DreamWalker: Substituting Real-World Walking Experiences with a Virtual Reality

Jackie (Junrui) Yang[1,2], **Christian Holz**[1], **Eyal Ofek**[1], **Andrew D. Wilson**[1]
[1]Microsoft Research, Redmond, WA, USA [2]Stanford University, Stanford, CA, USA
jackiey@stanford.edu, {cholz, eyalofek, awilson}@microsoft.com

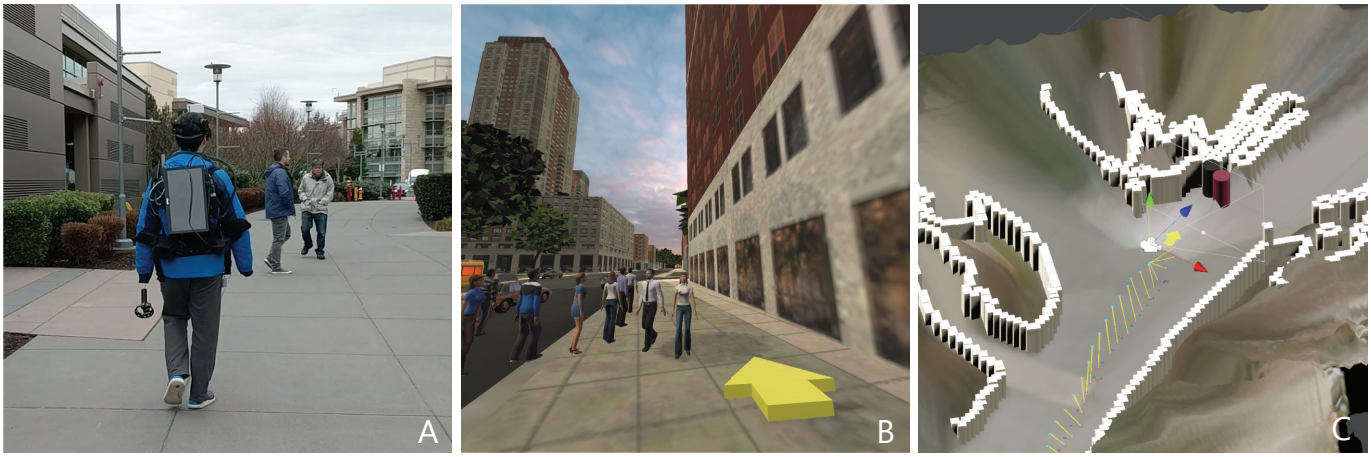

Figure 1. DreamWalker is a virtual reality system that enables users to walk in (A) an uncontrolled, previously unscanned outdoor world while (B) real-walking and staying immersed inside a large virtual reality environment. (C) In real-time, DreamWalker fuses two RGB depth sensors, Windows Mixed Reality's inside-out tracking, and GPS position frames in its positioning system to compute a walking map and guide the user through the space collision-free. This debug image results from DreamWalker's tracking, showing the detected ad-hoc and dynamic obstacles (people) as cubes and cylinders, respectively, and illustrates our three tracking sources: depth height map (rendered as a point cloud), GPS coordinates (purple), and Windows Mixed Reality tracking coordinates (cyan). Yellow lines represent DreamWalker's matches between coordinates to compensate for drift across tracking sources.

ABSTRACT

We explore a future in which people spend considerably more time in virtual reality, even during moments when they transition between locations in the real world. In this paper, we present DreamWalker, a VR system that enables such real-world walking while users explore and stay fully immersed inside large virtual environments in a headset. Provided with a real-world destination, DreamWalker finds a similar path in a pre-authored VR environment and guides the user while real-walking the virtual world. To keep the user from colliding with objects and people in the real-world, DreamWalker's tracking system fuses GPS locations, inside-out tracking, and RGBD frames to 1) continuously and accurately position the user in the real world, 2) sense walkable paths and obstacles in real time, and 3) represent paths through a dynamically changing scene in VR to redirect the user towards the chosen destination. We demonstrate DreamWalker's versatility by enabling users to walk three paths across the large Microsoft campus while enjoying pre-authored VR worlds, supplemented with a variety of obstacle avoidance and redirection techniques. In our evaluation, 8 participants walked across campus along a 15-minute route, experiencing a lively virtual Manhattan that was full of animated cars, people, and other objects.

UIST '19, October 20–23, 2019, New Orleans, LA, USA.

ACM ISBN 978-1-4503-6816-2/19/10 ...$15.00.
https://doi.org/10.1145/3332165.3347875

Author Keywords

VR; Real-walking; Redirection; Tracking; Inside-out; GPS.

INTRODUCTION

The latest wave of mobile technologies has been pushing virtual reality (VR) systems into the consumer market. Customers can now readily get their hands on affordable standalone devices on the go [10, 16], unleashed from stationary and cumbersome tracking systems and the headsets used in former decades. Current headsets track the user's motions inside-out, relying only on sensors inside the headset and thus allow the user to freely move about their surroundings.

Application developers are seizing the opportunity and are releasing a manifold of content to consumers. Starting with popular games [22], collaborative environments [11] and productivity applications were quick to follow [2]. Major software development companies, too, have announced porting their offerings to VR [41].

Therefore, we believe that future computer users will spend significantly more time in VR, where they remain immersed in entertainment, education, and work environments for long periods of time. It is entirely possible that future VR users may interrupt their VR sessions merely to accomplish real-world activities such as bathroom breaks or going to sleep.

We also believe that future VR users will still partake in real-world activities despite experiencing prolonged VR sessions. Until such a future arrives, people will at the very least retain the need to repeatedly visit various physical places, be it commuting to work, walking to the grocery store, or traveling to remote destinations.

In this paper, we explore to what extent the sometimes mundane experience of such real-world activities can be substituted by entertaining VR experiences. We choose walking task that we still expect future users to perform—albeit immersed in VR while actually walking in the real world. We present DreamWalker, a VR system that guides users through the large outdoor real-world to target destinations while enabling them to explore a virtual world through real-walking.

DreamWalker: Real-Walking a Virtual Reality

Figure 1 shows DreamWalker in operation. A company employee needs to walk to a different building across campus, a task he carries out multiple times a week. To spice up his routine walk, he exits the building, puts on DreamWalker's mobile VR system (Figure 1a), selects a target destination in the system on a map, and enters the virtual world. The system places him in Manhattan's downtown area in VR (b), full of tall buildings, traffic, people standing, walking, and running into various directions—a seemingly wide-open space, infinite, and ready for exploration. The user starts walking and is guided by an arrow. Cars move and people run, but the user finds ways to follow the arrow's navigation, thereby following and crossing virtual streets, squares, and parks—all while using real-walking as the sole locomotion technique inside VR.

Behind the (VR) scenes, DreamWalker ensures that the user safely reaches their real-world destination through visual (re)direction at normal walking speeds, detecting obstacles along the way and presenting corresponding VR objects to prevent collisions. DreamWalker fuses the data streams from two RGB depth cameras, optical inside-out tracking, and GPS position data in real-time to detect the user's surrounding 3D obstacles and other people. DreamWalker tracks the user's 3D environment (Figure 1c) and continuously updates a map of walkable areas that is registered with real-world coordinates and landmarks, such as buildings and walkable paths. DreamWalker then represents real-world obstacles through static and animated virtual elements and predicts the path forward based on walkable spaces to navigate the user through VR.

In this paper, we make four specific contributions:

1. a real-time tracking and positioning system that accurately maintains the user's real-world location while walking in previously unseen and unscanned physical environments, achieved by fusing Windows Mixed Reality's inside-out tracking, filtered low-accuracy GPS positions, and RGB depth frames,

2. an obstacle detection system that extracts surrounding static and dynamic physical objects from the moving head-mounted RGB depth sensor on-the-fly while walking, achieved through spatially registering, filtering, and classifying depth environments *without* the need for pre-scanning the environment,

3. three obstacle avoidance and redirection rendering techniques that guide users through two large pre-authored virtual environments with different affordances (e.g., available spaces, turns, path widths), populated with static and dynamic objects that map to real-world obstacles, and

4. a feasibility evaluation with 8 participants who real-walked a virtual and animated Manhattan scene and rated DreamWalker's experience as immersive and enjoyable.

Taken together, DreamWalker's novelty is in the experience it enables through its real-time outdoor tracking system, performing redirected walking beyond controlled rooms [38] or empty flat soccer fields [21]. DreamWalker works in unseen large-scale, uncontrolled and public areas on contiguous paths in the real-world that are void of moving vehicles.

RELATED WORK

DreamWalker is related to work in three categories: Wide-area Augmented Reality/Virtual Reality, obstacle detection and environment construction, as well as redirected walking.

Wide-area Augmented Reality and Virtual Reality

A number of research and commercial projects have implemented Augmented Reality (AR) systems in large uncontrolled environments. As users look at the world around them through AR displays, they can naturally navigate and avoid obstacles. However, the view of the real world also limits the scope of the applications that are supported.

