Optimized Graph Extraction and Locomotion Prediction for Redirected Walking

Optimized Graph Extraction and Locomotion Prediction for Redirected Walking

Markus Zank* Andreas Kunz†

ICVR
ETH Zurich
Switzerland

Abstract

Redirected walking with advanced planners such as MPCRed or FORCE requires both knowledge about the virtual environment - mostly in the form of a skeleton graph representing the virtual environment - and a robust prediction of the user's actions. This paper presents methods for both parts and evaluates them with a number of test cases.

Since frame rate is crucial for a virtual reality application, the computationally heavy extraction and preprocessing of the skeleton graph is done offline while only parts directly linked to the user's behavior such as the prediction are done online. The prediction is done using a target-based long-term prediction and the targets are determined automatically and combined with targets predefined by the designer of the virtual environment.

The methods presented here provide a graph that is well suited for planning redirection and allows prediction techniques previously only demonstrated in studies to be applied to large scale virtual environments.

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

1 Introduction

Real walking has been well established as a navigation technique for large virtual environments. It is closest to the way humans navigate the real world and was shown to have many positive aspects such as improving the cognitive map users build of the environment [7, 13]. Moreover, the user is not required to learn a new navigation metaphor which is often a disadvantage of more abstract or semi-natural interfaces as discussed in [8]. However, real walking as a way of navigating virtual environments requires a large open space for the user to walk around in, which is probably the most limiting factor for use with large virtual environments. To reduce the required amount of space, Razzaque et al. introduced Redirected Walking [12]. Here, the user's movements are not translated to the virtual environment one-to-one, but instead an imperceptible mismatch is introduced. This mismatch causes the user's real and virtual path to deviate and allows infinite environments to be explored in a limited physical space by directing the user away from the boundaries of the available physical space.

In the following years, different types of mismatch were proposed including scaling the user's forward movement or rotations or warping a straight line into a circle (see [14] for a summary). For each of these techniques their potential and limits were researched in various environments and combinations to determine the maximum gain (≈strength of the mismatch) that could be applied before negatively affecting user experience. However, in many situations it is possible to reduce gains or apply different redirection techniques for a better user experience. To this end, it was proposed to include multiple techniques, allow a range of gains, and use a planning method to select the one to be applied at the moment [9, 18]. Zmuda et al.'s FORCE (Fully Optimized Redirected Walking for Constrained Environments) and Nescher et al.'s MPCRed (Redirected walking using Model Predictive Control) already showed a significant improvement in performance compared to a "steer to center" approach where the user is always redirected with a fixed gain towards the center of the physical space. However, to properly plan the redirection, it is necessary to know the user's future walking trajectory. Both Zmuda et al. and Nescher et al. use a manually defined skeleton graph with assigned probabilities for planning in their respective studies.

At the same time, prediction for redirected walking was done based on the user's current movement or facing direction [6, 14] or the recent history thereof [15]. These techniques all use the user's current position in the tracked space and make no use of knowledge of human locomotion or user's position in the virtual environment. At that time, Nitzsche et al. had already proposed a distinction of prediction methods in two categories [11]: Short-term prediction that uses a model of human locomotion to predict the user's future path based on current or recent tracking data; and long term-prediction that is based on the user's intention to move to a certain location. While all methods mentioned above fall in the first category, newer methods include a varying degree of knowledge about the environment. Nescher et al. [10] is a very early example that is functionally very similar to previous methods, but includes two distinct target locations and the user's smoothed movement direction is compared against them. Later on, a prediction based on models of human locomotion was proposed by Zank et al. [16]. They use the fact that human locomotion is stereotypical [5], which allows generic models of human locomotion to be combined with predefined target points to assign probabilities to individual targets. Other approaches also use distinct target locations in combination with eye tracking to estimate the user's intention [4, 17].

While these new approaches offer good prediction possibilities, they need additional information about the virtual environment as previously pointed out [2, 11]. To use these techniques in practice, it is necessary to generate both the skeleton graph required for the planning as well as the selection of locomotion targets for prediction automatically so that they can be used by a designer creating applications and without having in-depth knowledge of redirection and prediction algorithms. At the same time, it must be possible for the designer to add application-specific knowledge into the prediction such as knowledge on the user's task, since this knowledge could improve the prediction significantly.

Azmandian et al. proposed a first approach to retrieve possible future paths directly from the virtual environment [2]. Using a navigation mesh, they retrieve paths of fixed length from a graph

*e-mail: mzank@ethz.ch
†e-mail:kunz@iwf.mavt.ethz.ch

2017 IEEE Symposium on 3D User Interfaces (3DUI)
March 18-19, 2017, Los Angeles, CA, USA
978-1-5090-6715-2/17/$31.00 © 2017 IEEE

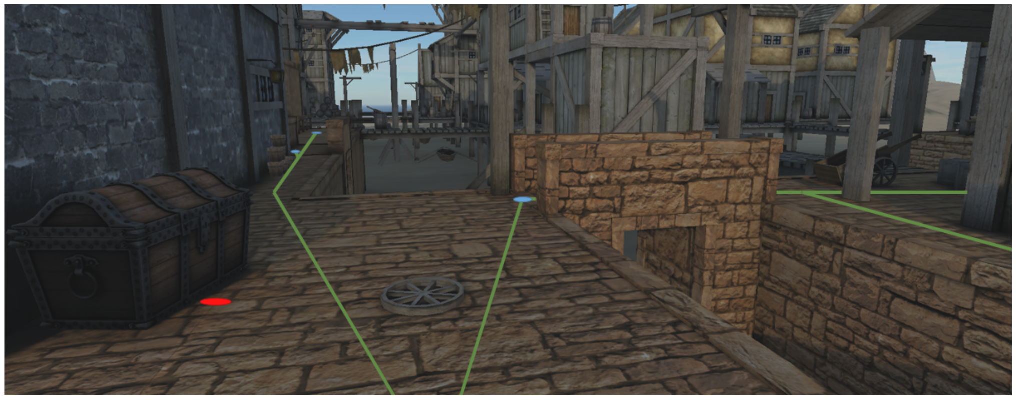


Figure 1: Example of the skeleton graph (green) with automatically generated waypoint targets (blue) and designer-added target (red).

