are scaled uniformly, whereas vertical head bobbing movements along the y-axis are unaffected.

However, this approach still results in the problem that lateral head movements are scaled while a user walks, which can be distracting for the user [11]. Instead of scaling all horizontal motions with a translation gain, Interrante et al. [11] proposed scaling translations only in a user-specified walking direction (i. e., the *seven league boots* metaphor). Using a similar approach, Steinicke et al. [32] proposed using the yaw orientation of the user's head as approximation of walking direction [1] to scale translational movements. The latter approach can be implemented even without additional user instrumentation by changing the mapping to:

$$\begin{pmatrix} x_v^{(n)} \\ y_v^{(n)} \\ z_v^{(n)} \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & x_v^{(n-1)} \\ 0 & 1 & 0 & y_v^{(n-1)} \\ 0 & 0 & 1 & z_v^{(n-1)} \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} cos(\tilde{y}_v^{(n-1)}) & 0 & sin(\tilde{y}_v^{(n-1)}) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(\tilde{y}_v^{(n-1)}) & 0 & cos(\tilde{y}_v^{(n-1)}) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot$$

$$\begin{pmatrix} g_{T[x]} & 0 & 0 & 0 \\ 0 & g_{T[y]} & 0 & 0 \\ 0 & 0 & g_{T[z]} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} cos(-\tilde{y}_r^{(n-1)}) & 0 & sin(-\tilde{y}_r^{(n-1)}) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(-\tilde{y}_r^{(n-1)}) & 0 & cos(-\tilde{y}_r^{(n-1)}) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \Delta x_r^{(n)} \\ \Delta y_r^{(n)} \\ \Delta z_r^{(n)} \\ 1 \end{pmatrix},$$

which allows to scale head position changes with separate gains relative to the user's locomotion state. In particular, walking distances in the virtual heading direction can be scaled with a gain $g_{T[z]} \in \mathbb{R}$, lateral distances can be scaled with a gain $g_{T[x]} \in \mathbb{R}$, and vertical distances can be scaled with a gain $g_{T[y]} \in \mathbb{R}$.



(a) physical volume **(b)** scaled virtual volume

Fig. 2 Illustration of translation gains: (a) physical interaction volume in the virtual reality laboratory with size $(lab_x, lab_y, lab_z) \in \mathbb{R}^3$, and (b) virtual environment interaction volume with size $(g_{T[x]} \cdot lab_x, g_{T[y]} \cdot lab_y, g_{T[z]} \cdot lab_z) \in \mathbb{R}^3$ scaled by translation gains.

4.3 Redirected Walking

Although scaling self-motions as introduced in Section 4.2 can be used to redirect a user, e. g., by scaling head rotations to reorient the user away from an obstacle in the physical workspace, the approach has practical limitations. In particular, assuming the user walks straight ahead in the laboratory workspace without performing head rotations, then the virtual travel distance can be scaled relative to the physical walking distance, but at some point the user will eventually reach the end of the physical workspace, and potentially collide with an obstacle. To avoid this problem, researchers proposed various solutions [7, 14, 19, 20, 22, 23, 26, 32, 34, 38, 39], including techniques based on instructing the user to stop walking and start rotating the head, such that rotation gains can be applied to reorient the user away from physical obstacles [22, 38]. However, the most prominent solution for unrestricted walking was presented by Razzaque et al. [23], who proposed to use subtle virtual camera rotations while a user performs translational movements in the physical laboratory workspace. This causes the user to change the heading direction when walking in the real world according to the rotations in the virtual environment. The approach can be implemented with *curvature gains*.

Curvature Gains

Curvature gains define ratios between position changes of the user's head in the real world and virtual camera rotations [32]. For example, when the user walks straight ahead in the physical workspace, a curvature gain that causes reasonably small iterative camera yaw rotations to one side forces the user to walk along a curved path in the opposite direction in the real world in order to stay on a straight

path in the virtual world. If the injected manipulations are reasonably small, the user will compensate for the virtual camera rotations without being able to consciously detect the manipulations. Curvature gains $g_C \in \mathbb{R}_0^+$ denote the resulting bending of a user's path in the physical workspace, which is determined as $g_C = 1/r$ for a circular arc with radius $r \in \mathbb{R}^+$. In case no curvature is applied, i. e., $r = \infty$, this corresponds to a curvature gain $g_C = 0$. If an applied curvature gain causes the user to rotate by 90 degrees after $\frac{\Pi}{2}$m walking distance, then the user has covered a quarter circle with radius $r = 1$, which corresponds to a curvature gain $g_C = 1$.

