Journal of Artificial Intelligence Research 74 (2022) 917-955 Submitted 12/2021; published 06/2022

Path Counting for Grid-Based Navigation

Rhys Goldstein RHYS.GOLDSTEIN@AUTODESK.COM
Kean Walmsley KEAN.WALMSLEY@AUTODESK.COM
Jacobo Bibliowicz JACKY.BIBLIOWICZ@AUTODESK.COM
Alexander Tessier ALEX.TESSIER@AUTODESK.COM
Autodesk Research, Toronto, ON, Canada

Simon Breslav SIMON.BRESLAV@TRAX.CO
Azam Khan AZAM.KHAN@TRAX.CO
Trax.Co, Toronto, ON, Canada

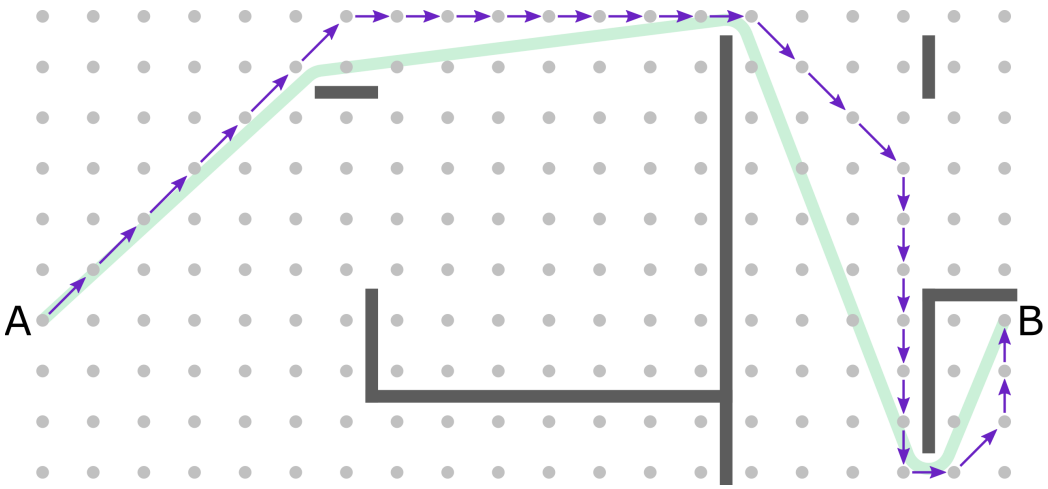
Abstract

Counting the number of shortest paths on a grid is a simple procedure with close ties to Pascal's triangle. We show how path counting can be used to select relatively direct grid paths for AI-related applications involving navigation through spatial environments. Typical implementations of Dijkstra's algorithm and A* prioritize grid moves in an arbitrary manner, producing paths which stray conspicuously far from line-of-sight trajectories. We find that by counting the number of paths which traverse each vertex, then selecting the vertices with the highest counts, one obtains a path that is reasonably direct in practice and can be improved by refining the grid resolution. Central Dijkstra and Central A* are introduced as the basic methods for computing these central grid paths. Theoretical analysis reveals that the proposed grid-based navigation approach is related to an existing grid-based visibility approach, and establishes that central grid paths converge on clear sightlines as the grid spacing approaches zero. A more general property, that central paths converge on direct paths, is formulated as a conjecture.

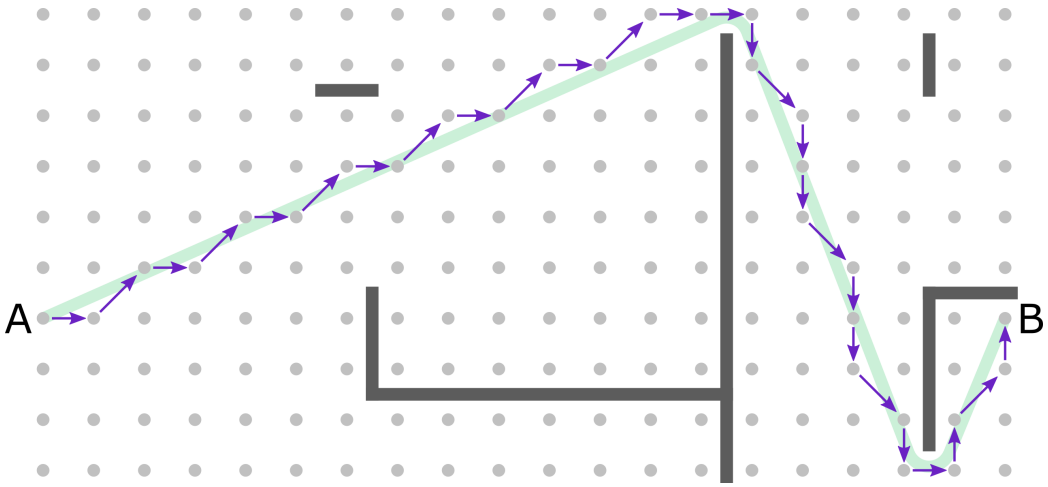
1. Introduction

Computational methods for navigation are essential to many AI-related applications involving spatial environments, particularly in the domains of video games (Botea et al., 2013), robotics (Noreen et al., 2016), and architectural design (Pelechano et al., 2008; Nagy et al., 2017). Some of these methods adhere to a grid-based approach in which (1) the spatial environment is represented by a regular grid of vertices; (2) the vertices are processed using some form of graph, tree, or array traversal algorithm; and (3) after processing each vertex, information is propagated to neighboring vertices. A typical grid-based navigation method will handle straight-line paths perfectly if they are oriented such that they pass directly through neighboring vertices, but not if they are oriented at any other angle. The purpose of this paper is to demonstrate a strictly grid-based approach for approximating straight path trajectories that head in arbitrary directions. The new approach is based on the simple and well-known procedure of counting the number of shortest paths on a grid.

The outcome of this work is a practical solution to a common problem in grid path planning. If there exists a grid path from some vertex A to another vertex B, there are often many shortest grid paths with the same length. An example of a case involving multiple shortest grid paths is shown in Figure 1. Typical grid-based implementations of Dijkstra's algorithm and A* prioritize grid moves using arbitrary tie-breaking conventions,



(a) A relatively indirect shortest grid path



(b) A more direct shortest grid path

Figure 1: Two shortest grid paths (purple arrows) are shown alongside illustrations of corresponding smoothed paths (green curves). Whereas a simple rule for prioritizing moves might generate a grid path like the one in (a), the grid path in (b) is more direct and should yield a shorter smoothed path.

which tends to produce relatively indirect paths. For instance, a convention to lead with diagonal moves would produce the relatively indirect shortest grid path in Figure 1a. The problem is how to select more direct shortest grid paths like the one in Figure 1b. The first grid path strays conspicuously far from line-of-sight trajectories as it traverses obstacle-free regions of the environment. The second grid path adheres reasonably well to sightlines in open regions. Either grid path could be smoothed in a post-processing step, but the grid path in Figure 1b should yield a shorter smoothed path than the one in Figure 1a.