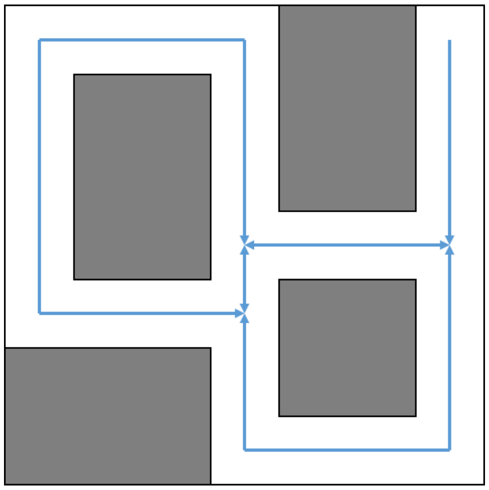


Figure 2: Example of a skeleton graph required for planning with MPCRed or FORCE.

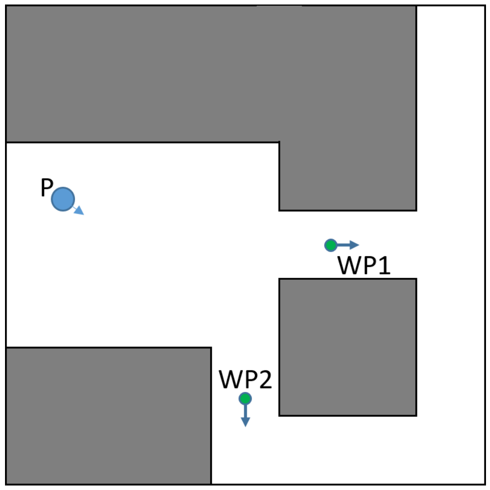


Figure 3: Example of waypoint targets and directions (WP) for prediction with user position *P*.

spanned by the centroids of the mesh's polygons, which can result in a zig-zagged path depending on the triangulation. However, both Zmuda et al.'s and Nescher et al.'s planning approaches consider kinks in the skeleton graph to be good locations to apply rotational gains, since they expect the user to execute a sharp turn. Therefore, by using this kind of triangulation of the environment, the redirection potential is overestimated.

In this paper, we present methods for retrieving a skeleton graph suitable for redirection planning and for finding potential waypoint targets for prediction. The graph-retrieval as well as preprocessing for the target-retrieval is done offline while the target-retrieval itself is done online depending on the user's position in the virtual world.

2 Requirements and Assumptions

The proposed algorithm is split in an online and an offline part. In order to improve real-time performance, as much computation as possible is done offline and only computations directly connected to the user's actions should be conducted online.

The following list summarizes the requirements that have to be met by such an algorithm:

1. Automatically generate a skeleton graph that represents the walkable area (WA) in the virtual environment and can be used for redirection planning (see Figure 2);
2. The skeleton graph follows all paths of the environment, but never intersects the boundary of the WA;
3. The skeleton graph has no kinks where the user is not expected to turn in order to minimize the overestimation the redirection potential;
4. Waypoint targets are automatically generated online for human locomotion prediction (see Figure 3);
5. The designer of the virtual environment can add custom target points when building the environment that are combined with the automatically generated waypoint targets.

Figure 1 shows an example of a virtual environment with a skeleton graph (green lines), waypoint targets (blue disks) and a designer added target (red disk) at a location where the designer expects the user to go based on the task.

When talking about "prediction" we follow the convention used in [4, 10, 11] and are referring to an online estimation process that returns a probability distribution over a set of possible user actions. Each probability indicates how likely it is that the user will take the respective action. A locomotion target is a point and direction that corresponds to a location where a user might deliberately walk to. The addition of a direction to the target is crucial to ensure that a path model correctly represents the user's direction of movement at

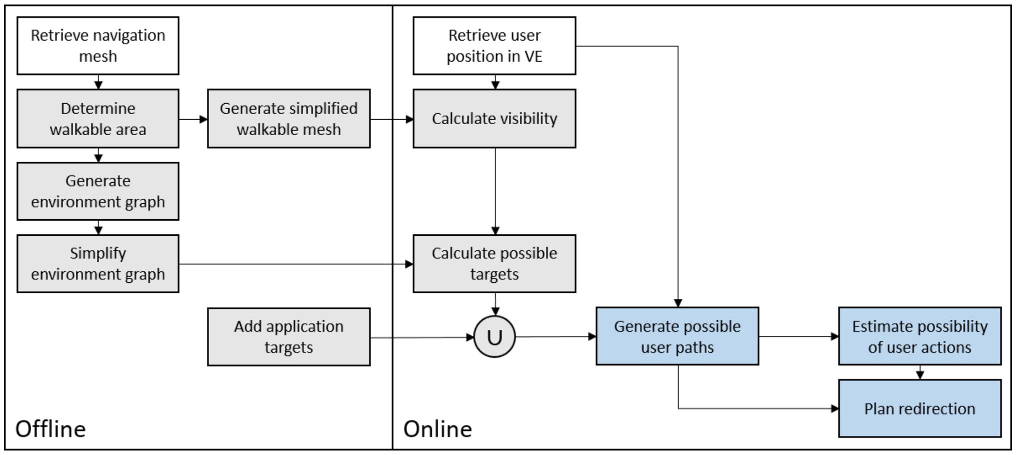

Retrieve navigation mesh
Determine walkable area
Generate simplified walkable mesh
Generate environment graph
Simplify environment graph
Add application targets
Offline
Retrieve user position in VE
Calculate visibility
Calculate possible targets
U
Generate possible user paths
Estimate possibility of user actions
Plan redirection
Online

Figure 4: Outline of the overall procedure. The gray boxes represent steps presented in this paper, the blue boxes represent steps that have been published previously such as the planning of the redirection and the prediction of user actions. White boxes represent data that is provided directly be a sensor or the game engine.

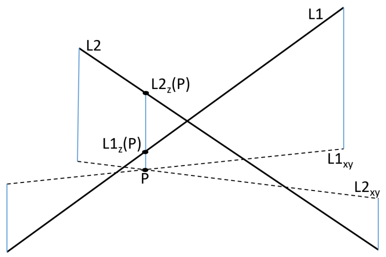


Figure 5: Intersection of L1 and L2

end of the path like for the waypoints in Figure 3 or for a picture hanging on a wall. Given a starting position and a target, locomotion models are used to approximate the path a human is likely to take when walking from the starting position to the target.

2.1 Definitions

Even though the presented implementation was done in the Unity game engine, the math presented here assumes a right-handed coordinate system with the z-axis pointing up. The following algorithms use points in 3D space and lines or edges that are defined only be the two points they connect. Triangles are the only faces allowed and a triangle is defined by three edges. P_x is defined to be the x component of point P and $L1_{xy}$ is the projection of line L1 onto the xy plane.