Curvature mappings can be described using the following pseudo code:

Algorithm 4 Curvature mapping from tracked head to camera coordinates

for all rendering frames $n \in \mathbb{N}$ **do**

// Get current head tracking state:

$(x_r^{(n)}, y_r^{(n)}, z_r^{(n)}) \leftarrow$ tracked head position (in $\mathbb{R}^3$)

$(\tilde{y}_r^{(n)}, \tilde{p}_r^{(n)}, \tilde{r}_r^{(n)}) \leftarrow$ tracked head orientation (in $[0,360)^3$)

// Compute relative changes:

$(\Delta x_r^{(n)}, \Delta y_r^{(n)}, \Delta z_r^{(n)}) \leftarrow$ head position change (in $\mathbb{R}^3$)

$(\Delta \tilde{y}_r^{(n)}, \Delta \tilde{p}_r^{(n)}, \Delta \tilde{r}_r^{(n)}) \leftarrow$ head orientation change (in $[-180,180)^3$)

// Compute changes relative to curvature:

$(\Delta d_r^{(n)}, \Delta s_r^{(n)}) \leftarrow$ straight and strafe motion (in $\mathbb{R}^2$)

$\Delta \alpha_{\tilde{y}_r}^{(n)} \leftarrow$ arc angle (in $\mathbb{R}$)

// Set virtual camera state:

$$\begin{pmatrix} x_v^{(n)} \\ y_v^{(n)} \\ z_v^{(n)} \end{pmatrix} \leftarrow \begin{pmatrix} x_v^{(n-1)} \\ y_v^{(n-1)} \\ z_v^{(n-1)} \end{pmatrix} + \begin{pmatrix} v_{v_x}^{(n-1)} \cdot \Delta d_r^{(n)} + v_{v_z}^{(n-1)} \cdot \Delta s_r^{(n)} \\ \Delta y_r^{(n)} \\ v_{v_z}^{(n-1)} \cdot \Delta d_r^{(n)} - v_{v_x}^{(n-1)} \cdot \Delta s_r^{(n)} \end{pmatrix} \quad \textit{// position}$$

$$\begin{pmatrix} \tilde{y}_v^{(n)} \\ \tilde{p}_v^{(n)} \\ \tilde{r}_v^{(n)} \end{pmatrix} \leftarrow \begin{pmatrix} \tilde{y}_v^{(n-1)} \\ \tilde{p}_v^{(n-1)} \\ \tilde{r}_v^{(n-1)} \end{pmatrix} + \begin{pmatrix} \Delta \tilde{y}_r^{(n)} - \Delta \alpha_{\tilde{y}_r}^{(n)} \\ \Delta \tilde{p}_r^{(n)} \\ \Delta \tilde{r}_r^{(n)} \end{pmatrix} \quad \textit{// orientation}$$

end for

In the pseudo code, $\Delta d_r^{(n)} \in \mathbb{R}$ denotes the arc length of the traveled circular path along the two-dimensional ground plane in the physical workspace, and $\Delta s_r^{(n)} \in \mathbb{R}$ the strafe distance relative to the center of the circular path, with $\Delta \alpha_{\tilde{y}_r}^{(n)} \in \mathbb{R}$ the corresponding arc angle as shown in Figure 3(a). Figure 3(b) illustrates mapping of the arc length to a straight motion in the virtual environment, whereas if the user strays from the circular path in the physical workspace, this is mapped to a strafe motion in the VE. In this example, user movements are mapped relative to the user's real and virtual two-dimensional view direction, denoted as view direction along the xz-plane $(v_{r_x}^{(n)}, v_{r_z}^{(n)}) \in \mathbb{R}^2$ with $\| (v_{r_x}^{(n)}, v_{r_z}^{(n)}) \| = 1$ in the physical workspace shown in Figure 3(a), as well as $(v_{v_x}^{(n)}, v_{v_z}^{(n)}) \in \mathbb{R}^2$ with $\| (v_{v_x}^{(n)}, v_{v_z}^{(n)}) \| = 1$ in the virtual workspace shown in Figure 3(b). Other implementations may use arbitrarily placed