It is possible to select relatively direct grid paths by conducting numerous line-of-sight tests or performing other geometric calculations. However, we find that decent results can be achieved using a simple path counting technique. Our solution is to count the number of shortest grid paths which traverse each vertex en route from A to B, then select the vertices with the highest counts. The approach still requires a basic grid path planning method such as Dijkstra's algorithm or A*, as one must construct a directed acyclic graph of all shortest grid paths. Once this is done, the counting procedure can be applied. We refer to this approach as central grid path planning, and introduce Central Dijkstra and Central A* as specific methods for computing central paths. We also present a theoretical analysis consisting of (1) observations on the relationship between path counting and an existing grid-based visibility approach; (2) an argument based on the central limit theorem that central grid paths converge on clear sightlines as the grid spacing approaches zero; and (3) a conjecture that central paths also converge on direct paths.

2. Background and Assumptions

We begin by defining terms and reviewing the basic concepts and assumptions used throughout the paper. The review presents related work on paths, grids, grid paths, grid path planning, any-angle path planning, and grid-based visibility.

2.1 Paths

A *path* is a directed curve which stretches from one point to another without intersecting itself or passing through obstacles. If one imagines a path as a string, then pulling on the string will generally cause the path to shorten until it is pulled taut. A *taut path* is a path that does not get shorter when one "pulls" on the ends (Oh & Leong, 2017). We introduce the concept of a *direct path*, which means that a sightline between any pair of points on the path must be part of the path itself. A direct path is always taut, but a taut path is not necessarily direct. There is also the well-known concept of a *shortest path*, a path with the minimum length of all possible paths between a given pair of endpoints. A shortest path is always direct, but a direct path is not necessarily among the shortest. Non-taut, taut, direct, and shortest paths are illustrated in Figure 2.

Applications involving navigation have diverse requirements. Nevertheless, we assume shortest paths are generally preferred over direct paths, direct paths over taut paths, and taut paths over non-taut paths. In addition, we assume paths which share the same topology as a shortest path are preferred over paths which do not. Two paths with the same endpoints share the same *topology* if the region between them is free of obstacles. Of the paths in Figure 2, only the first two have a common topology.

Paths may partially or fully reside on the boundaries of obstacles as long as there is always a traversable region on at least one side. A taut, direct, or shortest path can only contact obstacle boundaries at certain places: at the endpoints of the path, at convex corners encountered along the path, or alongside straight or convex surfaces of obstacles. Everywhere else, such paths are perfectly straight. Paths may or may not be permitted to pass through single-point gaps, and the methods and associated theory presented in this paper should apply regardless of which convention is used. Examples of convex and concave surfaces and corners, and a single-point gap, are indicated in Figure 2.

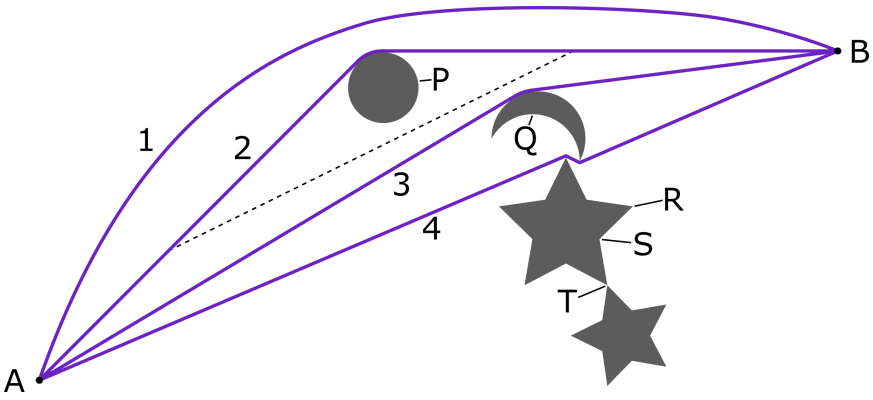


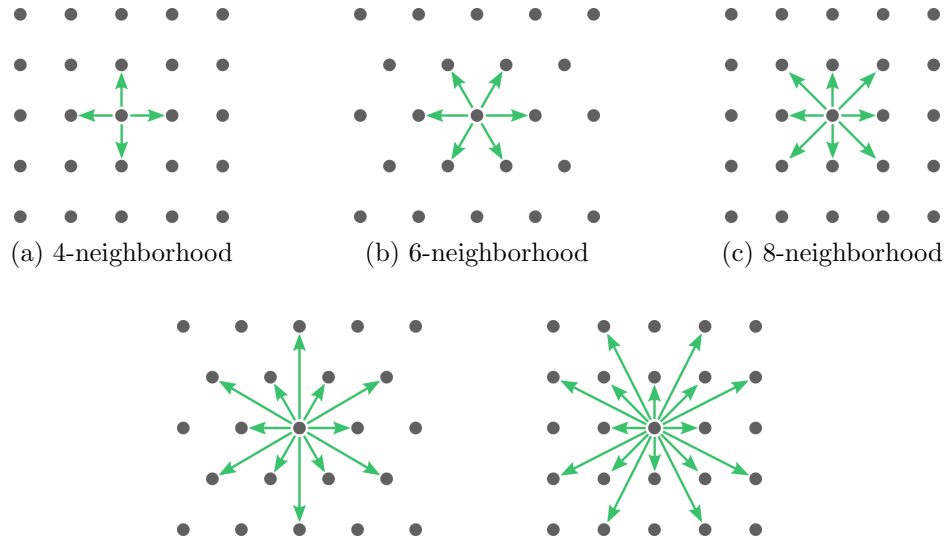
Figure 2: An example featuring a (1) non-taut path, (2) taut path, (3) direct path, and (4) shortest path between points A and B. The dashed sightline connecting two points on the taut path illustrates why the path is not direct. Also shown are a (P) convex surface, (Q) concave surface, (R) convex corner, (S) concave corner, and (T) single-point gap.

While our focus is on grid-based methods, paths can also be computed on a continuous domain using analytic approaches. One such approach is the visibility graph, where (1) obstacles are represented by polygons, (2) corners of polygons and endpoints of paths become vertices in a graph, (3) sightlines between those vertices become edges in the graph, and (4) paths are computed using a graph-based shortest path algorithm. The visibility graph was introduced by Lozano-Pérez and Wesley (1979), who also describe the useful technique of enlarging obstacles to enforce a minimum passage width. Analytic methods tend to give exact results if the analytic representation of the environment is exact. For example, the visibility graph gives exact shortest paths for environments with no curved surfaces.