Immersive virtual environments are in 3D space, however human locomotion takes place in a 2D plane. This allows using much simpler and faster 2D algorithms for most calculations. In order to still be able to apply the proposed procedure to arbitrary virtual environments, including multi-story environments, we use a special implementation of 2D geometry with layers: All differences in z-position between two points are ignored if the difference is below a certain value z_{res}. This means for example (illustrated in Figure 5) that two lines $L1$ and $L2$ intersect iff:

$$P = L1_{xy} \cap L2_{xy} \neq \emptyset \quad \text{and} \quad |L1_z(P) - L2_z(P)| \leq z_{res} \qquad (1)$$

If $|L1_z(P) - L2_z(P)| > z_{res}$, $L1$ and $L2$ do not interact in any way, meaning they cannot intersect or occlude points. Since the method is meant for human locomotion, it can be assumed that the user moves at a natural walking speed, in a continuous fashion (no discontinuities) and within the boundaries of areas defined as walkable.

3 Algorithm

Figure 4 outlines the methods presented in this paper and how they are used together with existing redirected walking algorithms and prediction approaches. The purpose of the methods presented here is to provide the predictor and the redirection planner with possible user paths and the following sections will explain step by step how these paths can be generated from only two inputs, namely the navigation mesh of the virtual environment and the user's position. Figures 6,9,12,16, and 17 at the beginning of each section show which part of the overall pipeline shown in figure 4 the respective section explains.

The navigation mesh represents the walkable area in a scene with a triangle mesh and is meant for simplified agent navigation. The procedure presented here is based on this triangle mesh, making the concept applicable to any engine that can provide such a mesh. The offline part of the algorithm is implemented as a Unity editor script, so that a designer can add his own points-of-interest to the list of potential targets and run the pre-calculations after finishing the environment. The online part is depending to the user's location and is updated as his position changes during run-time. No third party libraries were used except for Unity's integrated navigation mesh generation.

3.1 Walkable Mesh Preparation

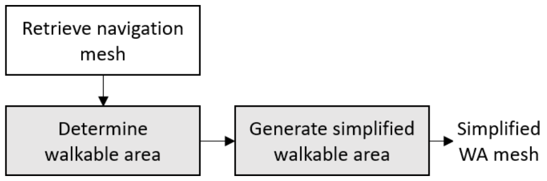


Figure 6: Walkable Mesh Preparation. Generate simplified walkable area from navigation mesh

In a first step, the navigation mesh needs to be prepared. When the Unity navigation mesh is created, it contains all the areas Unity considers walkable based on slope and area, even though they might be disconnected from each other. However, this might not represent the area that is actually reachable by a human by walking. To select the correct submesh, the designer interactively selects one vertex

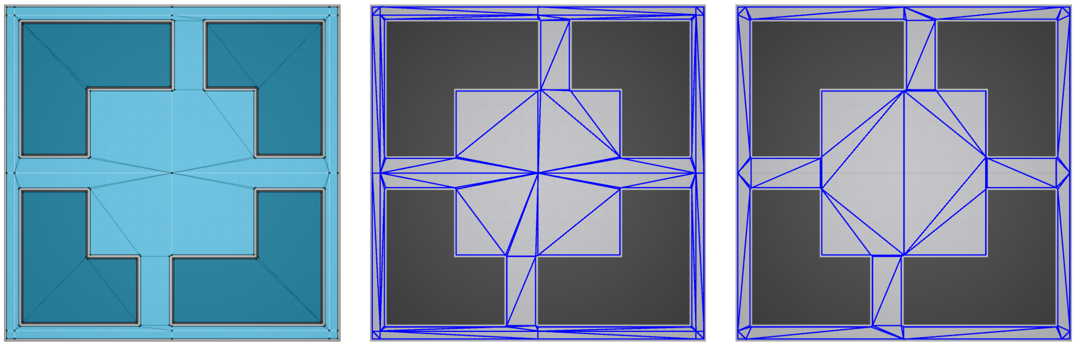

(a) Original navigation mesh (b) Retrieved triangle mesh (c) Removed inside vertices

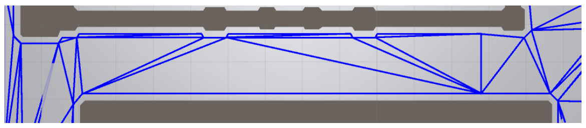
Figure 7: First steps of mesh processing. Starting with the Unity navigation mesh a, retrieved triangle mesh b, and after removing inside vertices and the Delaunay triangulation c.

in the mesh and all connected triangles are included in the Walkable Mesh (WM). The mesh is transformed into a custom WalkableMesh structure that is optimized towards fast operations rather than Unity's space-efficient mesh representation. At the same time, any vertices that are not on the outside edge of the mesh are removed and the mesh is Delaunay triangulated. Therefore any vertex has now exactly two connected outside edges that coincide with the outside border of the WM and every triangle has at least one outside edge. Figure 7 shows the steps applied to an example environment.

Since Unity generates a mesh for navigation of agents, it represents the environment relatively accurately. However, this accuracy is not required and will affect performance negatively since the runtime of most of the algorithms used afterwards scale with the number of edges in this mesh. Therefore the WM is simplified. First, vertices closer than 0.3 meters are collapsed into one. Then any vertex is removed where the outside edges have a dot product larger than 0.95. This is done starting at the outside vertex with the largest dot product (Algorithm 1). This value was chosen as a compromise between conserving the large-scale structure and removing unnecessary details. The resulting outline of the walkable mesh shown in Figure 8 has a reduced number of vertices but still accurately represents the environment on a scale relevant to human locomotion.

Algorithm 1 Simplify the outside polygon

```
Polygon $poly$ ← CCW outside polygon of the WM
SortedDictionary<float, vertex> $d$
for all vertex $v$ in $poly$ do
    add $v$ and the dot product of $v$'s neighboring edges to $d$
end for
sort $d$ by the dot product in descending order
while $d(0).key > 0.95$ do
    $n1, n2$ ← neighbors of $d(0).value$
    remove $d(0)$ from d
    update dot product of $n1$ and $n2$
end while
```


(a) Original walkable mesh. The depicted area is approximately 15x3 meters in size.

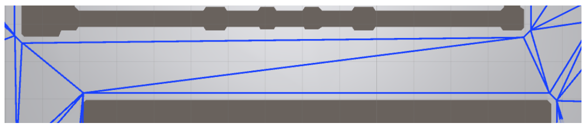

(b) Mesh without details

Figure 8: Original WM and and simplified mesh.

