Redirected Walking Strategies in Irregularly Shaped and Dynamic Physical Environments

Haiwei Chen* Samantha Chen† Evan Suma Rosenberg‡

USC Institute for Creative Technologies, Los Angeles, CA

ABSTRACT

Redirected walking (RDW) is a Virtual Reality (VR) locomotion technique that enables the exploration of a large virtual environment (VE) within a small physical space via real walking. Thus far, the physical environment has generally been assumed to be rectangular, static, and free of obstacles. However, it is unlikely that real-world locations that may be used for VR fulfill these constraints. In addition, accounting for a dynamically changing physical environment allows RDW algorithms to accommodate gradually mapped physical environments and moving objects. In this work, we introduce novel approaches that adapt RDW algorithms to support irregularly shaped and dynamic physical environments. Our methods are divided into three categories: novel RDW Greedy Algorithms that provide a generalized approach for any VE, adapted RDW Planning Algorithms that provide an optimized solution when virtual path prediction is available, and last but not least, techniques for representing irregularly shaped and dynamic physical environments that can improve performance of RDW algorithms.

Index Terms: H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems—Artificial, augmented, and virtual realities; I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Virtual reality

1 INTRODUCTION

Redirected walking (RDW) is a real-walking locomotion technique used in Virtual Reality (VR) that enables the users to explore a virtual environment (VE) that is larger than the available physical environment [6]. As the user moves, imperceptible adjustments such as rotations and translations are made to the virtual world in order to manipulate the user's path in the physical environment. The goal of redirected walking is to utilize these rotation and translation gains to guide the users to collision-free paths in the physical environment so that they may enjoy the benefits of real walking when exploring a larger VE.

Past research has proposed a number of different redirected walking strategies. Generalized steering algorithms, such as Steer-To-Center and Steer-To-Orbit, redirect the user towards or around the center of the physical environment in order to keep the user away from the boundaries [6]. These algorithms can be understood as greedy approaches, since the steering direction does not depend on previous or future states of the user. There also exist search-based optimization approaches [3, 11]. Contrary to the greedy approaches, these algorithms plan out a series of redirection gains based on a priori known or predicted paths in the VE. In this paper, we categorize RDW strategies into two main categories: 1) Greedy Algorithms and 2) Planning Algorithms.

*e-mail: chenh@ict.usc.edu

†e-mail: samchen@ict.usc.edu

‡e-mail: suma@ict.usc.edu

In general, RDW algorithms perform analysis between a geometrical representation of a physical tracked space, which provides the boundaries for real walking, and a geometrical representation of a virtual environment, which motivates navigation. Due to past designs of VR tracking systems, nearly all previous works in RDW conveniently represent the physical tracked space as a rectangle, with the exception of [9]. The benefit of such representation is twofold: 1) detecting physical path collision with rectangular boundaries is trivial and 2) a fixed center exists to serve as a target for greedy strategies such as Steer-To-Center. However, a rectangular tracked space does not represent many real-world locations where commodity VR systems can be deployed, such as the living rooms of peoples' homes. These locations are likely to be irregularly shaped and contain obstacles such as furniture.

In comparison, the VEs in RDW experiments are commonly designed to have complex structures in order to deliver meaningful immersive experiences. For VEs, the main concern is whether reliable predictions of the user's movement can be extracted from the geometry of the virtual world. To simplify virtual path prediction, many works have designed hallway-like VE structures or have assigned targets for the users to walk towards [5, 6, 8]. Others have developed strategies to predict possible paths in a VE based on navigation meshes [1, 10]. [4] also proposes a method for short-term path prediction solely based on head tracking data. The availability of path prediction is of particular importance to RDW Planning Algorithms, which will be discussed in section 1.2.

1.1 Dynamic Physical Environment

The development of wide-area, 6DOF tracking systems in past decades has made real walking locomotion interfaces feasible. Thus far, commerical VR systems have most commonly used outside-looking-in tracking with externally mounted trackers. Such design shapes the literature of RDW to assume the existence of a "tracking space" - a carefully setup stage with limited area. Leaving the tracking space therefore not only means facing risk of running into physical objects, but also the loss of tracking.