2.2 Grids

Conventional grid-based methods employ a regular grid of vertices, and restrict information flow such that the state of each vertex can directly influence only neighboring vertices. A *neighborhood* is a set of offsets that determine which vertices are considered neighbors. In 2D, commonly used neighborhoods include the 4- and 8-neighborhood on a rectangular grid and the 6-neighborhood on a triangular grid. Rivera et al. (2020) provide a detailed description of the broader set of 2^k-neighborhoods on rectangular grids ($k \geq 2$), and an analogous set of (3×2^k)-neighborhoods could be described for triangular grids ($k \geq 1$). We refer to these two sets of neighborhoods collectively as the *standard 2D grid neighborhoods*. Examples of these neighborhoods appear in Figure 3.

A *move* is a vector from one vertex to any of its neighbors. The 4-neighborhood and the 6-neighborhood include only *cardinal moves*, which point to the nearest vertices along the primary axes of the grid. We assume all cardinal moves have a length of one grid spacing. Each successive neighborhood of the same grid type is constructed by taking each pair of adjacent moves and inserting a new move between them. Each inserted move is the vector sum of the original two (Rivera et al., 2020). For example, each diagonal move in the 8-neighborhood is the vector sum of the two surrounding cardinal moves.



(a) 4-neighborhood (b) 6-neighborhood (c) 8-neighborhood

(d) 12-neighborhood (e) 16-neighborhood

Figure 3: Examples of standard neighborhoods for rectangular and triangular grids.

Unless otherwise stated, all grid path planning examples and illustrations in this paper use the 8-neighborhood. This should not be interpreted as an endorsement of 8-neighbor grids over the other options. The proposed methods and associated theory pertain to all of the standard 2D grid neighborhoods described above, and should generalize to certain 3D grids as well. Although grid-based navigation methods with larger neighborhoods tend to be more challenging to implement, they can be expected to yield shorter paths.

2.3 Grid Paths

A *grid path* begins at a start vertex, then follows a sequence of moves to other not-yet visited vertices until reaching a destination vertex. Using a standard 2D grid neighborhood, and assuming no obstacles, the shortest grid paths between two vertices A and B are sequences involving at most two distinct *bracketing moves*. If the vector from A to B is aligned with a neighborhood move, then that move is the sole bracketing move. Otherwise, the bracketing moves are the pair of adjacent neighborhood moves that lie on either side of the A-B vector. To get from $[0, 0]$ to $[6, 3]$ using the 8-neighborhood, for example, the bracketing moves are $[1, 0]$ and $[1, 1]$ and one needs three of each. More generally, if the vector from A to B is $[x, y]$, and if the two bracketing moves are $\mathbf{u} = [u_x, u_y]$ and $\mathbf{v} = [v_x, v_y]$, then the shortest possible grid paths include exactly m moves in the $\mathbf{u}$ direction and k moves in the $\mathbf{v}$ direction, where m and k are given by the coordinate transformation below.

$$\begin{bmatrix} m \\ k \end{bmatrix} = \begin{bmatrix} u_x & v_x \\ u_y & v_y \end{bmatrix}^{-1} \begin{bmatrix} x \\ y \end{bmatrix}$$

The transformation simplifies to the following.

$$m = \frac{v_y x - v_x y}{u_x v_y - v_x u_y} \qquad k = \frac{-u_y x + u_x y}{u_x v_y - v_x u_y} \tag{1}$$

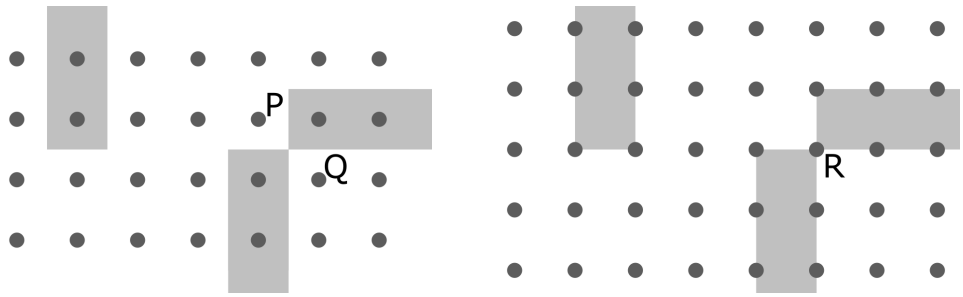
Adding the lengths of all moves in one of these sequences yields a well-known metric that we refer to as the *standard 2D grid distance*, and denote $h(x, y)$.

$$h(x, y) = m\sqrt{{u_x}^2 + {u_y}^2} + k\sqrt{{v_x}^2 + {v_y}^2} \tag{2}$$

The standard grid distance reduces to what is known as the Manhattan distance for 4-neighbor grids or the octile distance for 8-neighbor grids. Rivera et al. (2020) prove that $h(x, y)$ is the minimum possible grid path length for all standard rectangular 2D grid neighborhoods, and provide algorithms which compute this metric for up to 64 neighbors.

Obstacles have the effect of disallowing moves between certain pairs of vertices. In the presence of obstacles, a shortest grid path between A and B may be longer than the standard grid distance, or there may be no grid paths at all between the two points. Given a set of obstacles defined on a continuous domain, a grid-based approximation of the environment can be constructed by overlaying a grid of vertices. Moves between two neighboring vertices are allowed if and only if there is a straight-line path between them.

Sometimes the obstacle geometry itself takes the form of a grid, either because a preexisting continuous representation of the environment was rasterized or because the environment was originally designed as a grid. Grid-based obstacle geometry can usually be regarded as an array of square, triangular, or hexagonal cells, where each cell is either blocked or unblocked. When a rectangular grid of vertices is overlaid on an environment of square cells, it is common practice to place the vertices on the centers of cells, as in Figure 4a, or on the corners of cells, as in Figure 4b. We assume that vertices on triangular grids would be placed on the centers of hexagonal cells or on the corners of triangular cells.



(a) Vertices placed on cell centers (b) Vertices placed on cell corners

Figure 4: Vertex placement with grid-based obstacle geometry.

While theoretical and empirical studies have shown that placing vertices on square cell corners produces shorter paths (Bailey et al., 2015), the cell center convention may be more convenient for certain applications such as tile-based video games. The grid-based methods proposed in this paper work with either convention, and in either case we treat grid-based obstacles the same as continuous obstacles. If a straight-line path exists between two neighboring vertices, moves between those vertices are allowed. An important consideration arises for grid-based obstacle geometry with single-point gaps, such as the gap between vertices P and Q in Figure 4a or at vertex R in Figure 4b. Paths through such gaps may be either permitted or prohibited.