For example, Human Pacman [8] implemented a mixed reality game that blends virtual objects into real environments. The paper reports that walking inside the real world gave users a high level of sensory gratification in the game. Pokémon Go is a GPS-based mobile AR game that entices the user to interact with various location-based game elements while walking. Previous research also shows the risks of injury even when the physical environment is visible [32].

VR isolates the user from the surrounding world and allows for a complete visual overhaul of the surroundings. The resulting lack of physical feedback has confined most implementations to a controlled and mostly empty indoor environment [44]. Researchers have developed collision avoidance techniques between multiple users in the same space, such as by changing players' appearances [27] or by redirecting users during game play to prevent overlapping physical spaces [19]. RealityCheck [12] strikes a balance between both worlds and

selectively shows real-world objects such as furniture as part of a situated VR experience inside a room.

Procedural generation

Other research has investigated techniques for users to avoid real-world collisions while in VR. A popular technique is procedurally generating virtual geometry based on the known physical environment. Oasis leverages a pre-scanned representation of a space to generate a virtual environment inside [33], creating a 1:1 mapping with obstacles in the real world. Keller et al. integrate the user's surrounding game room into the virtual reality experience and enable real-walking within [15]. VR arcades, such as the Void [43], allow users to walk in a wide spaces while in VR. These arcades thereby provide passive props for the use of passive feedback during dedicated gaming experiences. These examples use *controlled* indoor environments where all obstacles were either cleared or registered with the system before use.

Perhaps closest to DreamWalker is our previous system VRoamer, which procedurally generates corridors and places pre-authored virtual rooms as the user real-walks open building spaces [7]. Although VRoamer also operates on previously unseen environments, it requires indoor environment with planar floors and large-enough obstacles. DreamWalker's approach replaces VRoamer's tracking pipeline by dynamically detecting obstacles lower than ground including slopes, obstacles with small heights such as street curbs, non-planar vertical obstacles, and tracks people standing and walking around the user. DreamWalker additionally fuses it with GPS frames and correcting for drift in each of the three tracking sources.

With DreamWalker, we take real-walking to the next level, enabling users to walk inside uncontrolled outdoor environments while simultaneously retrofitting a pre-authored virtual environment to the physical world. This makes a *given* virtual environment walkable in contrast to making a walkable physical environment virtual.

Redirected Walking

DreamWalker redirects the user using content in VR to prevent collisions with real-world obstacles. Redirected walking has been actively researched, starting with demonstrations to induce unintentional direction changes in users' walks by imperceptibly rotating the virtual scene [24]. Follow-up work has evaluated the limits of tolerable redirection, such as investigations into allowable ranges of rotation and translational gains [36] and experiments on acoustic-based redirection [31].

Alternatively, researchers have produced virtual distractions to subtly redirect the user. Chen et al. elicited head motions from the user through such distractors [5], while VMotion introduced novel visibility control techniques that temporarily switch the user's view and thus allow for redirections that are seamlessly integrated into virtual experiences [34, 35]. Langbehn et al. leveraged users' blink events, during which they changed the environment [17], while Sun et al. exploited eye gaze saccades to hide rotational gains [38]. Other research has investigated how to apply redirected walking in real life (e.g., in irregular rooms [4, 14, 13]). Other related efforts have investigated techniques to position the virtual world to more easily fit it into a small tracking space [42, 37].

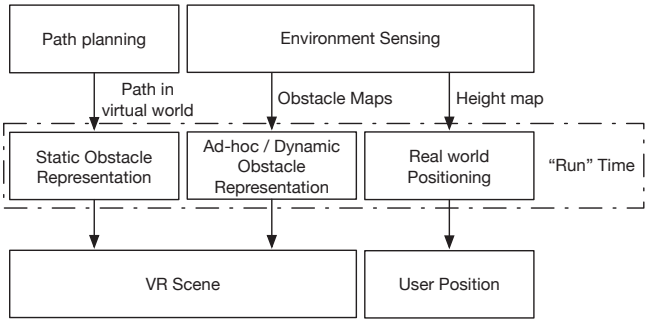

Figure 2. Overview of DreamWalker's system.

Building on related techniques, but deviating from previous implementations, DreamWalker utilizes redirected walking to safely keep the user on differing virtual and real outdoor paths at the same time.

Obstacle detection and environment reconstruction

For environment reconstruction, related work has fused RGB depth images into 3D spaces (e.g., using SLAM [3, 23]). Oasis uses this to allow users to scan a physical environment and generates a VR environment inside [33]. Also related is preliminary environment mapping for redirected walking, which constructs a map of obstacles in the user's tracking space for later redirection [14]. Regarding obstacles, both previous projects consider anything above ground level as an obstruction. In DreamWalker, this assumption does not apply due to changing altitudes and inclinations in outdoor environments.

Research on autonomous vehicles has explored sensing modalities to detect obstacles around cars, such as by recognizing them using LIDAR depth data [9], RGB depth images [47], or from RGB input alone[39]. Car-based sensors benefit from statically mounted cameras with a steady perspective. DreamWalker, however, uses head-mounted sensors that move as the user looks around and shake during walking.

DREAMWALKER'S SYSTEM DESIGN

DreamWalker is a tracking and navigation system that guides users through VR environments using real-walking while redirecting them to reach a real-world destination. In order to achieve this, DreamWalker implements three main components in its system as outlined in Figure 2: Path planning, Real-time environment detection, and a "Run" time.

The input for path planning is a given real-world path and the virtual world that the user intends to real-walk in. Path planning then finds paths in the virtual world that match the real-world path as closely as possible, choosing the path with the lowest error. DreamWalker "resolves" the remaining differences during runtime through redirection and slight variation of the user's virtual walking speed. Path planning identifies the locations where to apply such corrections and populates the virtual world with static objects that will prevent collisions with real-world obstacles that are known a priori from mapping data (e.g., buildings, streets, etc.).

Our real-time environment detection and runtime systems operate while walking to guide the user along the path. DreamWalker integrates three signal sources: 1) For 6-DOF inside-out tracking, DreamWalker builds on the Windows Mixed Reality system, which provides a relative position trace that begins drifting noticeably after ˜100 feet. 2) For absolute positioning, DreamWalker uses a dual band GPS sensor, which provides a coarse real-world location at low update rates. 3) DreamWalker uses two RGB depth cameras to obtain real-time information about the user's immediate surroundings and obstacles. DreamWalker's positioning system fuses and reconciles the data from all sources in real-time to estimate the user's precise real-world location and redirects them in the virtual world to stay on the planned path. DreamWalker dynamically represents detected obstacles in VR to create an experience that matches the dynamic nature of the real world.

Types of obstacles and their representation

DreamWalker's goal is preventing the user from colliding with obstacles. The tracking system must therefore detect obstacles reliably, in real time, and represent them to the user by generating virtual objects. We categorize obstacles based on the phase of operation in which they are detected:

1. *Static obstacles* are obstacles represented in map data whose location is known at path planning time, such as building facades, columns, walls, staircases, fences, and other non-walkable surfaces such as streets, intersections, patches of grass and so on. To prevent users from walking into such obstacles, DreamWalker blocks access to them by placing stationary virtual objects such as facades, cars, tables, hot dog stands, trashcans, barriers etc.

2. *Ad-hoc obstacles* are physical obstacles that are not known at path planning time, but interfere with walkable areas and must thus be discovered on the fly. Ad-hoc obstacles are stationary but often are only recognizable when they are well in the field of view of the user, such as parked cars, trashcans, road blocks, pillars or even smaller obstacles such as potholes or drinking cans. While these ad-hoc obstacles do not move, their potentially only late discovery requires DreamWalker to represent them using virtual elements that dynamically appear, such as virtual characters or traffic cones.

3. *Dynamic obstacles* are not known during path planning time either and must also be detected on the fly. Unlike ad-hoc obstacles, dynamic obstacles might move around or towards the user, for example other pedestrians, dogs, bikes, and cars. Similar to ad-hoc obstacles, DreamWalker represents dynamic obstacles as virtual characters that move within the virtual world to match the motion of their real-world counterparts.

Implementation and system components

DreamWalker runs on the HP OMEN Gaming Backpack with a GTX 1080 (Figure 3). The Windows Mixed Reality system provides inside-out tracking on a Samsung Odyssey VR headset, updating sensed 6D locations at 90 Hz. Empirically, we measured 1 m of drift over a course of just 30 m through the inside-out tracking alone. Two Intel RealSense 425 cameras provide RGB depth images, slightly angled and rotated 90 degrees to achieve a large field of view ($86^\circ \times 98^\circ$). We

Figure 3. DreamWalker's mobile virtual reality system, including sensors, computing platform, and display device.

built a custom adapter[1] for the backpack computer that converts Thunderbolt 3 to four USB 3 ports and thus supports the bandwidth required to stream both RGB depth cameras at a resolution of 640×480 (depth) and 640×480 (RGB) at 30 Hz. Finally, GPS data comes from the sensor inside a Xiaomi Mi 8 phone, which features dual band (L1/L5) GPS and updates the real-world location with a theoretical accuracy of 0.3 m at 1 Hz (though, empirically, the actual accuracy was ˜5 meters).