3.2 Skeleton Graph

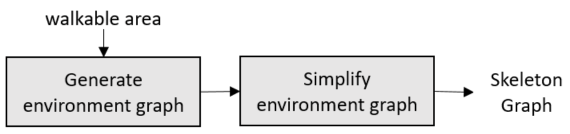


Figure 9: Generate Skeleton Graph from original Walkable Area

A common feature of prediction and planning of redirection is that there is a skeleton graph representation of the virtual environment. It is necessary to have a graph that represents the skeleton of the environment and contains all paths and junctions in the environment. Therefore, it is required that this graph follows corridors in the middle and in a smooth line. In open areas, it is expected to be curved smoothly and split and join close to narrow points and junctions.

The Voronoi diagram is a partition of space where a cell contains the area of space that is closer to a certain point than to any other point in the set. Reversely, this means that all the points on the edge of any cell have equal distance to all closest points of which there are at least two. This makes it well suited as a starting point for a skeleton graph of the virtual environment if the vertices of the WM are used as points. For an extended introduction refer to [1].

The Voronoi diagram (VD) can easily be generated from the Delaunay triangulation, due to its property that no other vertex must be inside a triangles circumcircle. The VD can therefore be generated by simply connecting the circumcircle centers of neighboring triangles.

When using circumcircles to generate this path, there is a risk that the center lies outside of the walkable area. This happens mainly for triangles with a single outside edge where this edge is significantly longer than one of other two edges. To overcome this, the maximum length for outside edges is limited and longer edges are split and re-triangulated before creating the VD.

This results in a Voronoi Diagram that consists of many short edges that increase the run-time of the online parts. Therefore it is simplified. However, the smoothing applied to the outside edges of the Walkable Mesh cannot be applied yet, because the unwanted artifacts include very large angles in corridors like shown in figure 10. Instead, the graph is smoothed before removing vertices.

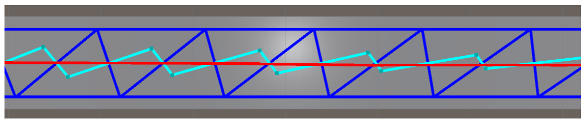


Figure 10: Original Voronoi Diagram (cyan) calculated from the mesh (blue) and the resulting simplified graph (red).

This is achieved by fitting a Bézier spline for every vertex with exactly two connecting edges that is supported by the vertex and the neighboring two vertices in both directions along the graph (Figure 11a). In case that one of the supporting vertices has more than two neighbors, no additional vertices are included in this direction (Figure 11b). Vertices with only one or more than two neighbours are not moved. This ensures that ends are not modified and junctions also do not move. After this step, vertices with small angles are removed in the same way as for the outside edges. In a last step, vertices closer than 1m to each other are collapsed into one vertex.

This results in a graph that represents the structure of the virtual environment with a good tradeoff between accuracy and number of vertices and edges. In case MPCRed should be used, an additional step is required to replace corners by arc segments [9].

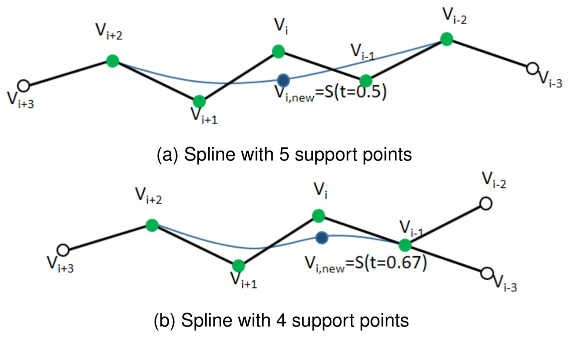


Figure 11: Examples for smoothing spline for V_i. Green vertices are used as support for the spline S, white vertices are not used. $V_{i,\text{new}}$ is the resulting new position of V_i

3.3 Visibility Polygon

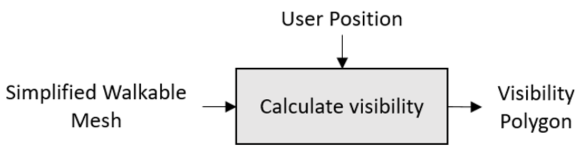
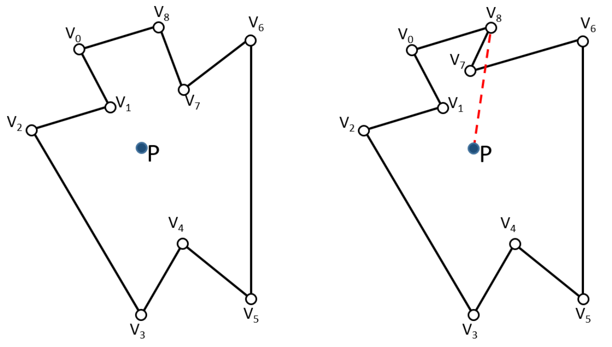


Figure 12: Calculate Visibility Polygon from the simplified walkable mesh and current user position

During the online phase, it is crucial to know what area the user can see to determine potential locomotion targets. However, it is not required to know this in 3D space, but rather in the 2D representation explained above. At this point, the user's orientation is not considered and a 360° visibility polygon is calculated. This makes the calculation independent of the HMD used and does not ignore targets that are slightly outside of the field of view.

The visibility calculation starts from the WalkableMesh triangle the user is currently located in. This is done because the vertices of this triangle are the only ones guaranteed to be visible from the user's position. In the first frame, the triangle is determined by brute force, but in every following frame, the triangles are searched starting with the triangle the user was in during the last update and then checking neighboring triangles, if he is not there anymore. This will increase the search performance greatly, since the triangles are large compared to the typical human walking speed. This means that in most cases, there are only one or three triangles that need to be checked, before the user is found (see Algorithm 2). This is because either the user is still in the same triangle as before or one of the two neighboring triangles (every triangle has at least one outside edge, because all inside vertices were removed).

The result of the visibility calculation will be a star-shaped polygon. This is because a star-shaped polygon is defined to be a polygon where there exists a point inside from which the entire polygon can be seen, which in this case is the user's position. The fact that the resulting polygon will be star-shaped allows certain simplifications. All vertices will be connected by a single line of edges and there are no disconnected parts (in contrast to a polygon with holes such as the outline of the WM). From this, it follows, that given the user position P and the vertices of the polygon sorted CCW $V_1, \ldots V_n$ the angle $\alpha_i = \tan(\frac{V_{i,y}-P_y}{V_{i,x}-P_x})$ will be increasing ($\alpha_{i+1} \geq \alpha_i \forall i = [1,n]$).