2.4 Grid Path Planning

The classic approach to grid path planning involves setting up a grid-based representation of an environment, as described in Section 2.3, then applying Dijkstra's graph-based shortest path algorithm (Dijkstra, 1959), or its A* variant (Hart et al., 1968), to find one or more shortest paths.

Grid-based implementations of Dijkstra's algorithm typically compute a hierarchy of paths between a *source vertex* and all accessible vertices in the environment. Every vertex is assigned an initial grid distance of infinity, except the source which has a grid distance of zero. The algorithm proceeds by expanding not-yet processed vertices in order of grid distance. When a vertex is expanded, finite grid distances are computed for its neighboring vertices making them eligible for expansion. If different grid distances are computed for the same vertex, the shortest is always selected. Once this search procedure is complete, shortest grid paths can be generated by retracing moves from any processed vertex to the source vertex. If needed, paths can be reversed so that they start at the source.

Grid-based implementations of A* are similar, except that vertices are expanded in an order that more efficiently reaches a pre-selected *goal vertex*. With A*, vertices are ordered by their grid distance from the source plus a lower-bound estimate of their distance to the goal. In this paper, we assume the lower-bound estimate is the standard 2D grid distance $h(x, y)$ specified in (2), though other heuristics are sometimes used. If the goal is among the unprocessed vertices with the minimum combined distance, the search procedure is terminated and a shortest grid path is generated from the goal to the source. The path can be reversed if needed.

Typical implementations of Dijkstra's algorithm or A* employ some form of tie-breaking during the search procedure, thereby selecting a single solution from the multitude of shortest grid paths that usually exist between the source and goal. One tie-breaking convention is to generate what are known as *canonical paths* by prioritizing diagonal moves over cardinal moves on an 8-neighbor grid (Sturtevant & Rabin, 2016). The shortest grid path in Figure 1a is an example of a canonical path, assuming vertex A is the source. The approach can be generalized to larger neighborhoods by prioritizing moves that are vector sums of the two surrounding moves (Rivera et al., 2020).

Canonical paths arise out of the work of Harabor and Grastien (2011), who demonstrate that prioritizing diagonal moves allows the A* algorithm to jump over certain vertices during the search procedure. The result is a faster variant of A* called Jump Point Search. Sturtevant and Rabin (2016) use the same ordering of moves to propose the Canonical Dijkstra and Canonical A* methods. Canonical Dijkstra incorporates jumping into Dijkstra's algorithm to reduce the number of vertex expansions. Canonical A* avoids jumping, but demonstrates that canonical ordering alone can reduce the number of neighbors that need to be visited for each vertex expansion in A*.

For implementations of Dijkstra's algorithm or A* that do not enforce a canonical ordering, the selected shortest grid path is typically determined by other tie-breaking conventions. These conventions may include a clockwise or counterclockwise ordering of neighborhood moves, a lexicographical ordering of vertex coordinates, or any tie-break rule for nodes in a heap-based priority queue. The resulting paths may not be as extreme as the example in Figure 1a, but they tend to be relatively indirect compared with the path in Figure 1b.

An alternative to the above tie-breaking conventions is to represent all possible shortest grid paths leading back to the source vertex. This adaptation of Dijkstra's algorithm or A* is largely a matter of recording, for every visited vertex, the set of predecessor vertices that are traversed by any shortest grid path en route to the source. The result is a directed acyclic graph of paths, rather than a hierarchy. For A*, a slight change to the termination condition is also required. To capture all shortest paths, the search procedure must be continued until the goal vertex is not just one of the unprocessed vertices with the minimum combined distance, but rather the only unprocessed vertex with the minimum combined distance. For long, meandering paths, this stricter stopping criterion should not cause a significant increase in the number of vertices that need to be expanded. Our work shows how the all-paths variants of Dijkstra's algorithm and A* can be used to select relatively direct grid paths, improving quality rather than speed.

2.5 Any-Angle Path Planning

Any-angle path planning methods make use of a path planning grid and typically assume grid-based obstacle geometry, yet depart in some way from the usual constraints of grid-based approaches to achieve shorter, more direct paths. Field D* departs from grid-based constraints by allowing paths to bend at points located between pairs of vertices (Ferguson & Stentz, 2006). Theta* uses line-of-sight checks to connect grid vertices across distances far greater than the size of the neighborhood (Daniel et al., 2010). In the Anya method, the hierarchy that is constructed during the search procedure is not a hierarchy of vertices, but rather a hierarchy of line segments (Harabor et al., 2016). Whereas most any-angle path planning methods approximate shortest paths, Anya can be used to find exact shortest paths assuming the grid-based obstacle geometry is exact.

Some any-angle methods speed up the path planning process by performing a precomputation on the environment before any source or goal vertices are selected. Subgoal Graphs is an any-angle path planning method that uses a precomputation phase to identify key vertices, called subgoals, at the corners of grid-based obstacles (Uras et al., 2013). Paths are found on a graph of these subgoals, similar to the analytic visibility graph method, and then refined as needed. Block A* uses a hierarchical approach, partitioning grid-based environments into blocks of, for example, 5-by-5 vertices (Yap et al., 2011). Shortest path distances are precomputed between the grid vertices on the boundaries of each block, allowing subsequent A* searches to treat blocks rather than vertices as nodes. Block A* is an any-angle method assuming an analytic or any-angle method is used to precompute distances within each block. If a strictly grid-based method is used for the precomputation, then the overall method could also be considered grid-based.

Any-angle path planning is often contrasted with the simple approach of using a classic grid path planning method, as in Section 2.4, then smoothing the resulting grid path in a post-processing step. Botea et al. (2004) describe what has become a well-known path smoothing technique employing a succession of line-of-sight checks. Beginning at a vertex on one end of a grid path, the procedure is to repeatedly delete the subsequent vertex as long as the resulting path does not go through an obstacle. This greedy algorithm is usually successful at straightening and shortening paths, though it sometimes leaves conspicuous triangle-shaped detours unresolved. Han et al. (2020) propose an enhanced

"string-pulling" algorithm where line-of-sight checks are used not only to delete points, but also to insert new points on obstacle corners around which the path is "pulled" taut. Paths can also be smoothed using relaxation approaches that slightly shift the position of each point (Richardson & Olson, 2011), or interpolation approaches that replace piecewise linear intervals with polynomial curves, Bézier curves, or splines (Ravankar et al., 2018). Such local adjustments can be used to soften sharp turns or increase clearance from obstacles, but may be inefficient at straightening highly indirect grid paths. Regardless of the approach, smoothing a path does not change its topology.

2.6 Grid-Based Visibility

Computational methods for visibility have several similarities to those developed for navigation. First, visibility methods are applied in many of the same domains, including video games (Cohen-Or et al., 2003), robotics (Morini et al., 2010), and architectural design (Turner et al., 2001; Nagy et al., 2017). They are also formulated in a similar way, with analytic or grid-based representations of obstacles in a 2D or 3D spatial environment. In a visibility context, obstacles are interpreted as optical barriers rather than travel barriers, and the paths of interest are generally straight-line paths called *sightlines*.