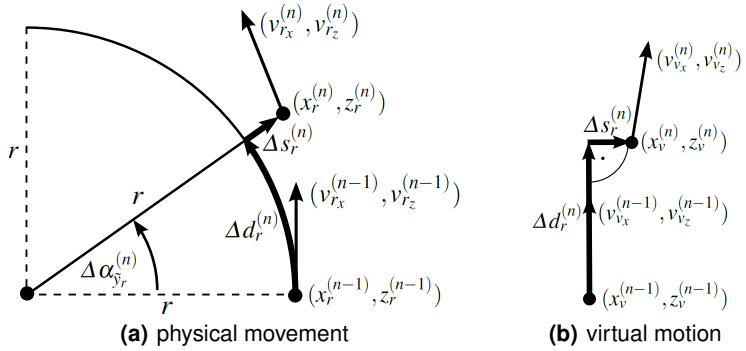


Fig. 3 Illustration of two-dimensional mappings in the xz-plane from (a) tracking coordinates to (b) virtual coordinates for an applied curvature gain $g_C = 1/r$ with radius $r \in \mathbb{R}^+$.

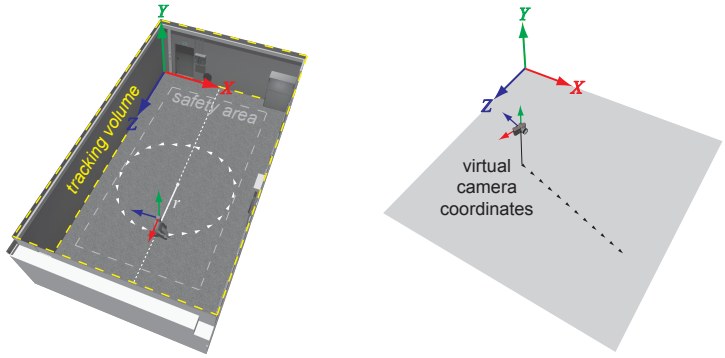
curvatures in the physical workspace, e. g., based on real and virtual path planning and transformations [20, 26, 34]. Figure 4 shows an example of a predicted straight motion in the VE being mapped to a circular path in the physical workspace.

It is important to note that curvature transformations are based on the assumption that the user will adapt to induced virtual rotations by changing the walking direction in the physical workspace. In particular, if the manipulations are overt, the user has to consciously follow the induced virtual rotations. If the user does not adapt to an induced rotation in the virtual environment, e. g., if the user is walking with eyes closed, or does not have a target in the virtual scene, it is possible that the user may stray off the path planned with curvature gains.

In general, such curvature gains can be applied not only to yaw rotations, but also to pitch and roll rotations, e. g., to simulate slopes in a virtual scene [17]. Moreover, such virtual camera rotations can be applied time-dependently, i. e., not caused by translational or rotational movements of the user in the VR laboratory, which can be described as a simple extension of the above mapping. However, anecdotal evidence suggests that virtual rotations that are not coupled to self-motions are usually easily detectable by users, and potentially distracting [23, 32].

A Basic Redirection Controller

Sophisticated implementations of unrestricted virtual walking with redirection techniques, i. e., *redirection controllers*, are usually based on information about the extents of the physical workspace, the structure of the virtual scene, and assumptions about typical user behavior. For instance, if a user is turning towards a door in a virtual building model, redirection controllers may predict the user's future virtual path to determine how to optimally scale rotations and compress distances, as well as to apply curvature gains, such that the user will be able to walk through the virtual door, without being able to detect applied manipulations [5, 20, 34]. However, in many



(a) physical movement **(b)** virtual motion

Fig. 4 Illustration of (a) path redirection with curvature gains in the physical workspace, for (b) a predicted virtual straight path.

cases such optimizations with virtual path prediction are not possible, e. g., when no information about the virtual scene is available. Some redirection controllers can be adapted to such cases, including works by the research groups of Razzaque et al. [23], Field and Vamplew [7], Peck et al. [22], Williams et al. [38, 39], Steinicke et al. [34], and Nitzsche et al. [20, 26].