(a) Star-shaped polygon. The entire polygon is visible from P

(b) V_8 is occluded and not visible from P]

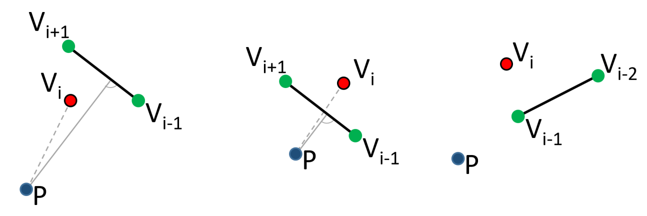
Figure 13: Examples of polygons with ordered vertices. Note that in the star-shaped case α is non-decreasing whereas in case b) $\alpha_7 > \alpha_8$.

It therefore makes sense to sort all vertices in WM counter-clockwise around the user. Now, every vertex that is not occluded will be visible to the user and other edges in WM are the only thing that can occlude a vertex. However, naively checking occlusion for every edge for every vertex has a run-time of $\mathcal{O}(n^2)$. Instead, the properties of star-shaped polygons described above allow a preselection of edges E that can potentially occlude a vertex V from the user at position P (see Figure 14 for examples of these cases):

- A vertex can only be occluded by edges that have at least one point closer to P than the distance from P to V. For this check, the orthogonal distance of P to the edge can be used, because the orthogonal distance is the shortest distance to P and if no point on the edge is closer to P than V, the edge cannot occlude V.
- V and P must be on opposite sides of E.
- A vertex V can only be occluded by an edge E that has one vertex in clockwise and one in counter-clockwise direction of V

Therefore, by sorting all vertices in an angular fashion and all edges by orthogonal distance to P, the visibility polygon can be calculated more efficiently.

The resulting algorithm is summarized in Algorithm 3. The algorithm starts at one of the vertices of the triangle the user is currently in. From this vertex, the list of sorted vertices is traversed CCW and the visibility of each vertex is checked until a visibile vertex is



(a) V_i cannot be occluded by (V_{i-1}, V_{i+1}) (b) V_i can be occluded by (V_{i-1}, V_{i+1}) (c) V_i cannot be occluded by (V_{i-2}, V_{i-1})

Figure 14: Cases of occlusion of V_i from user position P depending on angle and side relative to an edge.

found. This vertex becomes the new current vertex and is added to the list of visible vertices.

However, this is still scales poorly for larger environments since creating a sorted list alone takes $\mathcal{O}(n \cdot \log(n))$ time. The run-time is already reduced by simplifying the walkable mesh as described above, but certain additional precalculations can be made to further improve online performance (see Algorithm 4).

For the visibility calculation, it is important to know which vertices of the WM are visible to the user and while the user moves, the mesh vertices do not. Since a point V in the WM can be seen from a point P iff P is inside the visibility polygon spanned by V, we can precalculate the area from which any vertex can be seen. Therefore, a visibility polygon is calculated from the position of each vertex in the WM during the offline phase and is stored for later use. During the online phase, we only need to check if P is inside the precalculated visibility polygon of each vertex. If he is, he can see the respective vertex.

However, the runtime still scales with the overall number of vertices in WM. Therefore, we also calculate ahead of time which vertices can potentially be seen from a certain triangle, which is done by intersecting the triangle with the already calculated visibility polygons. If the intersection is non-empty, there exists a point in the triangle from which the vertex can be seen. This list of potentially visible vertices can be retrieved at run-time with the knowledge of which triangle currently contains the user. This reduces the run-time of the algorithm from $\mathcal{O}(n^2)$ to $\mathcal{O}(k^2)$ where n is the total number of vertices in the WM and k is the total number of vertices that can be seen from any point within the user triangle T. For any virtual environments with limited visibility, k can be expected to be much smaller than n.

This provides a list of all vertices in the WM that are visible from the user's current position. To get the required star-shaped polygon, these vertices need to be connected as summarized in Algorithm 5. Since the resulting list of vertices is already sorted counterclockwise around the user, the algorithm can go through the list and try to connect consecutive vertices. There are two cases, either two consecutive vertices in *polygonPoints* are connected by an existing outside edge in the WM (see Figure 15a), in which case the connecting edge will be part of the visibility polygon, or they are not. This is the case if one or both vertices of an edge are occluded by other vertices (see Figures 15b and 15c). In this case, an additional edge needs to be created (this edge is only part of the visibility polygon, not of the WM). This is done by casting a ray from the user position through the current and the following vertex and intersecting them with the boundary of the WM. Then the intersections are analyzed to distinguish the cases shown in Figures 15b and 15c. Depending on the case, the required vertices and edges are added to the polygon.

Algorithm 2 Find user triangle

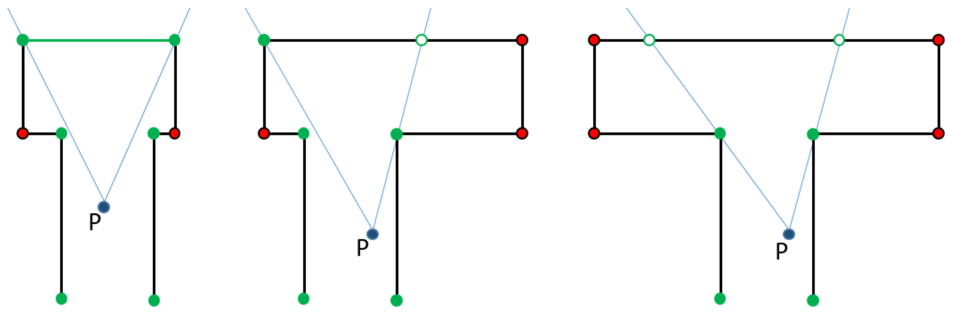
```
define list of triangles $toBeChecked$
define list of triangles $wasChecked$
if $LastPlayerTriangle \neq \emptyset$ then
    Add $LastPlayerTriangle$ to $toBeChecked$
else
    Add all triangles in WM to $toBeChecked$
end if
while $toBeChecked$ is not empty do
    if $toBeChecked(0)$ contains user then
        return $toBeChecked(0)$
    else
        Add $toBeChecked(0)$ to $wasChecked$
        for all neighbours $n$ of $toBeChecked(0)$ do
            if $wasChecked$ does not contain $n$ then
                Add $n$ to $toBeChecked$
            end if
        end for
        Remove $toBeChecked(0)$ from $toBeChecked$
    end if
end while
return $\emptyset$
```