We implemented DreamWalker in Unity 2018.2. While the core system runs at approximately 45 Hz, Windows Mixed Reality implements asynchronous time warp to interpolate frames and produces visuals at 90 Hz.

To experiment on a large area that is representative of real environments, we tested DreamWalker on Microsoft's large Redmond, WA campus. Using Openstreetmap data, we verified the accuracy of the annotated streets, paths, areas, and facades using satellite imagery and corrected them in Openstreetmap when necessary. We also verified the correctness of path labels such as staircases vs. ramps and finally retrieved the contiguous network of traffic-free paths shown in Figure 4. The campus is a suitable testing ground, with paths and roads that contain plenty of ad-hoc obstacles (e.g., tables, chairs, benches, lanterns, pillars, trees) and standing and walking people. Figure 4 highlights the three campus paths we show in our video figure (taking 8–15 minutes to walk), including the path participants walked during the evaluation (dotted, taking on average 15 minutes).

We tested DreamWalker with two pre-authored virtual environments that we downloaded from an online store. The first virtual world is downtown New York City [2], a large, walkable area with plenty of visual features, such as diverse and tall buildings, moving cars and buses, people on the sidewalks as well as stands, posters, and other features. The second world is Unity's Viking Village[3], a detailed ancient village area with narrow paths and turns, wooden buildings, beaches, and many

[1]Custom adapter to mount depth cameras to the VR headset: https://aka.ms/dreamwalker_models
[2]Manhattan Lower Part01 Low Poly on Turbosquid: https://www.turbosquid.com/3d-models/m/1074847
[3]Viking village on Unity Asset store: https://assetstore.unity.com/packages/essentials/tutorial-projects/viking-village-29140

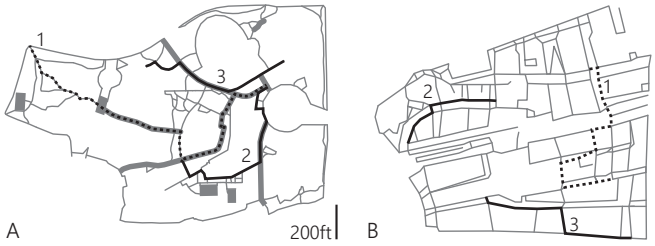

Figure 4. Campus map of contiguous traffic-free paths with the three paths users walked (dotted line walked during the study). DreamWalker's path planning system searches paths resembling (A) the real-world path provided from a map routing system in (B) the graph of virtual paths. During the matching phase, we consider acceptable redirection in motion speed-up, slow-down, and rotation redirection.

virtual obstacles, including fences, barrels, and stacks of material. Note that although DreamWalker can map an outdoor walk with an indoor world, we chose to map to virtual outdoor models as it is hard to remained entertained while walking only indoor VR scenes for the duration of ten or more minutes. It may also affect walking speeds.

PART 1: PATH PLANNING AND OBSTACLE POPULATION

DreamWalker's first prompts the user for a destination in the real world and then obtains a route from a routing service (e.g., Google Maps) as a series of GPS coordinates as shown in Figure 4a. DreamWalker then fits this series to the walkable path network of a virtual environment (Figure 4b). We explain DreamWalker's path planning at the example of the Battery Park area of Manhattan, which is composed of streets, parks, squares—all plausible walking paths suitable for our use-case.

Finding a matching virtual path

Given the real-world GPS series, DreamWalker finds a resembling path in the virtual world. We denote the real-world path as the series $P = \{P_i | i \in 1..n\}$ of $n-1$ linear segments. Although the real-world path and the virtual world path may have curvature in some cases, DreamWalker's path planning algorithm automatically breaks them into line segments. A curve with strong curvature is broken into multiple line segments, while a low curvature path is mapped to a single segment. The factor c_{diff} encourages a mapping of paths with similar curvatures.

The virtual world is represented as a graph $G = (V, E)$, where each edge $e \in E$ is linear and may have different widths (e.g., because of varying sidewalks or paths through parks). The quality of matching routes results from three factors:

1) *Length scale:* The length of the virtual walk should not vary from the actual distance by more than 33%. We assign a cost of 0 for matching a path segment if it maps to a virtual segment within this scale; otherwise, we assign a cost of the change in length beyond this amount, normalized by the length of the whole segment. The total cost of the path c_{len} results from the sum of the squares of all segment costs.

2) *Difference:* Paths may have variations in shape (e.g., to avoid obstacles), but local differences between the two paths must not extend into a street. We define the cost per segment as the maximum distance between the linear segment mapped to the virtual world and the matching virtual path. The total cost of these differences c_{diff} is the sum of the squares of each segment match cost.

3) *Redirection:* While the global shape of paths may differ, changes in curvature along the path should not exceed 45°. Beyond this angle, redirection becomes obvious to the user and may instead be substituted by scripted distractions [37]. DreamWalker tries to minimize the number of redirections c_{dir} (similar to the limits reported in prior work [28]).

The quality rating for a match of routes results from the convolution of these three factors, so that match cost is:

$$c_{match} = \sqrt{w_{len} \cdot c_{len}^2 + w_{diff} \cdot c_{diff}^2 + w_{dir} \cdot c_{dir}^2}$$

We use $w_{len}, w_{diff}, w_{dir}$ to balance between the constraints.

To find and fit a virtual-world path Q to the given real-world path P, DreamWalker implements a greedy search for candidate virtual paths. At each iteration, we start by generating a plausible mapping of each of the path vertices P_i to a matching vertex V_i in the graph, where the distance between $[V_i, V_{i+1}]$ is within the length scale of the corresponding real-world path segment. Figure 4 shows one example solution generated by DreamWalker's path planning algorithm, starting with a real-world path provided from a map routing system P (A), the virtual graph G (B), and ending with the closest match Q (highlighted path in (B)) that produces the lowest cost while considering redirection. For our examples, a path P including each of its segments mapped to a series of segments in G after 5000 iterations (˜5 minutes on a Core i5), generating possible matches and choosing the mapping with minimal cost.

Our path planning is different from others, as it fits the similarity of two paths while computing acceptable redirection. Since the search space is vast, DreamWalker implements random walk and greedy search. The algorithm currently requires precomputation and reuse during operation, but we believe that this can be sped up through simulated annealing in the future, reusing previously found sub paths, and iterating only on the rest of the path vertices. The duration of execution also depends on the virtual environment; VR worlds with wider open spaces allow for much easier and faster routing due to the lower number of constraints on finding virtual paths.

Note that since we have constraints on the scale, difference, and redirection, path planning may not always produce an acceptable match of paths in the virtual world given a path in the real world and a virtual scene. We believe this can be mitigated in the future when a large VR repository can be made for DreamWalker, and the user can select from scenes that have a good match for the path in the real world.

Static obstacle population

After obtaining a suitable mapping of the real-world route to a virtual route, we populate the virtual environment with objects along the walking path. To achieve a naturalistic blend between the obstacles and the virtual environment, we have authored one model with obstacles that fill up all paths for every virtual environment. We place virtual objects that can be used to represent static obstacles on every path as described above, such as roadblocks, trucks, poster stands, and so on.

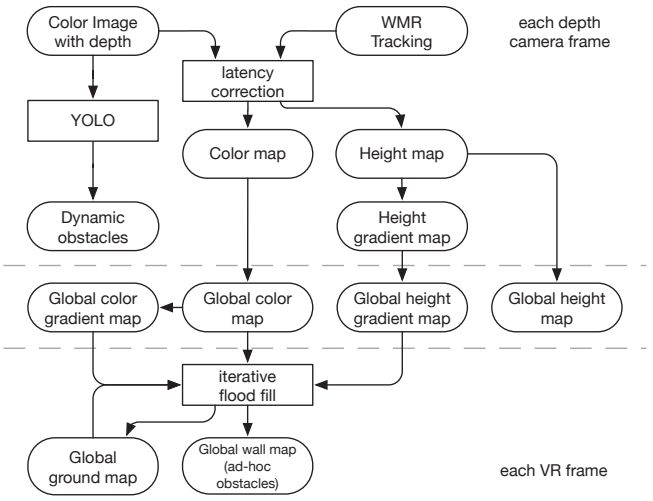

Figure 5. DreamWalker's environment sensing and processing pipeline.

We also add dynamic elements to the scene to make the virtual environment more vibrant, such as walking people and driving cars and buses. After the user selects a path and a virtual world, DreamWalker retrieves the shape of the virtual route, and removes any obstacles that collide with the user's route as well as any traffic that crosses it.

This step generalizes beyond the example of Manhattan and applies to any virtual world with walkable paths, such as Viking Village.