A basic redirection controller can be implemented using only curvature gains. For each rendering frame $n \in \mathbb{N}$ we read the current two-dimensional head position $(x_r^{(n)}, z_r^{(n)}) \in \mathbb{R}^2$, and compute the current two-dimensional view direction $(v_{r_x}^{(n)}, v_{r_z}^{(n)}) \in \mathbb{R}^2$ with $\| (v_{r_x}^{(n)}, v_{r_z}^{(n)}) \| = 1$ in the physical workspace (see Figure 3). Based on the prediction that the user will walk in the virtual view direction [1], we try to map the user's real movements onto a circular path in the physical workspace with largest possible radius, in order to minimize applied curvature manipulations. We accomplish that by computing the strafe view direction $(v_{r_z}^{(n)}, -v_{r_x}^{(n)}) \in \mathbb{R}^2$ in the physical workspace, and solving the optimization problem of finding the point $(x_r^{(n)}, z_r^{(n)}) - r \cdot (v_{r_z}^{(n)}, -v_{r_x}^{(n)})$, with $r \in \mathbb{R}$ that is located within the physical workspace and provides the largest circle through the current user position $(x_r^{(n)}, z_r^{(n)})$, while maintaining at least the same distance to all boundaries of the interaction space, and all obstacles in the laboratory (including a small safety offset, see Figure 4). Mapping user movements onto this computed maximal circle in the physical workspace corresponds to applying a curvature gain of $g_C = 1/r$, using the formulas described above. That means, for each frame the user is redirected onto the optimal circle in the physical workspace, assuming the user will walk straight in the computed view direction. This simple approach allows practitioners to implement a reasonable mapping, which enables users to explore infinite virtual scenes by real walking, and does not require information about the virtual scene. If more information about the user's movements is available, the prediction based on the view direction can be replaced by more sophisticated strategies.

5 Conclusion

In this chapter we have described the basic math required to set up real walking user interfaces in immersive VEs. We have shown how isometric and nonisometric transformations can be used to map user movements from a physical workspace to a virtual scene. While isometric transformations provide natural feedback to physical user movements, they limit the virtual space a user can explore by real walking to the size of the tracked physical workspace. We have described how this limitation can be alleviated by combining walking in a limited interaction volume with other traveling techniques (e. g., flying). With nonisometric angular, linear, and curvature transformations we have described how the limitations of interaction space can be broken to support unlimited omnidirectional walking, although this freedom is bought with less natural feedback to physical user movements.

Practitioners interested in implementing real walking user interfaces may follow these rough guidelines:

- If the virtual interaction space is smaller or equal to the tracked physical workspace, isometric transformations should be used, since these will provide optimal self-motion feedback.
- If the virtual places of interest are rather small, but considerably spaced apart in the virtual scene, isometric mappings should be combined with traveling techniques based on additional devices or sensors.
- If the virtual scene consists of one large area of interest that could be explored by walking, then redirected walking with nonisometric mappings is recommended.

As explained above when using nonisometric mappings, the virtual view moves in a different way than the user's head in the tracked physical environment. One interesting question is how much deviation between these motions is tolerated by the user. Recently, several experiments have been reported which have identified detection thresholds for these nonisometric mappings. Interested readers may refer to works by Steinicke et al. [31, 32], Neth et al. [19] and Engel et al. [6].

In summary, movements of a user in immersive VEs have to be transferred to a virtual scene to provide the user with virtual feedback about self-motions, which can be a faithful simulation of real-world movements, or manipulated using different approaches. Since each of the approaches has different advantages and limitations, it depends on the structure of the virtual scene and the application as to which approach is best suited. In the next chapter, these approaches are discussed in more detail.

Acknowledgements The authors of this work are supported by the German Research Foundation (DFG 29160962).