Algorithm 3 Find visibile vertices (without precalculations)

```
define list of vertices $visibility$
find triangle $T$ in WM containing user position $P$
list of vertices $vertByAngle \leftarrow$ all vertices in WM
sort $vertByAngle$ in counter clockwise order around $P$
list of vertices $vertByDist \leftarrow$ all vertices in WM
sort $vertByDist$ by distance to $P$
$V \leftarrow$ any vertex in $T$
while $V$ is not the original vertex again do
    $V_{next} \leftarrow$ Find vertex in $vertByAngle$ that has smallest angle in
    CCW direction
    if $V_{next}$ is not hidden from $P$ by any edge then
        add $V$ to $visibility$
        $V \leftarrow V_{next}$
    end if
end while
```

Algorithm 4 Find visibile vertices (with precalculations)

```
define list of vertices $visibility$
find triangle $T$ in WM containing user position $P$
list of vertices $potentialVert \leftarrow$ retrieve potentially visible ver-
tices from dictionary
for all vertices $V$ in $potentialVert$ do
    if $P \in visibilityPolygon(V)$ then
        add $V$ to $visibility$
    end if
end for
```



(a) Common edge (b) Edge with one visible vertex (c) Edge without visible vertices

Figure 15: Three cases of connecting visible vertices. The user is located at position P, green vertices are visible from his current position, red vertices are not.a Connecting two vertices with common outside edge. b Connecting two vertices with one partial outside edge and one inside edge. c Connecting two vertices with one partial outside edge and two inside edges.

Algorithm 5 Generate visibility polygon from vertices

```
define list of vertices $polygonPoints$
for $i = 0$ to number of vertices in $visibility$ do
    add $visibility(i)$ to $polygonPoints$
    if WM does not contain edge $visibility(i)$ to $visibility(i+1)$ then
        $I1 \leftarrow$ intersections of ray from $P$ through $visibility(i)$ with the outside edges of WM
        $I2 \leftarrow$ intersections of ray from $P$ through $visibility(i)$ with the outside edges of WM
        Determine intersection cases and add $I1$ and/or $I2$ to $polygonPoints$
    end if
end for
convert $polygonPoints$ to polygon
```

3.4 Target Locations

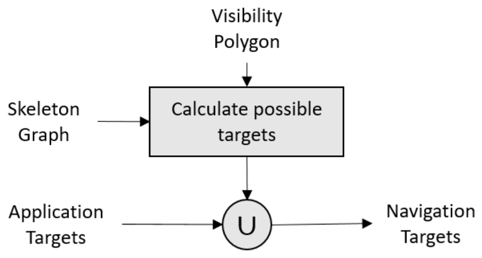


Figure 16: Determine navigation targets from visibility and skeleton graph and combine the result with set of predefined application targets.

For a locomotion prediction as discussed by Zank et al. [16] it is necessary to have a set of target points for locomotion in order to predict the user's intention. There are two kinds of targets that need to be distinguished: application targets and waypoint targets. Application targets are actual objects in the virtual environment defined by the application or the task the user has in the virtual environment. What exactly these objects are depends on the application but can include things like a treasure chest or character in a game or a terminal the user needs to operate in a training scenario. These objects have to be tagged by a designer or could potentially be learned later on from recorded user data in a data-driven approach. The waypoint targets on the other hand do not correspond to an object, instead they are pieces of geometry that shape the user's path such as doors, choke points or junctions where a user has to pass through to get to certain areas of the environment.

These target locations are used for prediction and describe locations the user is expected to walk to. In the past, these points were predefined manually for the small scale environments used in studies. However, this is not feasible for a large scale environments. Assuming that the user has no additional knowledge about the environment, it is likely that he plans out his own path to the limit of what he can see right now. This means that the intersections of the user's visibility polygon with the skeleton graph should closely match the points the user would plan his path to. In addition to the resulting target points, all predefined application targets inside the user's visibility polygon are added to the set of potential locomotion targets. This results in a list of points the user could potentially be moving to and redirection can be planned from the current user position to these points.

3.5 Prediction

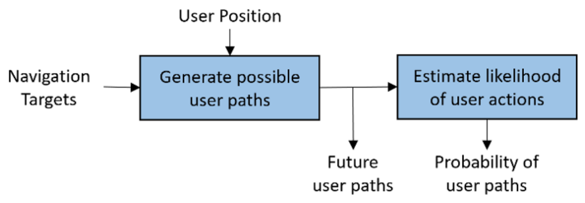

User Position
Navigation Targets
Generate possible user paths
Estimate likelihood of user actions
Future user paths
Probability of user paths

Figure 17: User model to generate future user paths and estimate their likelihood for planning redirection.

However, since in reality the user will only go to one of these targets, the planning is not optimal. Estimating the likelihood of the user going to each target allows to discount unlikely targets when planning the redirection and will improve the user experience for the more likely ones.

After more simple approaches of predicting a user's locomotion like [11, 14, 15], which are mainly based on extrapolating the user's current or recent movement, there is recent work by Gandrud et al. [4] and Zank et al. [16, 17]. Here, eye tracking or models of human locomotion are used to provide a more sophisticated prediction of a user's destination that incorporates knowledge of human locomotion as well as the Virtual Environment into the prediction. They are also able to provide a probability distribution over the targets instead of only giving a future direction of movement.

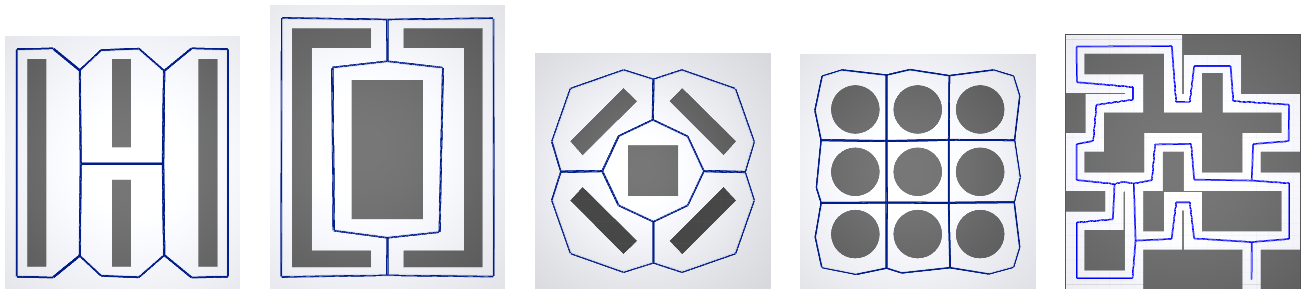

(a) Mesh A1 (b) Mesh A2 (c) Mesh A3 (d) Mesh A4 (e) Mesh A5

Figure 18: Resulting skeleton graph for artificial evaluation meshes A1-A5.