PART 2: REAL-TIME ENVIRONMENT DETECTION

DreamWalker detects obstacles and thus potential trip hazards based on head-worn, forward-facing sensors. In contrast to related work in robotics, we have no control over the motion of these sensors, as they are part of a mobile setup with limited computational power and sensing range.

DreamWalker's real-time environment detection component produces two signals for our runtime system (Figure 2: a *height map* and a *path map* (i.e., walkable paths)) that feed into our positioning and redirection system as well as a global *wall map* (containing the locations of obstacles) and dynamic obstacles that serve as input into our VR obstacle rendering engine (shown in Algorithm 1). Our environment detection continuously integrates the two depth and the two RGB images from the Intel Realsense cameras and aligns them with the transformations provided by the Windows Mixed Reality (WMR) optical inside-out tracking that DreamWalker corrects for drift.

Fusing RGB depth frames into the tracking space

By processing the RGB depth stream, DreamWalker detects two types of non-walkable areas, both of which qualify as ad-hoc obstacles (Figure 5). First, some non-walkable areas preclude walking simply by physically obstructing the path and thus produce a distinct height difference compared to the surrounding ground, such as benches, pillars, or walls. Second, other non-walkable areas have a similar height as the surrounding ground, but differ in texture and color, such as grass patches, dirt roads, or sidewalk strips.

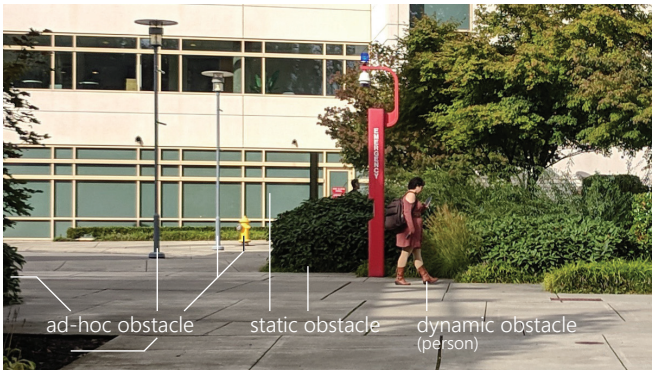

Figure 6. The three types of obstacles that DreamWalker's environment recognition system detects and tracks during the user's walk.

To detect and track non-walkable areas, DreamWalker constructs a height map from the depth image during operation. We start this process by projecting each RGB depth frame from the Intel cameras into the WMR tracking space in the form of point clouds. We then project these points onto a ground plane, which results in the *height map* and the *color map*. We downsample these projections to height and color images in which one pixel corresponds to 10 cm $\times$ 10 cm and 5 cm $\times$ 5 cm, respectively, for the purpose of memory reduction and speed of computation. DreamWalker implements custom shaders for these projections to produce color maps and two depth maps. To obtain a nearly-complete color map, our system renders triangles constructed from adjacent pixels. For the height maps, we extract the highest and the lowest point in the point cloud within the region of this pixel and produce a projection of highest and lowest points, respectively. We then define the gradient of a pixel as the maximum difference between the highest point and the lowest point for each pixel as well as adjacent pixels, resulting in the *height gradient map*. DreamWalker individually performs this process for each RGB depth frame, each of which is inherently noisy and thus not suitable to directly generate obstacles. Having computed a color map, height map, and height gradient map, DreamWalker aligns consecutive maps using the WMR tracking and temporal filtering. We finally derive the global color gradient map from the global color map.

Obstacle extraction

The final step of DreamWalker's environment detection system is generating the *path map* (i.e., the map of walkable paths) and the *wall map*, which encodes the locations where DreamWalker's runtime system must position virtual obstacles. The accuracy of the wall map is crucial for creating a compelling and collision-preventing virtual environment for the user.

Therefore, DreamWalker extracts real-world obstacles from the global *height gradient map*, not the global *height map* itself. The global *height map* is prone to misalignments due to small inaccuracies in pitch reported by the WMR headset, which generate large height gaps in between the aligned local *height map* and existing global *height map*. For example, 1° of misalignment may result in a height difference of 4 inches just 15 feet away—a sizable obstacle that would result in a false-positive VR representation. The global *height gradient map* is less prone to this error because when computing this

map, we only compare heights within a local height map and only use the *difference* in fusing them.

Extracting virtual representations based on the global *height gradient map* and thus extracting them from the edges of real-world obstacles is sufficient to prevent collisions. This is because large areas of non-walkable obstacles require a representation in VR solely in locations closest to the user—those that the user would risk walking into.

DreamWalker generates the *path map* from the global *height gradient map* through a flood fill starting at the user's location and only filling in areas where the color and height gradients are below a threshold. Because flood fill is unsuitable for parallelization, we leverage the fact that the global gradient map remains relatively consistent over time, because all ad-hoc obstacles are stationary in the real world. Thus, we merely perform an iterative flood fill, which usually advances the previous flood fill for 4 pixels per RGB depth update. Whenever our flood fill now cannot fill into an area, we mark that pixel as an obstacle in the *wall map*.

Next, we classify contiguous regions of colored areas in this synthetic RGB image based on predefined RGB distributions, such as green or red, as these areas tend to demarcate sidewalks from other parts of the pavement. For those areas that produce a sufficient score during the matching, we additionally mark them as non-walkable in the *path map*.

To detect *dynamic obstacles*, we process both RGB images for people and other moving objects. DreamWalker integrates YOLO's real-time object classification [25] in a separate thread, which reports 2D boxes of detected objects at 15 Hz. We project them into the environment using the closest distance in the depth images within a 10×10 pixel area of a YOLO box, our environment detection adds their locations to the *wall map* and reports them to the runtime system. This ensures that moving pedestrians get represented in VR and can thus be avoided by the user.

We refined DreamWalker's tracking and detection system extensively for over a month during daily walks. These walks led across various public and uncontrolled settings, including sidewalks full of pedestrians, campus areas, squares, underpasses, each time in unseen areas to verify proper operation. While developing our obstacle detection, we experimented with state-of-the-art computer vision algorithms for self-driving cars, which unfortunately proved unsuitable for our purposes. Since LIDAR is impractical in DreamWalker due to weight and necessary computation, we tested RGB-based detectors, including Dilated Residual Networks [45], MultiNet/KittiSeg [39], and FCN/VGG [18]. They were unreliable at accurately segmenting walkable areas (i.e., paths on campus), most likely because they were trained on road views for cars, trucks, bikes, but not the branches, curbs, poles DreamWalker must detect.

PART 3: DREAMWALKER'S RUNTIME SYSTEM

DreamWalker's runtime implements three subsystems: 1) The positioning subsystem determines the user's accurate position in the real world. 2) The redirection subsystem compensates for drift between virtual world and real-world tracking by applying additional redirection according to the planned path (Part 1). 3) Obstacle representation produces virtual objects to guide the user and prevent collisions.

Positioning system

The goal of the positioning system (pseudo code shown in Algorithm 2) is to produce a rigid transform T that maps locations and orientations from the inside-out tracking space T_{origin} to world coordinates T_{gps}. Most important is the horizontal correction of the user's position (i.e., Unity coordinates X-Z axis or latitude-longitude for GPS).

The positioning system receives two types of input: the user's position $p_{user/gps}$,in real-world GPS coordinates T_{gps} and relative WMR inside-out locations. In contrast to GPS' coarse but globally grounded measurements, WMR tracking measures differential changes of positions and orientations. Its current orientation $T_{user/origin}$ is an accumulation of transformations from an arbitrary initial coordinate system at the beginning of the path, T_{origin}, to the current coordinate system around the user T_{user}. As time progress, accumulated error may generate a drift of T_{user} relative to the global coordinate system.

A naïve solution is to fit a rigid transformation between a history of recent GPS locations and corresponding user locations. In practice, the GPS positions $p_{user/gps}$ DreamWalker collects have systematic offsets from the user's true position due to multi-path effects, atmospheric effects, etc. For example, when walking along buildings in the real world, the received GPS coordinates may have a constant offset either towards the street or a location inside the building. The resulting transformation would create a path outside of the true walkable area and thus would increase risk of collisions.

To derive an accurate transformation during runtime, we leverage the assumption that the user is always walking on the planned path. This allows us to transform the WMR inside-out tracking position history $[p_{user/origin}]$ using the estimated transform of the naïve method above to derive $[p'_{user/gps}]$. We then accurately align the result with the part of the planned path $[p_{path}]$ that the user has already walked ($p'_{user/gps}$) using an iterative closest point matching algorithm. This alignment produces a correction matrix $T_{history}$, mapping the estimated GPS transformation to a more accurate GPS transformation. We denote this target rigid transform as $T_2 = T_1 * T_{history}$.