References

1. A. Berthoz. *The Brain's Sense of Movement*. Harvard University Press, Cambridge, Massachusetts, 2000.
2. B. Bolte, G. Bruder, F. Steinicke, K. H. Hinrichs, and M. Lappe. Augmentation techniques for efficient exploration in head-mounted display environments. In *Proceedings of the Symposium on Virtual Reality Software and Technology (VRST)*, pages 11–18. ACM Press, 2010.
3. D. Bowman, D. Koller, and L. Hodges. Travel in immersive virtual environments: an evaluation of viewpoint motion control techniques. In *Proceedings of the Virtual Reality Annual International Symposium (VRAIS)*, pages 45–52. IEEE Press, 1997.
4. D. Bowman, E. Kruijff, J. LaViola, and I. Poupyrev. *3D User Interfaces: Theory and Practice*. Addison-Wesley, 2004.
5. G. Bruder, F. Steinicke, and K. H. Hinrichs. Arch-Explore: a natural user interface for immersive architectural walkthroughs. In *Proceedings of the Symposium on 3D User Interfaces (3DUI)*, pages 75–82. IEEE Press, 2009.
6. D. Engel, C. Curio, L. Tcheang, B. Mohler, and H. H. Bülthoff. A psychophysically calibrated controller for navigating through large environments in a limited free-walking space. In *Proceedings of the Symposium on Virtual Reality Software and Technology (VRST)*, pages 157–164. ACM Press, 2008.
7. T. Field and P. Vamplew. Generalised algorithms for redirected walking in virtual environments. In *Proceedings of the International Conference on Artificial Intelligence in Science and Technology (AISAT)*, pages 58–63, 2004.
8. J. Foley and A. Van Dam. *Fundamentals of Interactive Computer Graphics*. Addison-Wesley, 1982.
9. J. Hollerbach. *Handbook of Virtual Environments: Design, Implementation, and Applications*, chapter Locomotion interfaces, pages 239–254. Lawrence Erlbaum Associates, 2002.
10. B. Insko, M. Meehan, M. C. Whitton, and F. Brooks. Passive haptics significantly enhances virtual environments. In *Proceedings of the Annual Presence Workshop*, 2001.
11. V. Interrante, B. Ries, and L. Anderson. Seven League Boots: A new metaphor for augmented locomotion through moderately large scale immersive virtual environments. In *Proceedings of the Symposium on 3D User Interfaces (3DUI)*, pages 167–170. IEEE Press, 2007.
12. V. Interrante, B. Ries, J. Lindquist, and L. Anderson. Elucidating the factors that can facilitate veridical spatial perception in immersive virtual environments. In *Proceedings of Virtual Reality (VR)*, pages 11–18. IEEE Press, 2007.
13. J. Jerald, T. C. Peck, F. Steinicke, and M. C. Whitton. Sensitivity to scene motion for phases of head yaws. In *Proceedings of the Symposium on Applied Perception in Graphics and Visualization (APGV)*, pages 155–162. ACM Press, 2008.
14. L. Kohli, E. Burns, D. Miller, and H. Fuchs. Combining passive haptics with redirected walking. In *Proceedings of the International Conference on Augmented Telexistence*, pages 253–254. ACM Press, 2005.
15. M. Lappe, F. Bremmer, and A. van den Berg. Perception of self-motion from visual flow. *Trends in Cognitive Sciences*, 3(9):329–336, 1999.
16. J. LaViola, Jr. and M. Katzourin. An exploration of non-isomorphic 3d rotation in surround screen virtual environments. In *Proceedings of the Symposium on 3D User Interfaces (3DUI)*, pages 49–54. IEEE Press, 2007.
17. M. Marchal, A. Lecuyer, G. Cirio, L. Bonnet, and M. Emily. Walking up and down in immersive virtual worlds: Novel interactive techniques based on visual feedback. In *Proceedings of the Symposium on 3D User Interfaces (3DUI)*, pages 19–26. IEEE Press, 2010.
18. M. Mine. Virtual environments interaction technqiues. Technical Report Computer Science TR95-018, University of North Carolina at Chapel Hill, 1995.
19. C. T. Neth, J. L. Souman, D. Engel, U. Kloos, H. H. Bülthoff, and B. J. Mohler. Velocity-dependent dynamic curvature gain for redirected walking. In *Proceedings of Virtual Reality (VR)*, pages 1–8. IEEE Press, 2011.