In recent years, techniques such as Simultaneous Localization and Mapping (SLAM) have started to be integrated into tracking of Augmented Reality (AR) systems. The Microsoft Hololens[1], for instance, demonstrates satisfactory inside-out tracking capabilities; the walkable space does not need to be limited or configured. Mapping of the environment is done on the fly, and thus the users can theoretically travel as far in the physical space as possible and still stay tracked. In other words, the concept of a tracking space becomes ambiguous with the use of such tracking systems. Inside-out tracking inspires us to consider an interesting question: to what extent could VR locomotion benefit? Different from AR, VR Head-Mounted Display (HMD) occludes the users' view of the physical world. Suppose one is allowed to navigate "blindly" in real-world locations, where the entire physical space is reachable but at the same time not carefully prepared. It appears that two challenges arise: 1) Sensors on HMD that can "see" the world must ensure safety of the users and prevent collisions with physical objects. 2) Although the

[1] https://www.microsoft.com/en-us/hololens

enlarged physical space provides more freedom for real walking, it is still often small and topologically different compared to the VE in which the user navigates.

In light of this, we believe that extending RDW techniques to support irregularly shaped and dynamically updated physical environment can be valuable for the future development of VR systems with inside-out tracking capabilities. In this work, we focus on challenges of adapting existing RDW algorithms and devising new algorithms to support a more flexible representation of the physical world. Therefore, for the following methods, we assume that a dynamic mapping of the physical space is given by an oracle machine (a black box that provides perfect information). In general, a physical space can be represented as a weakly simply polygon (simple polygon with interior holes). We thus define the mapping of an irregularly shaped and dynamic physical environment as follow:

- The mapping is given as a Graph $G(V,E)$
- Every vertex V stores a unique 2D Cartesian coordinate.
- Every vertex V is connected to exactly 2 edges.
- Every two edges E_1, E_2 may only intersect at the end points.
- At every frame, the oracle reports either the same graph or a new graph.

Notice that the above definition gives a set of cycle graphs. Each cycle graph either represents an outer boundary or an interior obstacle in the physical environment. Graph updates given by the oracle do not distinguish between a physical environment that contains dynamically moving objects and a static physical environment that is iteratively scanned. Both cases are simply treated as a new graph.

1.2 RDW Strategies Based on VE Geometries

In the following sections, we analyze two categories of RDW algorithms: Greedy Algorithms and Planning Algorithms. The reason for such categorization is based on the observation that the effectiveness of these algorithms depend on the characteristics of the provided VE. Imagine two types of VEs: 1) a virtual maze VE_1, 2) a wide-open virtual plaza VE_2. For VE_1, a set of virtual paths that the users can take can be extracted. The paths may be singular or branching. In this case, both greedy approaches and planning approaches can be used to redirect the users. Since the planning approaches can optimize over the entire virtual paths, we speculate that it outperforms the greedy algorithm in most cases, as the latter does not take into consideration the future states of the users. However, for the case of VE_2, unless the users are instructed to walk towards specific targets, they are allowed to walk in any direction. Therefore, the only approach to predict movement direction is based on the velocity of the user. However, this results in a virtual path that is typically not long enough for the planning algorithm to optimize over. It appears that greedy approach is the only solution.

The above discussion provides the context for us to present two categories of RDW algorithms in parallel at section 2.1 and 2.2. Each faces a set of unique challenges when being adapted to irregularly shaped and dynamic physical environments. At the same time, the selection of algorithms is advised to consider the characteristics of the VE. In the following parts of the paper, Section 2.1 describes two novel Greedy Algorithms that are able to support any VE. Section 2.2 describes adaptation of Planning Algorithms that perform best in VEs where long-term virtual path prediction can be extracted from the VE. In section 2.3, we present techniques for representing physical environments, aiming to improve performance for both types of RDW algorithms.

2 Method

RDW algorithms provide a means to select and apply redirection techniques (RETs) as the user navigates in a VE. For the algorithms discussed in our work, the RETs include rotational gain, curvature gain, translational gain, and reset. Different RETs are applicable under different conditions: rotation gain is the extra percentage of rotation applied to head rotation, and thus applicable only if head rotation is observed; curvature gain is a small amount of rotation applied to the user's view in VE and applicable only if they walk at a certain speed; translational gain is the adjustment of the walking speed in the VE; reset is a special type of RET that requires the users to turn in place if other RETs cannot fully redirect them to avoid collisions with the physical environment. A common goal of RDW algorithms is to minimize the number of resets, as resets generally interrupt the user's virtual experience.