While T_2 now accurately maps locations *across* the user's path, inaccuracies may still occur *along* the path, and thus in a direction that most likely contains physical obstacles next to the user, such as buildings. Such behavior could occur after walking a long and straight path segment, for example, just before turning right at an intersection in the real world. Since $T_{history}$ transforms the user's location history to the already-walked path, it only corrects inaccuracies perpendicular to the user's walking direction. When reaching a turn on the planned path, for example at an intersection, the system cannot accurately determine where and when the user should make this turn using only this transformation, because locations parallel to the current walking direction are inaccurate. We thus complement our previous transformations with another correction matrix T_3, which we generate based on the global *height map* that we assembled in the previous section. We

first project the user's future path onto this *height map* and search for the best fit in the possible transformation space of $[T_2^{-1} * T_{future}]$ across offsets in the two horizontal dimensions as well as rotation offsets. Through a custom compute shader, we obtain the total height changes along each one of these possible paths. Since the path originated from DreamWalker's path planning, we know that it traverses the world across a flat ground, meaning a low total change in heights during this search. We thus pick the transform $T_{future} \in [T_{future}]$ with the lowest total height change, and apply this correction to the original transform T_2. As a result of this correction, we obtain an accurate transformation that compensates for inaccuracies and jumps in *both* dimensions of the GPS updates: $T_3 = T_{future}^{-1} * T_2$. In the remainder of this section, we use T_3 as the transform that accurately maps coordinates from the WMP inside-out tracking space T_{origin} to the GPS space T_{gps}.

Redirection system

During runtime, DreamWalker tracks the user's position in two reference systems: where the user is physically located (as reported by the positioning system described above) and the location in the virtual world. While the user traverses both reference systems through walking, DreamWalker ensures that the visuals presented to the user result in their accurately walking the real world without collisions. In the case of a discrepancy between both reference systems, DreamWalker applies a gradual correction of the VR rendering to *implicitly* redirect the user's walk without their noticing (i.e., similar to related redirected walking systems [24]). Below, we explain how DreamWalker smoothly adapts its space transformations to smoothly guide the user (back) towards the planned path (pseudo code shown in Algorithm 3).

Redirection method

DreamWalker implements redirected walking by limiting transformation changes according to the user's actual viewport motion. The transformation matrix T derived above may include large differences frame to frame and may even change when the user performs no motion, for example because of inaccurate GPS updates. Resulting changes in the visual experience may thus be confusing or discomforting to the user.

In each frame f_n, we record the current transform $T_{current}[n]$ as well as the user's transform $T_{user/origin}[n]$ from the WMR inside-out tracking origin. In each following frame f_{n+1}, we obtain the ideal transform $T[n+1]$ from our positioning system as well as the user's new WMR transform from the origin $T_{user/origin}[n+1]$.

We determine how much the user moves frame-to-frame in the GPS space $M_{expected}[n+1] = T_{user/origin}[n+1] * T_{current}[n] - T_{user/origin}[n] * T_{current}[n]$. The amount of this actual motion corresponds to the visual changes the system must produce to satisfy the user's expectations. If DreamWalker now directly applied the transform from the positioning system, the user's transform would amount to $T_{user/origin}[n+1] * T_{n+1}$ and thus a (visual) movement of $M_{target}[n+1] = T_{user/origin}[n+1] * T_{n+1} - T_{user/origin}[n] * T_{current}[n]$. Assuming that the transform $T_{current}[n+1]$ results from this frame, the movement the user will experience in response to this frame amounts to $M_{actual}[n+1] = T_{user/origin}[n+1] * T_{current}[n+1] - T_{user/origin}[n] * T_{current}[n]$.

Having computed the movements, the challenge is to produce a pleasant motion in VR. The actual movement $M_{actual}[n+1]$ must be within the human detection threshold of the expected movement $M_{expected}[n+1]$ while remaining close to the target movement $M_{target}[n+1]$. Thus, we generate the range of movements that are acceptable around the expected movement $M_{expected}[n+1]$ and scale the target movement $M_{target}[n+1]$ back to be within that threshold, resulting in the actual movement $M_{actual}[n+1]$ following this frame. With this actual movement, we compute the transform of this frame by reversing the movement equation: $T_{current}[n+1] = (T_{user/origin}[n+1])^{-1} * (T_{user/origin}[n] * T_{current}[n] + M_{target}[n+1])$. DreamWalker generates the acceptable movement in line with the redirection limits set in path planning.

Naïvely relying purely on estimated position may yield inconsistent transformations frame to frame, thus discomforting the user. Therefore, the implementation detailed above allows DreamWalker to provide a smooth experience to the user while gradually redirecting them towards the more accurate position.

Mapping the real-world position into the virtual world

Using the corresponding control points in the real world and the virtual world that we place during path planning as described in Part 1, we need to match all intermediate locations during runtime for smooth redirection. Our goal is to generate a transform for every point in the real world, such that the closest real-world control point maps to a location in the virtual world with ideally no offset to the corresponding virtual control point. This entails that adjacent points in the real world retain their spatial neighborhood relationship in the virtual world. This transform also needs to produce continuous rotations offset across the campus map, as the user may experience sudden change of position or rotation when walking otherwise. DreamWalker's implementation of this feature is based on feature-based image metamorphosis [1], transforming one set of position to another set using "line pairs" and linear interpolation. We extend this approach to use the more granular pairs of control points by connecting adjacent control points to line segments and then matching series of them. We also incorporate the rotation transforms mentioned above into the matching of position pairs, treating rotation as an additional dimension and applying the same interpolation.

Obstacle representation

Having computed the user's position on the map, the redirected viewport, and having detected obstacles around the user as described in Part 2, DreamWalker finally fuses all this information to curate the user's VR experience. Since our system is managing a virtual world that may have been populated with animated objects and characters, additional virtual objects need to be introduced now to represent detected obstacles.

Static obstacles: As discussed in Part 1, the path planning system populates the virtual environment with static obstacles based on map topography if spaces are too open in the virtual environment (e.g., in Manhattan). This step is unnecessary in confined virtual areas (e.g., the narrow paths in the Viking

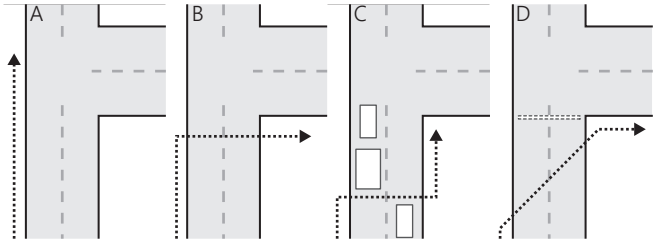

Figure 7. Plausibly guiding users through virtual intersections: a) optimal case, b) regular street crossing that can be facilitated through stopping cars, c) mid-street crossing that requires traffic blocks, such as firetrucks, d) diagonal crossing that requires shutting down the street.

Village). Routing a path through the virtual environment that resembles the physical route automatically allows us to use the facades and street object as static obstacles.

In some cases, the physical route cannot be perfectly routed through the virtual environment while staying within the constraints of tolerable redirection. Instead, the path planning system may determine a virtual route that diagonally crosses a street—an action that is uncommon in the real world. Figure 7 shows how DreamWalker handles these cases. In the regular case of crossing at an intersection (b), DreamWalker simply stops the traffic to yield to pedestrians, optionally supplemented by traffic lights. In the case where streets have to be crossed midway (c), DreamWalker finds a plausible solution of stopping traffic in the middle of the street, such as by placing a firetruck or police block. If the path planning system cannot produce a better path than a diagonal crossing (d), DreamWalker shuts down the entire road by populating the ends with road blocks, thus preventing virtual traffic and allowing the user to plausibly "safely" cross the VR street.

Ad-hoc and dynamic obstacles: DreamWalker implements three different modes of representing discovered obstacles as shown in our video figure.

Yellow pellets inspired by Human Pacman [8] indicate the path for the user to walk. DreamWalker generates them by thinning the forward-filled paths in the *height gradient map*, eliminating all paths except for the one closest to the real-world path Q generated during path planning. Dynamic obstacles are represented by moving characters, typically causing the user to pause until they have passed. Users stay collision-free as long as they closely follows the pellet-indicated path.

Traffic cones or rocks border the walkable path ahead. Similar to the pellets, DreamWalker finds these cone locations by picking the path ahead from the *height gradient map* and, as part of the flood fill, generating the outside delimiters minus a safety margin. DreamWalker then places cones or rocks in the virtual world and represents dynamic obstacles through animated virtual characters. This technique prevents collisions as long as users stay within the demarcated cone area.

Animated virtual characters ("humanoids") move into the location of detected obstacles and guide the user towards the destination. This visualization has the potential to appear most "natural" and fit the virtual narrative best, it is also the most challenging to get right [40]. Humanoids solve the need for representing uncertain ad-hoc and dynamic obstacles while

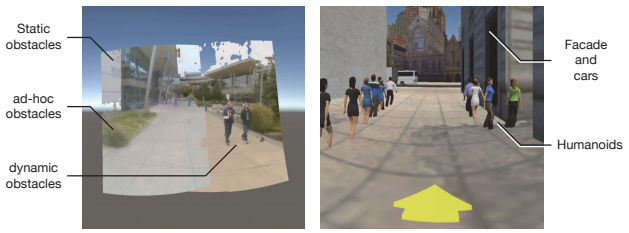

Figure 8. DreamWalker represents static obstacles through stationary virtual objects, such as buildings or cars. Ad-hoc and dynamic obstacles are represented through animated humanoid characters that walk towards the location of obstacles.

giving the virtual environment a lively feeling. We briefly describe DreamWalker's animation of humanoids.