20. N. Nitzsche, U. Hanebeck, and G. Schmidt. Extending telepresent walking by motion compression. In *Proceedings of Human Centered Robotic Systems (HCRS)*, pages 83–90, 2002.
21. R. Parent. *Computer Animation: Algorithms and Techniques*. Morgan Kaufmann, 2007.
22. T. C. Peck, H. Fuchs, and M. C. Whitton. Improved redirection with distractors: A large-scale-real-walking locomotion interface and its effect on navigation in virtual environments. In *Proceedings of Virtual Reality (VR)*, pages 35–38. IEEE Press, 2010.
23. S. Razzaque. *Redirected Walking*. PhD thesis, University of North Carolina at Chapel Hill, 2005.
24. W. Robinett and R. Holloway. Implementation of flying, scaling and grabbing in virtual worlds. In *Proceedings of the Symposium on Interactive 3D Graphics (i3D)*, pages 189–192. ACM Press, 1992.
25. W. Robinett and R. Holloway. The visual display transformation for virtual reality. Technical Report Computer Science TR94-031, University of North Carolina at Chapel Hill, 1994.
26. P. Rößler, F. Beutler, and U. D. Hanebeck. A framework for telepresent game-play in large virtual environments. In *Proceedings of the International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, pages 150–155, 2005.
27. R. A. Ruddle and S. Lessels. The benefits of using a walking interface to navigate virtual environments. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 16:1–18, 2009.
28. D. Shreiner. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Versions 3.0 and 3.1 (7th Edition)*. Addison-Wesley, 2009.
29. M. Slater. Place illusion and plausibility can lead to realistic behaviour in immersive virtual environments. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 364(1535):3549–3557, 2009.
30. J. M. Speigle and J. M. Loomis. Auditory distance perception by translating observers. In *Proceedings of the Symposium on Research Frontiers in Virtual Reality*. IEEE Press, 1993.
31. F. Steinicke, G. Bruder, J. Jerald, H. Frenz, and M. Lappe. Analyses of human sensitivity to redirected walking. In *Proceedings of the Symposium on Virtual Reality Software and Technology (VRST)*, pages 149–156. ACM Press, 2008.
32. F. Steinicke, G. Bruder, J. Jerald, H. Frenz, and M. Lappe. Estimation of detection thresholds for redirected walking techniques. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 16(1):17–27, 2010.
33. F. Steinicke, G. Bruder, B. Ries, K. H. Hinrichs, M. Lappe, and V. Interrante. Transitional environments enhance distance perception in immersive virtual reality systems. In *Proceedings of the Symposium on Applied Perception in Graphics and Visualization (APGV)*, pages 19–26. ACM Press, 2009.
34. F. Steinicke, H. Weltzel, G. Bruder, and K. H. Hinrichs. A user guidance approach for passive haptic environments. In *Short Paper and Poster Proceedings of the Eurographics Symposium on Virtual Environments (EGVE)*, pages 31–34, 2008.
35. R. M. Taylor II, T. C. Hudson, A. Seeger, H. Weber, J. Juliano, and A. T. Helser. VRPN: A device-independent, network-transparent VR peripheral system. In *Proceedings of the Symposium on Virtual Reality Software and Technology (VRST)*, pages 55–61. ACM Press, 2001.
36. M. Usoh, K. Arthur, M. C. Whitton, R. Bastos, A. Steed, M. Slater, and F. Brooks. Walking $>$ walking-in-place $>$ flying, in virtual environments. In *Proceedings of SIGGRAPH*, pages 359–364. ACM Press, 1999.
37. B. Williams, G. Narasimham, T. P. McNamara, T. H. Carr, J. J. Rieser, and B. Bodenheimer. Updating orientation in large virtual environments using scaled translational gain. In *Proceedings of the Symposium on Applied Perception in Graphics and Visualization (APGV)*, pages 21–28. ACM Press, 2006.
38. B. Williams, G. Narasimham, B. Rump, T. P. McNamara, T. H. Carr, J. Rieser, and B. Bodenheimer. Exploring large virtual environments with an HMD when physical space is limited. In *Proceedings of the Symposium on Applied Perception in Graphics and Visualization (APGV)*, pages 41–48. ACM Press, 2007.
39. X. Xie, Q. Lin, H. Wu, G. Narasimham, T. P. McNamara, J. Rieser, and B. Bodenheimer. A system for exploring large virtual environments that combines scaled translational gain

and interventions. In *Proceedings of the Symposium on Applied Perception in Graphics and Visualization (APGV)*, pages 65–72. ACM Press, 2010.
40. G. Zachmann. A language for describing behavior of and interaction with virtual worlds. In *Proceedings of the Symposium on Virtual Reality Software and Technology (VRST)*, pages 143–150. ACM Press, 1996.