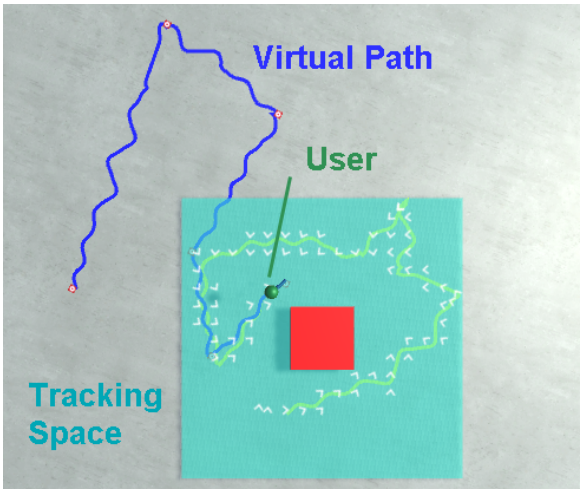

Figure 1: A visualization of the Steer-to-Farthest algorithm using a simulated walker, a VE with a large open space, and a tracking space (cyan) with an obstacle (red). We show the virtual path (blue), the physical path (yellow), and the selected steering direction (white) in each grid cell along the user's physical path. In this simulation, we set $n = 18$ and $k = 3$.

2.1 RDW Greedy Algorithms

Greedy algorithms provide a generalized approach to apply RDW in any VE as only short-term predictions of the user's walk direction are required. According to [4], walk direction can be predicted from a temporal analysis of the tracking data. Unlike planning algorithms, greedy algorithms are able to provide RDW solutions to VEs with large open spaces, where it is infeasible to find long virtual paths that reliably predict the user movement. However, existing greedy algorithms, such as Steer-To-Center and Steer-To-Orbit, are limited in that they assume the existence of a center in the physical environment and the direction towards this center always leads away from the physical boundaries. A physical center, however, is not well-defined for irregularly shaped physical environments. For example, although a geometric center can be computed for a room-shaped environment, the center could be occupied by an obstacle (e.g. a table). In light of this, we develop two novel greedy algorithms that do not assume the existence of a physical center and therefore can support RDW in irregularly shaped, dynamic physical environments.

2.1.1 Steer-to-Farthest

We introduce an algorithm named Steer-To-Farthest (see Algorithm 1), which steers the user towards one of the farthest physical boundary points from the user's current position. Given the user's position, the algorithm uniformly samples n directions and performs raytracing to measure the distance to the closest physical boundary in each direction. From these n directions, we select the k directions that

Data: X: User State; $grid$: Grid; n: n; k: k

```
SteerToFarthest(X, grid, n, k)
    cell = grid.getCell(X)
    if cell.isUpdated() then
        /* Get stored k-farthest directions */
        kFarthest = cell.getDirections()
    end
    else
        /* Compute k-farthest directions */
        possibleDirections = SampleDirections(X, n)
        kFarthest = SelectFarthest(possibleDirections, k)
        cell.setDirections(kFarthest)
        cell.markUpdated()
    end
    /* Select lowest cost direction */
    steeringDirection = null, lowestCost = ∞
    for dir in kFarthest do
        if cost(X, dir) < lowestCost then
            steeringDirection = dir
            lowestCost = cost(X, dir)
        end
    end
    SteerToTarget(X.pos + steeringDirection)
```

Algorithm 1: Steer-to-Farthest algorithm for selecting the current steering direction. Given this direction, gains can be applied similarly to Steer-To-Center so that the user's direction converges to the desired direction.

correspond to the largest distances but are also at least $\theta = \frac{360^\circ}{k}$ apart. This ensures that the set of k directions have a certain amount of variance. For convenience, we refer to the k directions as the k-farthest directions. The algorithm then selects one of the k-farthest directions as the steering direction via the following cost function:

$$cost(\vec{d}) = w_1 * angle(\vec{d}_{user}, \vec{d}) + w_2 * distFromBoundary(\vec{d})$$

in which $\vec{d}$ is a potential steering direction, $\vec{d}_{user}$ is the user's current direction, and w_1 and w_2 are tuned weight parameters. Given a steering direction, a standard Steer-To-Center algorithm can be used for redirection, except the direction towards the center needs to be replaced by the direction of interest. It is therefore named *SteerToTarget* in Algorithm 1. Steering the user in one of the k-farthest directions essentially steers the user away from the closer physical boundaries. Through this greedy approach, Steer-to-Farthest aims to prevent user collisions with the physical boundaries.