When the positioning system gradually changes T frame-by-frame, humanoids placed based on the previous frame's transformation T must move to the now updated obstacle locations. Thus, for each frame, DreamWalker updates the target locations of humanoids using the *wall map* and the location of dynamic obstacles. Since many obstacles may exist in the *wall map*, we only animate humanoids within a certain radius of the user and prune obstacles such that no two humanoids share the same direction from the user's perspective. When many ad-hoc obstacles in the *wall map* share a similar location (e.g., sidewalk strips), DreamWalker creates a sparse set of humanoids, giving each a "personal space" of ± 1 m.

DreamWalker animates humanoids to walk towards their assigned target locations (Figure 8), "drafting" close-by humanoids one by one to ideally keep an overall crowd moving around the user. A humanoid's walking speed depends on the distance to the user; humanoids outrun the user to an obstacle, but may never reach an assigned target location if it is far enough from the user and there is no risk of collision. To approach reasonable behavior, DreamWalker drafts only humanoids that are visible and spawns new humanoids behind the user. When humanoids are no longer useful to represent obstacles, DreamWalker discards them as soon as they are outside the user's field of view.

In the case of insular ad-hoc obstacles (e.g., pillars or lanterns), DreamWalker drafts a humanoid and stops them at the target location. This idling behavior could be supplemented by animations such as tying shoelaces, playing with their phone, or looking up in future versions.

Guiding users towards their destination in VR

DreamWalker points into the direction the user needs to follow similar to a real-world navigation system, such as Google Maps. The pointing arrow at the bottom of the user's field of view thereby respects discovered obstacles and points into the direction of a walkable path in the *height gradient map*. In the case of the third visualization technique, the animated humanoids provide a perhaps even stronger cue for the walkable path. The use of the arrow serves two purposes. It keeps navigation to a minimum, revealing just enough for the user to maintain on track. It also helps our redirected walking algorithm to hide world rotations. The user needs to walk roughly in the direction of the arrow in order to reach their destination, but can freely explore the virtual world by walking given

that they do not collide with any virtual obstacles (including generated humanoids).

EVALUATION

The purpose of this evaluation was to test DreamWalker's performance with inexperienced participants. Our key interest was in determining if DreamWalker can deliver on the two goals we started out with: preventing collisions while real-walking an enjoyable and immersive virtual experience.

Task and Procedure

Participants' task during each trial of the evaluation was to put on the headset and start walking, thereby following the direction of the arrow and avoiding static as well as dynamic obstacles in VR. Participants finished the trial when they arrived at a virtual stop sign that indicated the end of the walk.

We chose two common locations on our campus that participants walked between and obtained a GPS route between the two from Google Maps (Path 1 in Figure 4). This route led through a shaded tree area with changing inclination, through open spaces, traffic-free roads, along hedges, through an overpass, and through a busy commons area full of chairs, tables, pillars, umbrellas, glass facades, and people. Path planning produced a path in Manhattan that matched the GPS trace with acceptable redirection as described above, such that participants walked along virtual sidewalks, streets, and squares populated with moving cars, pedestrians, and parked cars, trucks, road blocks, benches, hot dog stands, and idling people.

Participants completed the study in two conditions, performing one walk during each. In the first condition ('DreamWalker'), participants used our DreamWalker system as described in this paper (i.e., including environment sensing, positioning, redirection, and obstacle management) and received no instructions beyond avoiding virtual objects and following the arrow's direction. In the second condition ('RGBD'), participants also used DreamWalker, but this time only DreamWalker's redirection system was switched on in addition to blending in the RGB depth cameras' raw texturized meshes within an area of five meters (same range of RGB-D data as the system processed in 'DreamWalker'). This allowed participants to see the texture of the ground, upcoming obstacles (including their height and texture), as well as objects next to them. No humanoids or other obstacle representations were produced during runtime in 'RGBD'.

After each walk, participants immediately filled out the iPQ presence questionnaire [29, 30] as well as a questionnaire about their perceived safety and enjoyment.

Participants walked the same path in both conditions and received no training beforehand. The order of conditions was counterbalanced across participants. A single walk typically took between 9 and 15 minutes (see the video figure for a full walkthrough). We took a number of measures to ensure that participants were walking safely beyond their use of DreamWalker. First, the path between the two GPS locations did not cross any active road or area that may risk participants' safety, though it did contain various small pillars, tactile domes, mounts that are easy to stumble over, curbs, and so on.

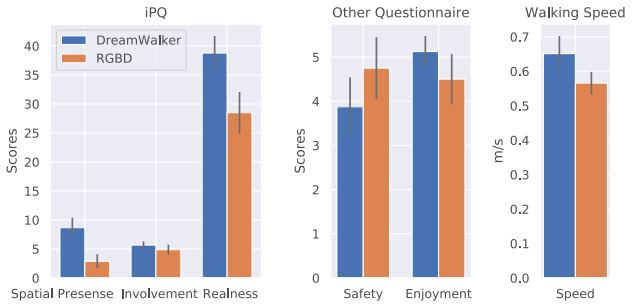

Figure 9. Results of the evaluation: A) iPQ, B) perceived safety and enjoyment, and C) walking speed.

Second, two experimenters accompanied each participant at all times, trailing them in close proximity to either verbally or, if need be, physically stop them from walking into obstacles by grabbing their upper arm.

Participants

We recruited 8 participants from our institution (ages 24–51, median = 39, 2 female). One participant had frequent prior experience with virtual reality, two participants reported having used VR less than 5 times, whereas five participants stated that they had never used VR before. All participants gave written informed consent (according to the declaration of Helsinki) and received a gift card as compensation. This study was approved by an Institutional Review Board. None of the authors or people familiar with the project participated.

Results

We analyzed all the data logged during the study to compare the performance of DreamWalker's ad-hoc and dynamic obstacle detection and representation with the gold standard that is human perception and recognition of surrounding obstacles. We implicitly also evaluated that our DreamWalker system worked well for a variety of different users.

Figure 9 summarizes the results of the iPQ and the questionnaire. The iPQ showed higher presence for 'DreamWalker' in each one of the categories (including 'General' and 'Sum', which are hidden in the chart) compared to 'RGBD'. On average, 'DreamWalker' was also rated higher in entertainment as well as in "being in the virtual world" compared to 'RGBD'. However, participants, on average, felt more confident and slightly safer walking in 'RGBD' than in 'DreamWalker'.

While walking, participants used a faster average walking speed in 'DreamWalker' than in 'RGBD' (0.65 m/s, $SD = 0.47$ vs. 0.58 m/s, $SD = 0.38$). Finally, during the 9–15 minutes of each walk, we had to correct participants' walk an average 4.5 times ($SD = 0.5$) in 'DreamWalker' compared to 3.6 times ($SD = 0.45$) in 'RGBD'. Since user saw the same amount of information in 'RGBD' as our system saw and processed in the 'DreamWalker' condition. The fact that participants in the 'DreamWalker' condition needed only little more correction than in 'RGBD' shows the effectiveness of our algorithm. We think this will improve further with higher-res depth cameras.

DISCUSSION

Our evaluation showed that DreamWalker successfully navigated participants through the real world while they experienced a vastly different virtual reality, which they rated highly enjoyable. The fact that the average walking speed in 'DreamWalker' was higher than in the baseline condition confirms participants' ratings in that they felt a high level of presence in VR, which also supports our initial design goals. This behavior might in part be explained by participants carefully watching the ground texture in 'RGBD', which resulted in lower walking speeds but higher confidence in walking.

Upon closer inspection, the locations at which the experimenter had to correct some participants' walks during the study occurred at two distinct locations. Put differently, DreamWalker did safely navigate participants throughout the entire ˜15 minute walk apart from two specific spatial locations, which bears some resemblance with recent events in self-driving cars.

We reviewed the geometry and terrain around these two locations and discovered that the narrow pavement makes a right turn, followed by a strip of grass, and another right turn. The challenge in navigating this terrain is correctly positioning the user to ensure guiding the user to the correct turn. Our environment sensing system, however, either could often not see far enough to correctly detect and tell apart both turns, thus aligning the first right turn with the expected, second turn in the planned path. The resulting discrepancy between the user turning right and the right turn in the planned real-world path caused our navigation to point into a direction in between, which confused some participants.