While visibility problems are most often tackled with analytic methods, as reviewed by Ghosh (2007), or raycasting, as described by Roth (1982), there are also grid-based methods that will prove closely related to our path counting technique for navigation. Although grid-based visibility methods appear to enjoy only limited use outside of academia, they offer similar advantages to grid path planning and other grid-based approaches. They are easy to implement, and allow the trade-off between speed and accuracy to be conveniently adjusted by varying the grid spacing.

A grid-based visibility method from the level set community was derived from a partial differential equation by Tsai et al. (2004) and further validated by Kao and Tsai (2008). The method approximates the region visible from a source point. The original formulation assumes implicit obstacle geometry defined on a 4-neighbor grid, but here we adapt the method to use an explicit representation consistent with Section 2.3. We represent obstacles using a set of binary values $\mathcal{V}_{\mathbf{q}}^{\mathbf{r}}$, where $\mathcal{V}_{\mathbf{q}}^{\mathbf{r}} = 1$ means neighboring vertices $\mathbf{q}$ and $\mathbf{r}$ are mutually visible and $\mathcal{V}_{\mathbf{q}}^{\mathbf{r}} = 0$ means the sightline between them is blocked. The output of the method is a set of visibility scores $\psi_{x,y}$ in the range $0 \leq \psi_{x,y} \leq 1$. If $\psi_{x,y} \geq 0.5$, we classify $[x, y]$ as visible. Assuming the source is at $[0, 0]$ and the grid spacing is 1, the visibility scores for the first quadrant of a 4-neighbor grid are given by (3).

$$
\begin{aligned}
&\psi_{0,0} = 1 && \\
&\psi_{x,0} = \psi_{x-1,0}\mathcal{V}_{x-1,0}^{x,0} && x > 0 \\
&\psi_{0,y} = \psi_{0,y-1}\mathcal{V}_{0,y-1}^{0,y} && y > 0 \\
&\psi_{x,y} = \left(x\psi_{x-1,y}\mathcal{V}_{x-1,y}^{x,y} + y\psi_{x,y-1}\mathcal{V}_{x,y-1}^{x,y}\right)/(x+y) && x, y > 0
\end{aligned}
\tag{3}
$$

The calculation in (3) can be applied using a standard array traversal algorithm (e.g. $[x, y] = [0, 0]$, $[1, 0]$, $[2, 0]$, ..., $[0, 1]$, $[1, 1]$, $[2, 1]$, ...). It can be extended to the other three

quadrants by negating the x coefficient, the y coefficient, or both. It can be generalized to 3D by adding the obvious z terms to the numerator and denominator of the quotient.

Although the original level set formulation assumes the 4-neighborhood, we can extend the method to employ any of the standard 2D grid neighborhoods in Section 2.2. Recall from Section 2.3 that every vertex $[x, y]$ is bracketed by at most two moves $\mathbf{u}$ and $\mathbf{v}$. Having identified these vectors, the coordinate transformation in (1) can be used to obtain the number of moves m and k in the respective directions. These move counts can then be substituted into the extended formula below. The effect of these steps is to transform the coordinate space such that $\mathbf{u}$ is mapped to $[1, 0]$, $\mathbf{v}$ is mapped to $[0, 1]$, and $[x, y]$ is mapped to $[m, k]$. This linear transformation allows the validated 4-neighbor formula to be applied with $[m, k]$ in place of $[x, y]$.

$$\begin{aligned}
&\psi_{0,0} = 1 && \\
&\psi_{m,0} = \psi_{m-1,0}\mathcal{V}^{m,0}_{m-1,0} && m > 0 \\
&\psi_{0,k} = \psi_{0,k-1}\mathcal{V}^{0,k}_{0,k-1} && k > 0 \\
&\psi_{m,k} = \left(m\psi_{m-1,k}\mathcal{V}^{m,k}_{m-1,k} + k\psi_{m,k-1}\mathcal{V}^{m,k}_{m,k-1}\right)/(m+k) && m, k > 0
\end{aligned} \tag{4}$$

Consider applying this extended method to the 8-neighborhood, specifically within the cone between the positive x axis and the $x = y$ diagonal. Any vertex $[x, y]$ in this region is bracketed by the moves $[1, 0]$ and $[1, 1]$. Substituting these vectors into (1) yields $m = x - y$ and $k = y$, and substituting these expressions into (4) yields the formula below. Similar equations could be derived for all 8 convex cones generated by pairs of adjacent moves.

$$\psi_{x,y} = \left((x-y)\psi_{x-1,y}\mathcal{V}^{x,y}_{x-1,y} + y\psi_{x-1,y-1}\mathcal{V}^{x,y}_{x-1,y-1}\right)/x \qquad 0 < y < x$$

Formulating this method for all 3D neighborhoods would require a 3D coordinate transform analogous to the one in Section 2.3, but let us focus on the 26-neighborhood and consider just the convex cone generated by $[1, 0, 0]$, $[1, 1, 0]$, and $[1, 1, 1]$. The visibility scores for vertices in this region are computed as follows, and similar equations exist for all 48 tetrahedral regions.

$$\psi_{x,y,z} = \left(\begin{array}{cl} & (x-y)\psi_{x-1,y,z}\mathcal{V}^{x,y,z}_{x-1,y,z} \\ + & (y-z)\psi_{x-1,y-1,z}\mathcal{V}^{x,y,z}_{x-1,y-1,z} \\ + & z\psi_{x-1,y-1,z-1}\mathcal{V}^{x,y,z}_{x-1,y-1,z-1} \end{array}\right)/x \qquad 0 < z < y < x$$

Another grid-based visibility method was proposed by Fisher-Gewirtzman et al. (2013) for the urban design community. Formulated specifically for 3D voxel grids using the 26-neighborhood, their method is similar to the 26-neighbor variation of the adapted level set method discussed above. However, instead of performing linear interpolations on sets of three neighboring vertices within 48 tetrahedral regions, Fisher-Gewirtzman et al. perform bi-linear interpolations on sets of four neighboring vertices within 24 pyramidal regions.

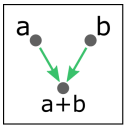
We refer to our adaptation of the level set visibility method as the *linear grid-based visibility* approach, and suggest that the 26-neighbor algorithm of Fisher-Gewirtzman et al. be called the *bi-linear grid-based visibility* approach. The linear approach has a particularly strong relationship to our proposed use of path counting for grid-based navigation.

3. Central Grid Path Planning

We now demonstrate how path counting can be used to select relatively direct grid paths, and explain how the technique can be implemented as an extension to Dijkstra's algorithm or A*. An empirical study compares the proposed Central A* method to A* and Theta*.