Rather than selecting the farthest direction, our algorithm chooses from among the top k directions using a cost function. This enables our algorithm to find a steering direction that is not only far from the physical boundary point but also relatively aligned with the user's current walk direction, making it an easier direction to redirect the user along. Additionally, taking walk direction alignment into account not only provides a way to break ties between different directions that correspond to similar distances but also helps select steering directions that are relatively similar as the user moves. This is necessary in order to effectively exploit the user's head rotations for redirection. If the selected steering directions are similar over a period of time, the applied rotation and curvature gains will accumulate in the same general direction, effectively steering the user either towards their left or their right. On the other hand, if the steering directions vary significantly as the user moves, it is possible that current steering target fluctuates between left and right of the user, causing the applied gains to "cancel out" and for redirection to be ineffective. Using an initial implementation of the algorithm and a simulated walker, we present a visualization of the selected steering directions along the user's path in Figure 1.

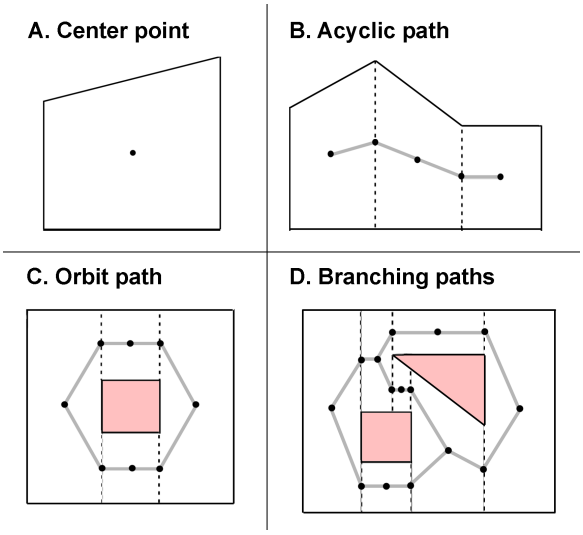

Figure 2: Diagrams of the trapezoidal road maps for four example physical environments. The dashed lines show the trapezoidal regions, and the gray lines show the road map. Our algorithm steers the user towards the center point in Figure A, along the acyclic path in Figure B, along the orbit path in Figure C, and along the lowest cost branching path in Figure D.

Reducing Computational Cost: As the user moves, the k-farthest directions must be recomputed using the updated user position to remain accurate. However, this is unnecessary work, especially in the case of a mostly static physical environment. To reduce computational costs, we can discretize the physical space and store the k-farthest directions. To accomplish this, we overlay a grid with a cell size of 0.5m x0.5m onto the physical environment. For each (partially) inbound grid cell, we use the center of the cell as the approximated user position and store the k-farthest directions. If the center lies outside the physical boundaries, we sample a random position in cell that is bounded by the physical boundaries for calculation. If an inbound position does not exist, we discard the cell. Given a static environment, the k-farthest directions only need to be computed when the user enters an unvisited grid cell. Alternatively, we can also precompute the entire grid offline.

Handling Environment Updates: When the physical boundaries of the environment are changed, it becomes necessary to recompute the k-farthest directions as the farthest directions may have changed. Depending on the expected frequency of environmental updates, the entire grid could be recomputed at once or on-the-go. While both methods are suitable for a mostly static environment, highly dynamic environments would likely benefit from the latter strategy. To compute on-the-go, each grid cell needs to be marked as "dirty" (requiring an update) when an environment change is detected. Then, as the user moves, a grid cell can be marked as up-to-date when its k-farthest directions are recomputed.

2.1.2 Trapezoidal Road Map Algorithm

Inspired by Steer-To-Orbit, our second algorithm (See Algorithm 2) focuses on finding a collision-free path for the user to follow in an irregularly shaped physical environment with obstacles. To find such a path, a similar approach is taken from robotic motion planning [2] which utilizes trapezoidal decomposition. This algorithm computes a trapezoidal map of the physical environment and then to builds a road map by connecting the center of each trapezoid with the midpoint of its sides (See Figure 2). The generated road map enables mobile robots to find a collision-free path between any two points in the environment, but in the case of redirected walking, it can be

```
Data: X: User State; map: Trapezoidal Road Map
1  TrapezoidMapAlg(X, map)
2      bestPath = null, lowestCost = ∞
3      region = map.getRegion(X)
4      Paths = map.getPaths(region)
       /* Find the lowest cost path                    */
5      for path in Paths do
6          if cost(X, path) < lowestCost then
7              bestPath = path
8              lowestCost = cost(X, path)
9          end
10     end
11     closestPoint = getClosestPoint(X, bestPath)
12     if Dir(X.pos, closestPoint) · Dir(bestPath) ≠ 0 then
13         SteerToTarget(closestPoint)
14     end
15     else
           /* Find better aligned path endpoint        */
16         p1, p2 = bestPath.getEndpoints()
17         angle1 = Angle(p1 − X.pos, X.dir)
18         angle2 = Angle(p2 − X.pos, X.dir)
19         target = angle1 < angle2 ? p1 : p2
20         ε = CalculateMisalignment(X, bestPath)
           /* k is a scalar                            */
21         gain = k · ε
22         gain = Clamp(gain, 0, maxGain)
           /* Always apply gain value passed in        */
23         SteerToTarget(target, gain)
24     end
```