LIMITATIONS AND FUTURE WORK

Our evaluation uncovered DreamWalker's current limitations in addition to our own observations during continuous testing. While the RGB-D data captured from the two Intel Realsense D425 cameras on the headset covered a wide sensing range, this range is still smaller than the headset's field-of-view. Therefore, DreamWalker may sometimes miss dynamic obstacles coming in quickly from the side. In addition, some locations on campus proved challenging for DreamWalker's segmentation approach despite a small change in RGB texture, such as when green-brown grass patches appeared too similar to the adjacent pavement.

Another limitation is DreamWalker's rendering of humanoids and their behavior, which may leave an unreliable impression despite DreamWalker's robust obstacle detection. Our ambition was to produce "natural" humanoid behavior, thus spawning them outside the user's wide field of view and constantly moving them. With emerging gaze trackers in VR headsets, DreamWalker could spawn and vanish humanoids when outside the fovea yet within the field of view [20], which could substantially reduce the required humanoid motion.

Participants also indicated their preference for more realistic events in VR for increased entertainment, including car accidents, guiding policemen, stopping bikes, pets, or human billboards that all serve as virtual obstacles. We believe future versions can improve on DreamWalker by providing better crowd dynamics and more diverse appearances as well as idle animations for humanoids. While current DreamWalker users are instructed to avoid contact with all virtual objects to prevent collisions, future versions could implement passive haptic feedback this way (e.g., sitting on a chair [33] or touching passive objects through gaze-inferred intentions [6]). Since DreamWalker enables users to explore large, outdoor spaces through walking, its tracking system could also serve for large-space training systems for users with visual impairments rendering haptic feedback [46].

Finally, complete safety is an obvious limitation of using DreamWalker in practice. We addressed this by always accompanying a DreamWalker user, much like self-driving cars require drivers to be able to intervene. Participants reported *feeling* safe in our study, but it is harder to establish that DreamWalker *is* safe. Organizations such as OSHA or WHO formulate safety as the number of incidents (i.e., near misses and accidents) over a period of time [26], and objectively compare industries that way to determine the effectiveness of their safety measures.

CONCLUSIONS

We have presented DreamWalker, a VR tracking system that allows users to navigate virtual experiences through real-walking real-world outdoor spaces. DreamWalker fuses RGB depth, inside-out, and GPS tracking in real-time to accurately register the virtual world with the real world. Using redirected walking to align two paths, one in the coordinates of either world, and an obstacle detection system, DreamWalker guides the user through VR using static and animated virtual objects in the scene that cause the user to adjust their walk. In our evaluation with inexperienced participants, each participant confidently walked for 15 minutes in DreamWalker, which showed the potential of our system to make repetitive real-world walking tasks more entertaining.

ACKNOWLEDGMENTS

We thank all the participants of our study for their time and useful comments after evaluating our system. We also thank Mike Sinclair and Anthony Steed for their comments as well as Lex Story for his help with the assembly mounts.

APPENDIX

Algorithm 1 Environment Sensing

procedure EACHCAMERAFRAME
  colorimg ← Color Image from Depth Camera
  depthimg ← Depth Image from Depth Camera
  trackingdata ← Tracking data from WMR
  yoloobjects ← YOLO process *colorimg*
  dynamicobstacles ← map *yolobojects* on to *depthimg*
  pointcloud ← synthesis *colorimg depthimg*
  $colormap, heightmap$ ← transform *pointcloud* by *trackingdata*
  $globalcolormap \leftarrow average(globalcolormap, colormap)$
  globalcolorgradient ← *gradient of globalcolormap*
  heightgradient ← *gradient of heightmap*
  $globalheightgradient \leftarrow average(globalheightgradient, heightgradient)$

procedure EACHVRFRAME
  floodfillmap ← iterative flood fill $globalheightgradient, globalcolorgradient, globalcolormap$
  globalpathmap ← Ground level > threshold in *floodfillmap*
  globalwallmap ← Region with high gradient and adjacent to ground *floodfillmap*

procedure ITERATIVEFLOODFILL
  for each pixel p **do**
    iswall ← $globalheightgradient$ and $globalcolorgradient < threshold$
    isgrass ← *globalcolormap* with in a range
    if *iswall* and *isgrass* **then**
      $floopfillmap[p] \leftarrow 0$
    else
      $floopfillmap[p] \leftarrow 2 * \text{sum } floodfillmap[q]$, where q denotes the pixels surrounding p

Algorithm 2 Positioning System

procedure EACHVRFRAME
  $[p_{user/origin}] \leftarrow$ WMR tracking history
  $[p_{user/gps}] \leftarrow$ GPS tracking history
  $T_1 \leftarrow$ Fit rigid transform$[p_{user/origin}], [p_{user/gps}]$
  $[p'_{user/gps}] \leftarrow T_1 * [p_{user/origin}]$
  $[p_path] \leftarrow$ planned path
  $T_history \leftarrow$ Iterative closest point$[p'_{user/gps}], [p_{path}]$
  $T_2 \leftarrow T_1 * T_{history}$
  for A small transform T_{future} **do**
    $[p_{future}] \leftarrow$ Next 10 points on path
    $value \leftarrow$ sum *pathmap* on all $[T_2^{-1} * T_{future} * p_{future}]$
  $T_{future} \leftarrow T_{future}$ with largest $value$
  $T_3 = T_{future}^{-1} * T_2$

Algorithm 3 Redirection System

procedure EACHVRFRAME
  $M_{actual}[n+1] \leftarrow T_{user/origin}[n+1] * T_{current}[n+1] - T_{user/origin}[n] * T_{current}[n]$
  $M_{target}[n+1] \leftarrow$ scale back within limit $M_{actual}[n+1]$
  $T_{current}[n+1] \leftarrow (T_{user/origin}[n+1])^{-1} * (T_{user/origin}[n] * T_{current}[n] + M_{target}[n+1])$

REFERENCES

[1] Thaddeus Beier and Shawn Neely. 1992. Feature-based image metamorphosis. *ACM SIGGRAPH Computer Graphics* 26, 2 (jul 1992), 35–42. DOI: http://dx.doi.org/10.1145/142920.134003

[2] Bigscreen. 2014. https://bigscreenvr.com/. (2014).

[3] Tomasz Byczkowski and Jochen Lang. 2009. A Stereo-Based System with Inertial Navigation for Outdoor 3D Scanning. In *2009 Canadian Conference on Computer and Robot Vision*. IEEE. DOI: http://dx.doi.org/10.1109/crv.2009.40

[4] Haiwei Chen, Samantha Chen, and Evan Suma Rosenberg. 2018. Redirected Walking in Irregularly Shaped Physical Environments with Dynamic Obstacles. In *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. IEEE. DOI: http://dx.doi.org/10.1109/vr.2018.8446563

[5] Haiwei Chen and Henry Fuchs. 2017. Supporting free walking in a large virtual environment. In *Proceedings of the Computer Graphics International Conference on - CGI '17*. ACM Press. DOI: http://dx.doi.org/10.1145/3095140.3095162

[6] Lung-Pan Cheng, Eyal Ofek, Christian Holz, Hrvoje Benko, and Andrew D. Wilson. 2017. Sparse Haptic Proxy: Touch Feedback in Virtual Environments Using a General Passive Prop. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, New York, NY, USA, 3718–3728. DOI:http://dx.doi.org/10.1145/3025453.3025753

[7] Lung-Pan Cheng, Eyal Ofek, Christian Holz, and Andrew D. Wilson. 2019. VRoamer: Generating On-The-Fly VR Experiences While Walking inside Large, Unknown Real-World Building Environments. In *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. IEEE.

[8] Adrian David Cheok. 2010. Human Pacman: A Mobile Augmented Reality Entertainment System Based on Physical, Social, and Ubiquitous Computing. In *Art and Technology of Entertainment Computing and Communication*. Springer London, 19–57. DOI: http://dx.doi.org/10.1007/978-1-84996-137-0_2

[9] Jan Effertz and Jörn Marten Wille. 2011. Vehicle Architecture and Robotic Perception for Autonomous Driving in Urban Environments. In *Experience from the DARPA Urban Challenge*. Springer London, 209–239. DOI:http://dx.doi.org/10.1007/978-0-85729-772-3_9

[10] Oculus Go. 2018. Oculus. https://www.oculus.com/go/. (2018).

[11] Rec Room® — Against Gravity. 2018. https://www.againstgrav.com/rec-room/. (2018).

[12] Jeremy Hartmann, Christian Holz, Eyal Ofek, and Andrew D. Wilson. 2019. RealityCheck: Blending Virtual Environments with Situated Physical Reality. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. ACM, New York, NY, USA, Article 347, 12 pages. DOI: http://dx.doi.org/10.1145/3290605.3300577

[13] Christian Hirt, Markus Zank, and Andreas Kunz. 2017. Real-time wall outline extraction for redirected walking. In *Proceedings of the 23rd ACM Symposium on Virtual Reality Software and Technology - VRST '17*. ACM Press. DOI:http://dx.doi.org/10.1145/3139131.3143416

[14] Christian Hirt, Markus Zank, and Andreas Kunz. 2018. Preliminary Environment Mapping for Redirected Walking. In *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. IEEE. DOI: http://dx.doi.org/10.1109/vr.2018.8446262

[15] M. Keller and F. Exposito. 2018. Game Room Map Integration in Virtual Environments for Free Walking. In *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. 763–764. DOI: http://dx.doi.org/10.1109/VR.2018.8446385

[16] VIVE™ | VIVE Focus Developer Kit. 2018. https://developer.vive.com/eu/vive-focus-for-developer/. (2018).