These approaches need targets that are fixed in space while the user walks, but determining the targets every frame using the method described above would result in targets that are constantly moving. However, the planning is also not updated every frame and a new prediction is only required when planning. After planning redirection, the selected technique is usually applied for a certain time, in the case of Nescher et al. [9] for 2.5 seconds. While this is also limited by computational time, it is also not required to re-plan redirection for every frame. Given a good prediction, the selected redirection should remain valid until the next planning step is completed. Therefore, new targets can be determined when planning is finished and movement behavior can be observed until the next planning cycle is executed at which point the probability distribution is retrieved and new targets are determined. Since the planning cycle is short which limits the distance the user can travel before the next planning cycle and we already consider a 360° field of view, it can be expected that this does not heavily impact the overall performance. This is due to the fact that the time the user needs to reach the border of his current visibility is much larger and rapid turns do not matter, because targets behind are already considered due to the 360° field of view.

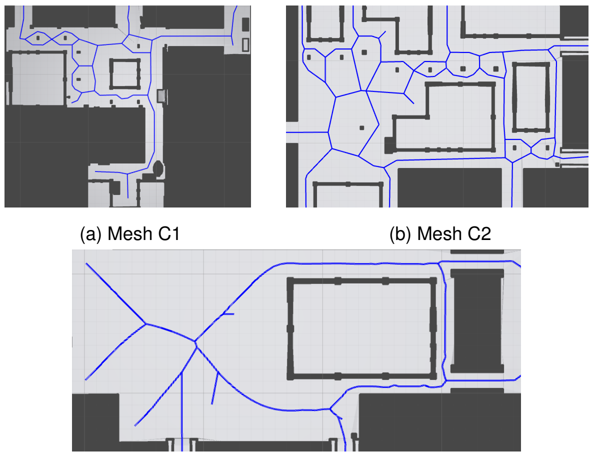
4 Evaluation

In the following section, the presented algorithms will be evaluated with both artificial and realistic test environments. The artificial environments demonstrate the overall performance in simple scenarios, while the realistic test cases show that the methods also work for more complex virtual environments typically encountered in real world applications.

The artificial environments A1-A4 are taken from [2] for better comparability. The environment A5 is the study environment [9]. The application environments C1-C3 are parts of a larger demonstration scene of a small village on a lake. It consists of houses and canals with walkways and open spaces in between and is shown in Figure 1. It was chosen for offering both narrow, maze-like structures as well as open spaces and, while only small segments are shown, the methods were applied to the whole environment.

Figures 18 and 19 show the skeleton graph extracted by the presented method. Our method results in a very small number of lines that represents the structure of the environment. They are mostly in the middle of corridors in maze like environments and are very similar to what Zmuda and Nescher propose in their respective papers. The only difference is that Nescher uses arc segments instead of corners, but this could easily be added. One issue is the creation of small tails that remain, especially in corners of the application meshes. However, removing them aggressively is not a solution since there are situations where they actually correspond to an important piece of the environment like seen in Figure 18e.

Figure 20 shows the resulting visibility polygon from the user position marked with the blue dot. Figure 21 shows the visibility


(a) Mesh C1 (b) Mesh C2

(c) Mesh C3

Figure 19: Resulting skeleton graph for application evaluation meshes C1-C3.

and resulting estimated paths for the example C3. The model used here is the one by Cirio et al. [3]. It can be seen that the paths indeed lead to reasonable target locations and they can be used to plan redirection.

4.1 Run-times

In the following section, the run-times for the online parts of the algorithm are evaluated. The evaluation was conducted on a XMG U506 laptop with an Intel i7-6700 CPU. The application environment is used and the processed walkable mesh contains 351 vertices, the skeleton graph contains 211 edges. The user position was

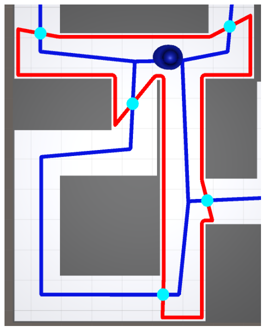


Figure 20: Intersections of skeleton graph (dark blue) and visibility (red) of user (blue dot) and resulting target locations in teal.

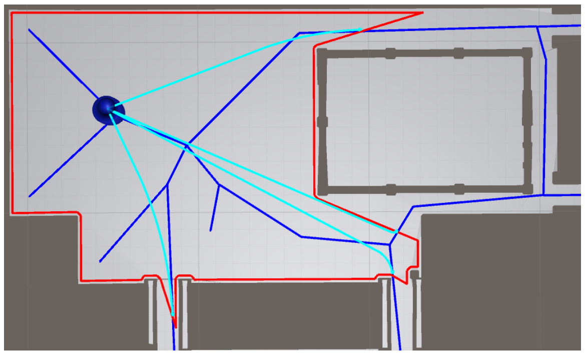


Figure 21: Mesh C3 with visibility (red) and estimated locomotion paths for four waypoint targets. The possible user paths that were generated using Cirio et al's path model [3].

driven by artificial data that moves the user along a predefined path of 142 meters length through the environment at a speed on 1m/s. Along the path, the calculation time for the visibility calculation and intersection with the skeleton graph is recorded. The average time for the visibility calculation is 19ms, Figure 22 shows the calculation time along the path as well as the resulting number of edges in the visibility polygon.