Algorithm 2: Trapezoidal Road Map Algorithm that steers the user along the lowest cost path.

used as a collision-free path to steer the user along. As each path segment is generated by using the trapezoid center and the midpoint of its side, the endpoints of each segment are the furthest away from physical boundaries in the vertical direction. This helps generate a path reasonably away from the physical boundaries.

Algorithm 2 steers the user along the road map path as follows. First, the algorithm finds the trapezoidal region the user is in, which we call K. If the direction from the nearest point p on the path in K to the user is not perpendicular to the path at p, our algorithm steers the user to that nearest point. This describes the situation in which the user's position cannot be projected on the path. Since moving between any two points within a trapezoidal region is collision-free and the nearest point in such a situation is the trapezoid center, steering to the nearest point is equivalent to Steer-to-Center with respect to the trapezoidal region. Once the position of the user can be projected onto the road map path, our algorithm applies rotational gains and curvature gains in order to reduce the misalignment, which is defined as

$$\varepsilon = w_1(\Theta_p - \Theta_u) + w_2(P_p - P_u) \tag{1}$$

where P_u, Θ_u are the position and orientation of the user and P_p, Θ_p are the position and orientation of the projected point on the path. The gain values are linearly dependent on the misalignment.

Note that while this algorithm is inspired by Steer-to-Orbit, it is possible for the road map to contain no orbit paths (See Figures 3A and 3B). This occurs when the physical environment contains no obstacles. In this case, the user is either steered towards a single point for trapezoid-shaped physical environments (identical to Steer-to-Center) or along a non-cyclic path for environments of other shapes.

Handling Branching Paths: The generated road map may also contain branching paths within some trapezoidal regions. This occurs for physical environments that contain at least two obstacles (See Figure 3D). To select a single path from the branches, we utilize a cost function to align the current user configuration with one of the branching paths (see line 5-9 in Algorithm 2). The cost functions, not specified here, is advised to take into account the alignment between the path direction and the user's walk direction, in addition to user safety metrics such as the narrowness of the next trapezoidal region that the path leads to.

Handling Environment Updates: When the physical environment is updated, it is only necessary to recompute the affected area of the road map. This area only includes any trapezoidal regions that are in contact with the updated physical boundaries. We believe this localness property is particularly beneficial for large, highly dynamic physical environments, in which small changes to environment frequently occur.

2.2 RDW Planning Algorithms

RDW planning is the task of finding a set of RETs that best redirect known or predicted virtual paths in a cost-minimization sense. Different optimization algorithms have been proposed in the past to handle both singular paths and branching paths [3, 11]. As a generalization, the planning algorithms concerned by this paper require as inputs a parametric model of the virtual path $s(t)$, a set of actions u consisting of possible range of RET values, a mapping of the physical environment G, and a cost function $g_k(s_k, u_k, G)$, which denotes the cost of redirecting a path s_k by some actions u_k in a physical environment G. The optimization program can therefore be described as:

$$\begin{aligned} \min \quad & g_k(s_k, u_k, G) \\ \text{s.t.} \quad & u \leq A \\ & u \geq B \end{aligned}$$

Existing algorithms MPCRed and FORCE [3, 11] describe combinatorial optimization approaches for RDW planning that fit into our formulation of the optimization problem. Their differences mainly lie in the modeling of virtual paths and the specific designs of the cost function. One strength of both algorithms is that they are not limited by the representation of the physical environment as these details are abstracted into the cost function. However, it also means that previous works have not taken into account the shape of the physical space and the possibility that the physical space can change. The following paragraphs focus on challenges that arise as RDW planning algorithms are adapted to irregularly shaped, dynamic physical environments.