3.1 Path Planning with Counting

Our path planning approach is inspired by Pascal's triangle, the famous number pattern illustrated in Figure 5. In Pascal's triangle, a 1 is placed at the apex and repeated along the two diverging sides. Every other number is generated by adding two preceding numbers.



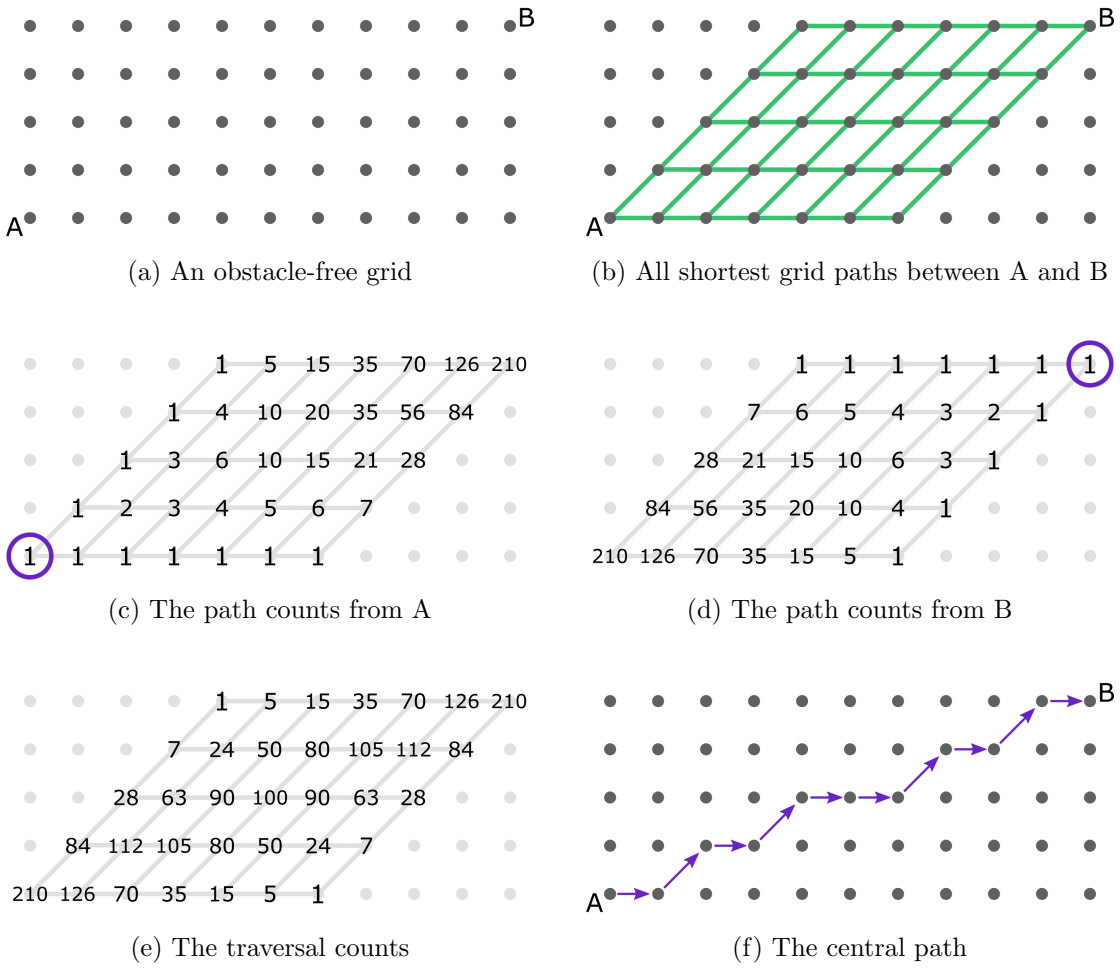
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1

Figure 5: The top several rows of Pascal's triangle.

It is widely understood that the recursive procedure which generates Pascal's triangle is also a path counting procedure. The apex of the triangle can be viewed as a source vertex on a standard 2D grid, and every integer is the number of shortest grid paths leading to that vertex from the source. For example, there are 70 shortest grid paths between the top of the triangle in Figure 5 and the vertex in the middle of the bottom row.

The relevance of Pascal's triangle to our path planning problem is most easily seen on a grid with no obstacles. Consider the task of selecting a relatively direct shortest grid path from vertex A to vertex B on the obstacle-free grid in Figure 6a. Figure 6b shows all of the shortest grid paths, which collectively form a parallelogram. To obtain a path that heads through the middle of the parallelogram, a first attempt might be to label each vertex with the number of paths from A. As shown in Figure 6c, this yields a parallelogram-shaped slice of Pascal's triangle. Prioritizing vertices with higher Pascal numbers would produce a path that heads relatively directly from A to the vertex labeled 70, and then proceeds toward B. The goal is a grid path that heads from A to B as directly as possible.

The key to approximating a direct path is to perform the counting procedure from both ends. As shown in Figure 6d, the path counts from B form the same Pascal number pattern as the path counts from A, except flipped across both axes. The next step is to multiply the two opposing sets of path counts to produce another type of path count. By taking the number of paths from A to a vertex, then multiplying by the number of paths from the vertex to B, one ends up with the number of paths which traverse the vertex en route