4.2 Non-static Environments

Many of the calculations presented here are done offline and assume a static environment. However, many virtual environments are not static, instead many parts of the environment can be moved by users or even move by themselves. There are multiple categories of moving objects that have a different impact on the performance and applicability of the presented methods. The first group contains small moveable objects like chairs, books, etc. These objects are too small to influence anything and can be ignored easily since they would be removed by the simplification of the WM anyways. The second group are doors. On the first glace one might think that a closed door should cut the skeleton graph at this location and block visibility. However, as long as the user can open the door this would just require the door to be added as an application target. Therefore, the result stays almost the same, only the path model will be slightly off because the user needs to stop and open the door before passing through. It is also possible that the environment contains agents that move on their own. They will mainly influence the user's locomotion path which can easily be taken into account with more sophisticated path models. The only case that will significantly influence the planning is a moving object that is significantly larger than the user so that it actually changes his paths and cannot be moved by him so he has to plan around it. However, this would be a change that influences the whole virtual environment in a significant way and can therefore be expected to be uncommon and infrequent. For this reason, we recommend to just generate two (or even more) skeleton graphs and visibility precalculations and swap them as needed during run-time.

5 Conclusion

In this paper, we present a method for automatically preparing virtual environments for redirection with planning by generating a skeleton graph from the navigation mesh provided by the respective game engine. Based on examples, we show that the method generates suitable results in various environments. In addition, a method for automatically generating waypoint targets at run-time is presented. This allows model-based prediction methods to be used without manually adding target positions, which makes it possible to use these methods easily. However, it still is unclear how much the model-based prediction will improve the user experience in terms of reduced number of resets and amount of gains applied. Using a model based prediction will also give the planner a better estimate of the true redirection potential that can be expected. However, redirection will still be necessary to plan beyond the limits of these prediction approaches which means it is still necessary to use predefined probabilities along the graph for planning. Therefore, it might be useful to record data movement data from users in the virtual environment. If the virtual environment is an application where the layout does not change between users like a virtual exhibition, probabilities can be learned from previous users and assigned to the edges in the skeleton graph which allows planning with better probabilities beyond the user's field of view.

Acknowledgements

The authors wish to thank the Swiss National Science Foundation, project number 205121 153243, for funding this work.

References

[1] F. Aurenhammer. Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, 23(3):345–405, 1991.

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

[3] G. Cirio, A.-H. Olivier, M. Marchal, and J. Pettre. Kinematic evaluation of virtual walking trajectories. *IEEE Transactions on Visualization and Computer Graphics*, 19(4):671–680, 2013.

[4] J. Gandrud and V. Interrante. Predicting destination using head orientation and gaze direction during locomotion in vr. In *Proceedings of the ACM Symposium on Applied Perception*, pages 31–38. ACM, 2016.

[5] H. Hicheur, Q.-C. Pham, G. Arechavaleta, J.-P. Laumond, and A. Berthoz. The formation of trajectories during goal-oriented locomotion in humans. i. a stereotyped behaviour. *European Journal of Neuroscience*, 26(8):2376–2390, 2007.

[6] V. Interrante, B. Ries, and L. Anderson. Seven league boots: A new metaphor for augmented locomotion through moderately large scale immersive virtual environments. In *IEEE Symposium on 3D User Interfaces*, pages 167–170. IEEE, 2007.

[7] F. Larrue, H. Sauzeon, G. Wallet, D. Foloppe, J.-R. Cazalets, C. Gross, and B. N'Kaoua. Influence of body-centered information on the transfer of spatial learning from a virtual to a real environment. *Journal of Cognitive Psychology*, 26(8):906–918, 2014.

[8] M. Nabiyouni, A. Saktheeswaran, D. A. Bowman, and A. Karanth. Comparing the performance of natural, semi-natural, and non-natural locomotion techniques in virtual reality. *IEEE Symposium on 3D User Interfaces*, pages 3–10, 2015.

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

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

[11] N. Nitzsche, U. D. Hanebeck, and G. Schmidt. Motion compression for telepresent walking in large target environments. *Presence: Teleoperators and Virtual Environments*, 13(1):44–60, 2004.

[12] S. Razzaque, Z. Kohn, and M. C. Whitton. Redirected walking. In *Proceedings of EUROGRAPHICS*, volume 9, pages 105–106, 2001.

[13] R. A. Ruddle, E. Volkova, and H. H. Bülthoff. Walking improves your cognitive map in environments that are large-scale and large in extent. *TOCHI '11: ACM Transactions on Computer-Human Interaction*, 18(2):10:1–10:20, July 2011.

[14] F. Steinicke, G. Bruder, L. Kohli, J. Jerald, and K. Hinrichs. Taxonomy and implementation of redirection techniques for ubiquitous passive haptic feedback. In *International Conference on Cyberworlds*, pages 217–223. IEEE, 2008.

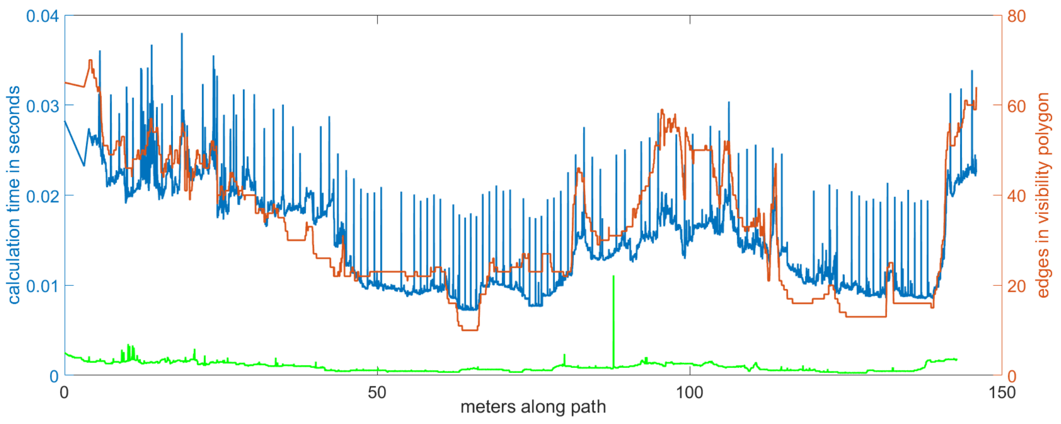


Figure 22: Runtime of the visibility calculation (blue), intersection with skeleton graph (green). The dashed red line shows the resulting number of edges in the visibility polygon.

[15] J. Su. Motion compression for telepresence locomotion. *Presence: Teleoperators and Virtual Environments*, 16(4):385–398, 2007.

[16] M. Zank and A. Kunz. Using locomotion models for estimating walking targets in immersive virtual environments. In *International Conference on Cyberworlds*, pages 229–236, 2015.

[17] M. Zank and A. Kunz. Eye tracking for locomotion prediction in redirected walking. In *IEEE Symposium on 3D User Interfaces*, pages 49–58. IEEE, 2016.

[18] 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.