Cost Computation for Irregularly Shaped Physical Environment: Introducing a geometrically more complicated physical mapping G means that methods for calculating cost need to be addressed. Time complexity of the computation also increases as the number of edges in G grows. We propose that given our representation of G, the computation boils down to a crossing number algorithm [7] that determines whether a point is in the walkable area (f_1), and an algorithm that computes the shortest distance to the nearest boundary (f_2). Fortunately, the crossing number algorithm gives the correct determination even when the mapping contains interior boundaries. Suppose a cost function can generally be defined as

$$g_k(s_k, u_k, G) = IsPathInBound(s_k, u_k, G) *$$

$$(w_1 * NumberOfResets(u_k) + w_2 * DistanceToBoundary(s_k, u_k, G))$$

where $IsPathInBound()$ returns two values $a_{InBound} << a_{OutOfBound}$. f_1 can be used to examine whether s_k is crossing physical boundaries of G, and f_2 can be used to calculate the distance term.

We integrated the cost computation into a MPCRed implementation, allowing physical boundaries to be represented by G. Figure

1 shows a visualization of the original predicted paths in VE (top figures) and the paths transformed by a set of optimized actions (bottom figures). The transformed paths lie completely in the cyan physical space and outside the red obstacle.

Online Planning: If the virtual path s and the physical mapping G are static, the planning algorithm can compute an optimized policy offline and apply it accordingly as the users walk through the virtual path. However, we suggest that there are two major benefits in online planning: 1) it supports a dynamic physical mapping G, where new edges may be added, or positions of vertices may change and 2) it supports online path predictions based on real-time states of the users. The challenge faced by online planning is that current planning algorithms are rather expensive in terms of computation time, as combinatorial optimization approaches are used. If a walking path has k segments that can be redirected with k actions, the upper bound of time complexity is $O(n^k)$, with n equal the number of possible actions. We propose that a path of many segments can be divided into several connected paths. each with a small number of segments. Each shorter path can be planned online as the user approaches its starting point. This approach reduces optimality of the best actions. But if the physical environment is constantly being updated, planning for a long path may not provide more benefits, since the planned actions may become obsolete before the user traverses the long virtual path. Furthermore, we propose that three heuristics can be used to determine when to plan for the next set of actions:

- Avoid planning when the user is turning. As rotational gain typically results in the most effective redirection, failure to apply it or applying an incorrect gain can cause the planned path to misalign with the actual physical path significantly.
- Plan for new actions only if the difference between an updated graph G' and the original graph G is larger than a threshold. A naive way of calculating difference is to calculate the sum of shifted distance for each pair of corresponding vertices.
- Plan for new actions if the users are found to deviate from the predicted physical path above a threshold. In this case, new path prediction needs to be computed.

Handling Misalignment: A drawback of online planning is that during computation for a new set of actions, the users may still be moving. If no RETs can be applied at this time, the actual physical path taken by the users will eventually misalign with the projected physical paths based on the planned actions. Additionally, the users may also deviate from the predicted virtual path as they walk in the VE. This causes the same kind of misalignment. Consider the misalignment defined by Equation 1 in the previous section 2.1.2. To handle misalignment, the following extra gains can be injected:

$$\rho^* = (\Delta\theta - \Delta\theta_0)\rho_{max} + \Delta\theta_0(\rho_{max} - \rho)$$

In the equation, $\Delta\theta$ is the total head rotation, $\Delta\theta_0$ is the expected head rotation, ρ_{max} is the maximum gain in one direction, and ρ is the gain being applied. The first part of the equation applies rotation gains to unexpected head rotation that arises from natural human movement, and the second part of the equation increases the planned rotational gain to the maximum value. The extra redirection can be injected until ε converges to zero as the user walks. This counter-misalignment approach can be loosely interpreted as a hybrid method of RDW planning and Steer-to-Orbit, since the additionally injected gains steer the user towards the planned physical path (analogous to a cut and deformed orbit) in a greedy manner.

2.3 Techniques for Representing the Physical Environment

Thus far, we have assumed that a representation of physical environment is given as a well-defined graph G (See Section 1.1). If the given representation does not follow our graph definition, algorithms are needed to convert it such that it can be used by our RDW algorithms. In this section, we consider two cases: 1) the representation is given as a point cloud (Section 2.3.1) and 2) the representation does not form a bounded region (Section 2.3.2).