(a) An obstacle-free grid

(b) All shortest grid paths between A and B

(c) The path counts from A

(d) The path counts from B

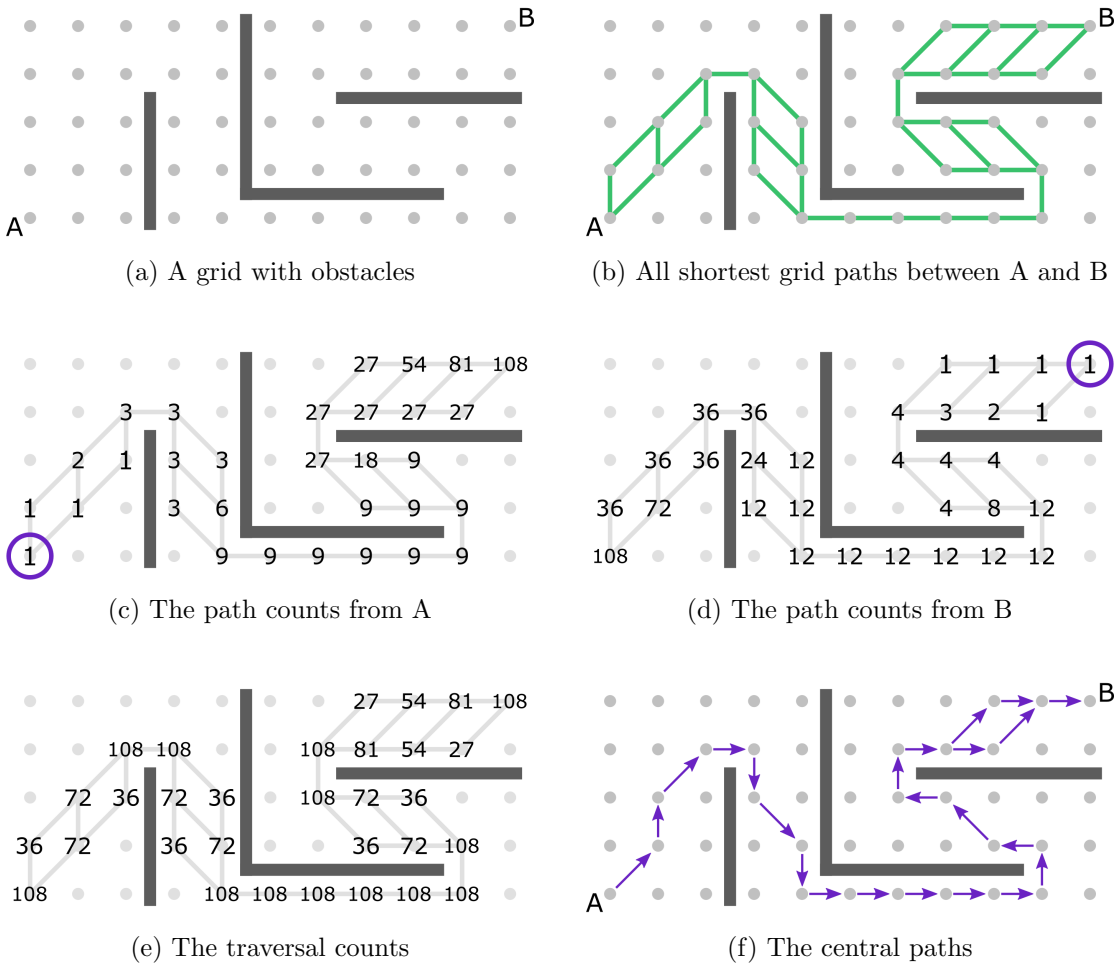
(e) The traversal counts

(f) The central path

Figure 6: The central path approach on a grid with no obstacles.

from A to B. We refer to this type of path count as a *traversal count*. Multiplying the path counts in Figure 6c and Figure 6d yields the traversal counts in Figure 6e. The final step is to generate a relatively direct grid path by starting at one end and selecting, for each move, the next vertex with the highest traversal count. Figure 6f shows the selected path for this obstacle-free example. We refer to such a path as a *central grid path*, or *central path*.

If obstacles are present in the environment, one cannot use Pascal's triangle directly. Yet it is still possible to employ the recursive counting procedure on which the famous number pattern is based. To illustrate, consider the scenario in Figure 7a. The task is to identify a relatively direct shortest grid path that circumnavigates the travel barriers. First, one must find all shortest grid paths from A to B, as shown in Figure 7b. Next, path counts are computed from one end, as in Figure 7c, and then the other, as in Figure 7d. Similar to Pascal's triangle, the path counting procedure begins by assigning 1 to the first vertex, then populating each succeeding vertex by adding the path counts of its predecessors. The remaining steps are the same with or without obstacles. Each vertex has two path counts,



(a) A grid with obstacles

(b) All shortest grid paths between A and B

(c) The path counts from A

(d) The path counts from B

(e) The traversal counts

(f) The central paths

Figure 7: The central path approach on a grid with obstacles.

one from A and one from B, and multiplying them yields the traversal counts in Figure 7e. One then selects the highest traversal counts, starting at A and proceeding toward B.

As indicated by the fork near the top right of Figure 7f, there are actually two central paths in this example. The reason for the two possible solutions is that, as can be seen in Figure 7e, one must choose between two vertices that both have a traversal count of 54. If there are multiple central grid paths, an implementation can choose any one of them. In general, central paths account for a small subset of all shortest grid paths between a pair of vertices. The purpose of the central path approach is to obtain a relatively direct shortest grid path by ensuring the selected path is a member of this subset.

Figure 8 shows what a central grid path may look like in practice. The path, which was generated using the architectural space analysis tool described by Goldstein et al. (2020), remains reasonably close to line-of-sight trajectories as it traverses obstacle-free regions of the environment. It is a considerably more direct path than one would expect from a typical grid-based implementation of Dijkstra's algorithm or A* with no path counting.

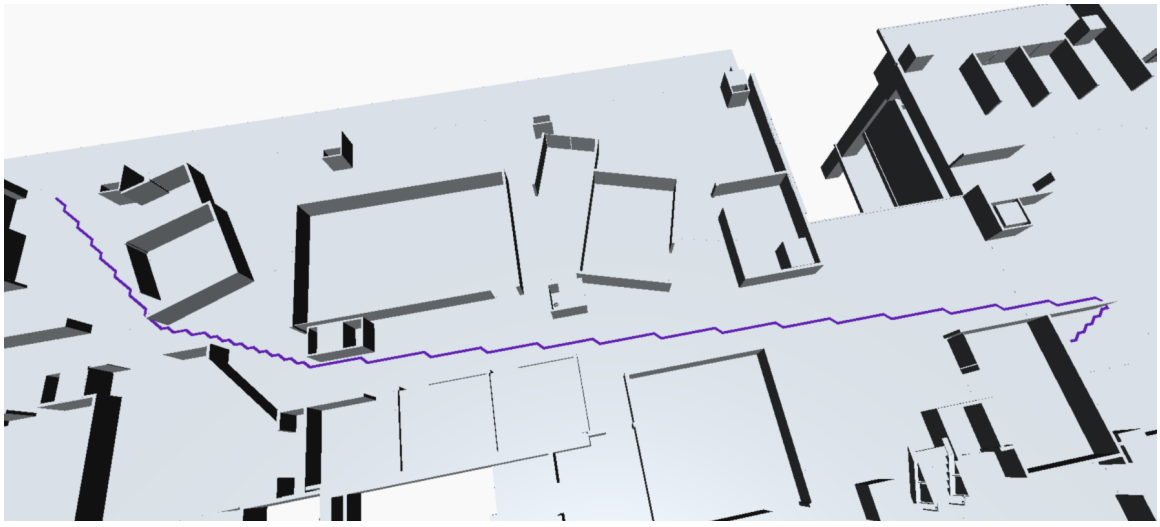


Figure 8: A central path computed on a building floor plan using a grid spacing of 25 cm.

3.2 Central Dijkstra and Central A*

Conceptually, the central path approach is as simple as finding all shortest grid paths, counting the numbers of paths that traverse each vertex, and selecting the vertices with the highest traversal counts. Implementing the approach is also reasonably simple, though the steps involved deserve some discussion.