[17] Eike Langbehn, Frank Steinicke, Markus Lappe, Gregory F. Welch, and Gerd Bruder. 2018. In the blink of an eye. *ACM Transactions on Graphics* 37, 4 (jul 2018), 1–11. DOI: http://dx.doi.org/10.1145/3197517.3201335

[18] Jonathan Long, Evan Shelhamer, and Trevor Darrell. 2015. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 3431–3440.

[19] Sebastian Marwecki, Maximilian Brehm, Lukas Wagner, Lung-Pan Cheng, Florian 'Floyd' Mueller, and Patrick Baudisch. 2018. VirtualSpace - Overloading Physical Space with Multiple Virtual Reality Users. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems - CHI '18*. ACM Press. DOI:http://dx.doi.org/10.1145/3173574.3173815

[20] Sebastian Marwecki, Andrew D. Wilson, Eyal Ofek, Mar Gonzalez-Franco, and Christian Holz. 2019. Mise-Unseen: Using Eye-Tracking to Hide Virtual Reality Scene Changes in Plain Sight. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology (UIST '19)*. ACM, New York, NY, USA, 13.

[21] Quake VR on soccer field. 2019. https://www.youtube.com/watch?v=acEz98Ol8NI. (2019).

[22] Fallout 4 VR on Steam. 2017. https://store.steampowered.com/agecheck/app/611660/. (2017).

[23] Brian Peasley and Stan Birchfield. 2013. Replacing Projective Data Association with Lucas-Kanade for KinectFusion. In *2013 IEEE International Conference on Robotics and Automation*. IEEE. DOI: http://dx.doi.org/10.1109/icra.2013.6630640

[24] Sharif Razzaque, Zachariah Kohn, and Mary C Whitton. 2001. Redirected walking. 9 (2001), 105–106.

[25] Joseph Redmon. 2013–2016. Darknet: Open Source Neural Networks in C. http://pjreddie.com/darknet/. (2013–2016).

[26] Occupational Safety, Health Administration, and others. 2016. Improve Tracking of Workplace Injuries and Illnesses. Final rule. *Federal register* 81, 92 (2016), 29623.

[27] Anthony Scavarelli and Robert J. Teather. 2017. VR Collide! Comparing Collision-Avoidance Methods Between Co-located Virtual Reality Users. In *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems - CHI EA '17*. ACM Press. DOI: http://dx.doi.org/10.1145/3027063.3053180

[28] P. Schmitz, J. Hildebrandt, A. C. Valdez, L. Kobbelt, and M. Ziefle. 2018. You Spin my Head Right Round: Threshold of Limited Immersion for Rotation Gains in Redirected Walking. *IEEE Transactions on Visualization and Computer Graphics* 24, 4 (April 2018), 1623–1632. DOI:http://dx.doi.org/10.1109/TVCG.2018.2793671

[29] Thomas Schubert, Frank Friedmann, and Holger Regenbrecht. 1999. Embodied Presence in Virtual Environments. *Visual Representations and Interpretations* (1999), 269–278. DOI: http://dx.doi.org/10.1007/978-1-4471-0563-3_30

[30] Thomas Schubert, Frank Friedmann, and Holger Regenbrecht. 2001. The Experience of Presence: Factor Analytic Insights. *Presence: Teleoperators and Virtual Environments* 10, 3 (Jun 2001), 266–281. DOI: http://dx.doi.org/10.1162/105474601300343603

[31] Stefania Serafin, Niels C. Nilsson, Erik Sikstrom, Amalia De Goetzen, and Rolf Nordahl. 2013. Estimation of detection thresholds for acoustic based redirected walking techniques. In *2013 IEEE Virtual Reality (VR)*. IEEE. DOI:http://dx.doi.org/10.1109/vr.2013.6549412

[32] Maeve Serino, Kyla Cordrey, Laura McLaughlin, and Ruth L. Milanaik. 2016. Pokémon Go and augmented virtual reality games. *Current Opinion in Pediatrics* 28, 5 (oct 2016), 673–677. DOI: http://dx.doi.org/10.1097/mop.0000000000000409

[33] Misha Sra, Sergio Garrido-Jurado, and Pattie Maes. 2017. Oasis: Procedurally Generated Social Virtual Spaces from 3D Scanned Real Spaces. *IEEE Transactions on Visualization and Computer Graphics* (2017), 1–1. DOI: http://dx.doi.org/10.1109/tvcg.2017.2762691

[34] Misha Sra, Sergio Garrido-Jurado, and Chris Schmandt. 2016. Procedurally generated virtual reality from 3D reconstructed physical space. *Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology - VRST '16* (2016). DOI: http://dx.doi.org/10.1145/2993369.2993372

[35] Misha Sra, Xuhai Xu, Aske Mottelson, and Pattie Maes. 2018. VMotion. In *Proceedings of the 2018 on Designing Interactive Systems Conference 2018 - DIS '18*. ACM Press. DOI: http://dx.doi.org/10.1145/3196709.3196792

[36] Frank Steinicke, Gerd Bruder, Jason Jerald, Harald Frenz, and Markus Lappe. 2008. Analyses of human sensitivity to redirected walking. In *Proceedings of the 2008 ACM symposium on Virtual reality software and technology - VRST '08*. ACM Press. DOI: http://dx.doi.org/10.1145/1450579.1450611

[37] Evan A. Suma, Zachary Lipps, Samantha Finkelstein, David M. Krum, and Mark Bolas. 2012. Impossible Spaces: Maximizing Natural Walking in Virtual Environments with Self-Overlapping Architecture. *IEEE Transactions on Visualization and Computer Graphics* 18, 4 (April 2012), 555–564. DOI: http://dx.doi.org/10.1109/TVCG.2012.47

[38] Qi Sun, Arie Kaufman, Anjul Patney, Li-Yi Wei, Omer Shapira, Jingwan Lu, Paul Asente, Suwen Zhu, Morgan Mcguire, and David Luebke. 2018. Towards virtual reality infinite walking. *ACM Transactions on Graphics* 37, 4 (jul 2018), 1–13. DOI: http://dx.doi.org/10.1145/3197517.3201294

[39] Marvin Teichmann, Michael Weber, Marius Zoellner, Roberto Cipolla, and Raquel Urtasun. 2018. Multinet: Real-time joint semantic reasoning for autonomous driving. In *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 1013–1020.

[40] Daniel Thalmann. 2007. *Crowd Simulation*. American Cancer Society. DOI: http://dx.doi.org/10.1002/9780470050118.ecse676

[41] Tom Warren The Verge. 2018. Microsoft is bringing the SharePoint work environment to virtual reality headsets. https://www.theverge.com/2018/5/21/17376422/microsoft-sharepoint-spaces-mixed-reality-virtual-\reality-features. (2018).

[42] Khrystyna Vasylevska, Hannes Kaufmann, Mark Bolas, and Evan A. Suma. 2013. Flexible spaces: Dynamic layout generation for infinite walking in virtual environments. In *2013 IEEE Symposium on 3D User Interfaces (3DUI)*. IEEE. DOI: http://dx.doi.org/10.1109/3dui.2013.6550194

[43] The VOID. 2016. https://www.thevoid.com/. (2016).

[44] David Waller, Eric Bachmann, Eric Hodgson, and Andrew C. Beall. 2007. The HIVE: A huge immersive virtual environment for research in spatial cognition. *Behavior Research Methods* 39, 4 (nov 2007), 835–843. DOI:http://dx.doi.org/10.3758/bf03192976

[45] Fisher Yu, Vladlen Koltun, and Thomas Funkhouser. 2017. Dilated Residual Networks. In *Computer Vision and Pattern Recognition (CVPR)*.

[46] Yuhang Zhao, Cynthia L. Bennett, Hrvoje Benko, Edward Cutrell, Christian Holz, Meredith Ringel Morris, and Mike Sinclair. 2018. Enabling People with Visual Impairments to Navigate Virtual Reality with a Haptic and Auditory Cane Simulation. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. ACM, New York, NY, USA, Article 116, 14 pages. DOI: http://dx.doi.org/10.1145/3173574.3173690

[47] Yaguang Zhu, Baomin Yi, and Tong Guo. 2016. A Simple Outdoor Environment Obstacle Detection Method Based on Information Fusion of Depth and Infrared. *Journal of Robotics* 2016 (2016), 1–10. DOI: http://dx.doi.org/10.1155/2016/2379685