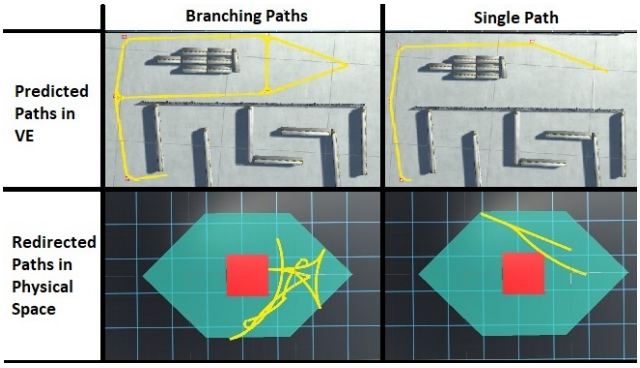

Figure 3: Path visualizations of an adapted planning algorithm that supports irregularly shaped physical environments. The figure shows results for both a branching path example (left column) and a single path example (right column). The top row depicts the predicted virtual paths in a VE. The bottom row depicts the corresponding physical paths in an irregularly shaped space. The red square represents an obstacle.

Another observation is that the representation itself affects the performance of the presented RDW algorithms. At the same time, the representation needs not accurately match with the actual physical environment. The input graph G is valid with respect to user safety as long as avoiding collision with edges in G ensures that the user does not collide with physical objects that bound the walkable area. Therefore, the following four potential strategies are designed to simplify or alter the mapping of the physical environment to enhance performance of the RDW algorithms described above. Each of the following strategies is visualized in Figure 4.

2.3.1 Handling Point Cloud Data

The time complexity of the aforementioned algorithms increase linearly with the number of edges in the physical environment, whether it is the computing cost in the planning approaches, ray casting in the Steer-To-Farthest approach, or generating the road map in the trapezoidal approach. Suppose the raw input of the physical environment mapping is a 3D scanned point cloud projected onto the xz-plane. It is likely that the point cloud contains numerous data points and noise. Therefore, it is important to consider approximating a minimum set of edges that can bound the point clouds. This allows computations in the presented RDW algorithms to remain lightweight.

A naive but effective approach (see Figure 4A) divides the 2D space into grid cells of a certain size. For each grid cell, if it contains enough points, the entire cell can be marked as occupied. The final representation of the physical environment is then simply the union of all the occupied grid cells. Since each cell is represented by a square, the extracted shapes give a much simplified representation of a 2D projected point cloud for fast computation.

2.3.2 Adaptive Pseudo Boundaries

When the physical space is relatively large, or a complete mapping of the physical space cannot be obtained immediately, the representation of the physical space may contain no edges or only disconnected edges. In either case, the representation is incompatible with the presented RDW algorithms. We suggest a method named Adaptive Pseudo Boundaries (See Figure 4B): by placing imaginary "pseudo" boundaries around the user, RDW algorithms will redirect the users

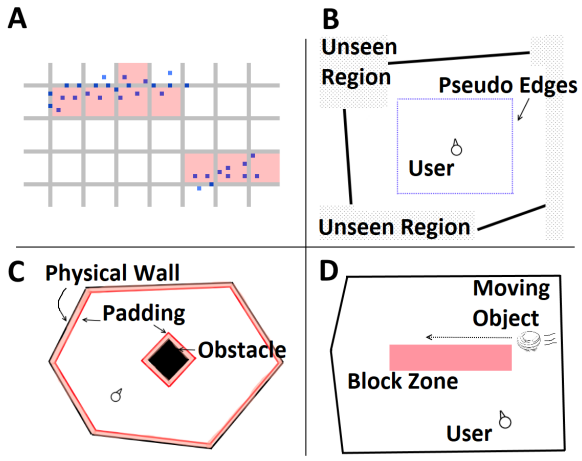

Figure 4: The four techniques for boundary representation described in Section 2.3. Figure A shows the bounding box representation of a point cloud. Blue dots represent locations of input points. Red area represents occupied grid cells. Figure B shows the use of pseudo edges in an incompletely scanned physical environment. Figure C shows an example of safe zone padding. Figure D shows an example of representing trajectory of a moving object with a block zone.

as much as possible in an attempt to keep them in the pseudo boundaries. This provides a temporary and lightweight solution when the actual physical environment has not yet been fully mapped. An interesting consideration is that the RDW algorithm does not need to perform a reset when it fails to fully redirect the users away from the pseudo boundaries. The area formed by them can be increased and decreased adaptively until it collides with physical boundaries in the environment. Such design also allows adaptive pseudo boundaries to be used as an acceleration technique for RDW algorithms, as it largely simplifies the representation of the physical environment.

2.3.3 Safe Zone Padding

User safety is a main concern for a view-occluded user who is walking freely in a physical environment. Even though we could assume that a RDW system is able to predict collision on time and trigger a reset warning, to ensure user safety, the system must take into account the reaction time needed by the users to notice the warning and stop walking. A protection mechanism can be implemented simply by padding every edge to create a safe zone in the representation of the physical space (See Figure 4C). In other words, for a given graph G, every outer boundary can be shifted a certain distance d in the direction of its normal that points to the interior, and every interior boundary can be shifted in the direction of its normal that points to the exterior. Locations of new vertices are given at intersections of the shifted edges.

2.3.4 Real-time Moving Obstacles

Suppose the physical environment contains dynamically moving objects, such as nearby people. If an oracle exists to provide an updated graph G at every frame to indicate movements of the objects, the presented RDW algorithms could redirect the users away from the moving objects by performing new analysis of the updated graph at a certain time step. An alternative approach that reduces computation overhead is to estimate a short-term trajectory of the moving object. The estimated trajectory can simply be represented as an elongated obstacle, or a "block zone", in G (See Figure 4D). RDW algorithms would then ensure that the user is redirected away from the entire trajectory, without having to frequently perform recomputation on updated graphs.

3 Conclusion

In this work, we focus on adapting RDW to work in irregularly shaped, dynamic physical environments. Grouping redirected walking algorithms into greedy algorithms and planning algorithms, we discuss the challenges that each type of algorithms face and present new algorithms and adaptation to support such physical environments. For greedy algorithms, we present two novel algorithms, Steer-to-Farthest and a Trapezoidal Road Map Algorithm, and discuss how environment updates can be handled. As for planning algorithms, we present techniques for cost computation, online planning, and misalignment handling so that planning algorithms in previous works can support irregularly shaped, dynamically updated physical environments. Lastly, we present techniques for altering and simplifying the physical environment representation. These techniques can be used to convert the representation of the physical environment to our defined mapping, increase user safety, improve performance of the presented RDW algorithms, and handle real-time moving obstacles. A limitation of this work is that the proposed methods are not formally evaluated. In the future, we plan to study and evaluate the effectiveness of these approaches in handling different types of physical settings.

Acknowledgments

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-1418060 and a Google VR Research Award. Any opinion, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] M. Azmandian, T. Grechkin, M. Bolas, and E. Suma. Automated path prediction for redirected walking using navigation meshes. In *3D User Interfaces (3DUI), 2016 IEEE Symposium on*, pp. 63–66. IEEE, 2016.

[2] M. De Berg, M. Van Kreveld, M. Overmars, and O. C. Schwarzkopf. Computational geometry. In *Computational geometry*, pp. 1–17. Springer, 2000.

[3] T. Nescher, Y.-Y. Huang, and A. Kunz. Planning redirection techniques for optimal free walking experience using model predictive control. In *3D User Interfaces (3DUI), 2014 IEEE Symposium on*, pp. 111–118. IEEE, 2014.

[4] T. Nescher and A. Kunz. Using head tracking data for robust short term path prediction of human locomotion. In *Transactions on Computational Science XVIII*, pp. 172–191. Springer, 2013.

[5] 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 *Virtual Reality Conference (VR), 2010 IEEE*, pp. 35–38. IEEE, 2010.

[6] S. Razzaque. *Redirected walking*. University of North Carolina at Chapel Hill, 2005.

[7] M. Shimrat. Algorithm 112: position of point relative to polygon. *Communications of the ACM*, 5(8):434, 1962.

[8] F. Steinicke, G. Bruder, J. Jerald, H. Frenz, and M. Lappe. Analyses of human sensitivity to redirected walking. In *Proceedings of the 2008 ACM symposium on Virtual reality software and technology*, pp. 149–156. ACM, 2008.

[9] Q. Sun, L.-Y. Wei, and A. Kaufman. Mapping virtual and physical reality. *ACM Transactions on Graphics (TOG)*, 35(4):64, 2016.

[10] M. Zank and A. Kunz. Optimized graph extraction and locomotion prediction for redirected walking. In *3D User Interfaces (3DUI), 2017 IEEE Symposium on*, pp. 120–129. IEEE, 2017.

[11] M. A. Zmuda, J. L. Wonser, E. R. Bachmann, and E. Hodgson. Optimizing constrained-environment redirected walking instructions using search techniques. *IEEE transactions on visualization and computer graphics*, 19(11):1872–1884, 2013.