Due to the need to count all shortest grid paths, it makes sense to begin with an all-paths implementation of Dijkstra's algorithm or A*. Extending these classic algorithms with path counting yields the Central Dijkstra and Central A* methods, which can be contrasted with the Canonical Dijkstra and Canonical A* methods reviewed in Section 2.4. Both the canonical and central approaches restrict the final output to a small subset of shortest grid paths. But whereas the canonical methods will likely yield somewhat shorter runtimes, the central methods offer more direct paths. The grid path in Figure 1a was previously identified as a canonical path. The grid path in Figure 1b is a central path.

Recall from Section 2.4 that an all-paths implementation of Dijkstra's algorithm or A* will record, for each visited vertex, the set of predecessor vertices that are traversed by any shortest grid path en route to the source. This directed acyclic graph of predecessors makes it easy to count paths from the goal vertex to the source, but not the other way around. Fortunately, when counting paths from the goal to the source, one can simultaneously construct a directed acyclic graph of successors pointing in the opposite direction. These successors can then be used to count paths from the source vertex to the goal.

Algorithm 1 formalizes the above strategy for generating a central grid path using data from an all-paths Dijkstra or A* search. Line 2 computes path counts from the goal vertex using predecessors from the all-paths search. Line 3 computes path counts from the source vertex using successors retrieved from the first path counting operation. Lines 4 to 14 generate the central path by starting at the source and selecting, for each move, a successor

vertex with the highest traversal count. Each traversal count is calculated on line 10 as the product of the path counts from the source and from the goal.

Algorithm 1: Generate a central path using data from an all-paths search

```
1  function generate_path(source, goal, predecessors, costs):
2      counts_from_goal, successors ← count_paths(goal, predecessors, costs)
3      counts_from_source, _________ ← count_paths(source, successors, costs)
4      path ← list()
5      path.append(source)
6      while path.last() ≠ goal do
7          best_successor ← none
8          highest_count ← −∞
9          foreach successor ∈ successors[path.last()] do
10             count = counts_from_source[successor] × counts_from_goal[successor]
11             if count > highest_count then
12                 best_successor ← successor
13                 highest_count ← count
14         path.append(best_successor)
15     return path
```

The costs argument in Algorithm 1 contains the grid distance from the source to each vertex, as computed by the initial all-paths search. For A*, these travel costs are referred to as g-values to distinguish them from the heuristics, or h-values, used to estimate distances to the goal. We suggest supplying these costs to the function that counts paths, as done on lines 2 and 3. While travel cost information is not strictly necessary for path counting, it provides a convenient way to ensure vertices are processed in a viable order.

Algorithm 2 defines the function that starts at a given source vertex and counts shortest grid paths represented by the supplied successors. Recall from above that this function is first invoked with the goal and predecessors of the initial path planning problem, which causes paths to be counted in the opposite direction as the initial search. Lines 2 and 3 initialize the path counts to 0 for all vertices except the source, which gets a path count of 1. Line 4 initializes all predecessors to the empty set. These predecessors may ultimately serve as successors in a subsequent invocation of the function. Lines 5 to 7 prepare a priority queue of unprocessed vertices, ordered using the travel costs from the initial all-paths search. If the cost of the source vertex is 0, the traversal is assumed to be from source to goal, so vertices with lower costs are processed first; otherwise, the traversal is assumed to be from goal to source, so vertices with higher costs are processed first. Either way, the source vertex and its cost are pushed onto the queue. Lines 8 to 15 repeatedly process the highest priority vertex in the queue by adding its path count to those of its successors.

The time complexity of Algorithm 2 is $O(N^2 \log(N))$, where N is the size of the spatial environment measured in grid spacings along one axis. The N^2 factor is proportional to the number of vertices that must be processed, and the $\log(N)$ factor assumes that the pop operation on line 10 is performed on a typical heap-based priority queue. Since an all-paths

Algorithm 2: Count paths using shortest path successors and costs

```
1  function count_paths(source, successors, costs):
2      counts ← array(costs.dimensions(), 0)
3      counts[source] ← 1
4      predecessors ← array(costs.dimensions(), ∅)
5      prioritize_lowest_cost ← (costs[source] = 0)
6      queue ← priority_queue(prioritize_lowest_cost)
7      queue.push(source, costs[source])
8      while ¬ queue.empty() do
9          vertex ← queue.top()
10         queue.pop()
11         foreach successor ∈ successors[vertex] do
12             if counts[successor] = 0 then
13                 queue.push(successor, costs[successor])
14             counts[successor] ← counts[successor] + counts[vertex]
15             predecessors[successor] ← predecessors[successor] ∪ {vertex}
16     return counts, predecessors
```

grid-based Dijkstra or A* search procedure is also $O(N^2 \log(N))$, introducing a subsequent path counting step does not increase the time complexity of the overall approach. In fact, since the counting procedure traverses a subset of the vertices visited during the initial search, intuition suggests that the additional runtime imposed by the counting step may be modest. This reasoning is explored experimentally in Section 3.3.

To ensure the central path methods are implemented efficiently and robustly, close attention should be paid to the data types used for predecessors (and successors), grid distances, and path counts. On a rectangular grid, one can keep track of a vertex's predecessors using an unsigned integer of one bit per neighborhood move. On an 8-neighbor grid, for example, recording all predecessors would require 8 bits, or one byte, per vertex. The successors would require another byte per vertex.

Since the exact length of a diagonal move is usually an irrational number, it may seem intuitive to represent grid distances using floating-point numbers. However, rounding errors characteristic of floating-point arithmetic will then produce small differences in the computed lengths of equally short paths. One can treat floating-point path lengths as equal if they differ by at most some small positive ϵ, but a simpler approach is to represent grid distances using fixed-point numbers or integers. On an 8-neighbor grid, cardinal and diagonal moves could be prescribed integer lengths such as 5 and 7, or 408 and 577. It was known long ago that $577/408 \approx \sqrt{2}$ (Kraft & Washington, 2014). Alternatively, one could assign all cardinal moves an arbitrary integer length such as 1000, then round the lengths of all other moves to the nearest integer.

Although integers should be preferred over floating-point numbers for grid distances, the opposite is true for path counts. Because they grow exponentially with distance, path counts will quickly overflow if they are implemented using any standard integer data type. Even

with floats, overflow remains an issue. Path counts can exceed the maximum representable value of a standard 64-bit floating-point number, a value greater than 10^{308}, after as few as 1030 moves. Note that restricting the width of an 8-neighbor grid to 1030 vertices or fewer, in both dimensions, does not necessarily avoid the problem; obstacles may cause shortest grid paths to meander around the environment and potentially surpass 10^{308} in number.

A solution to the overflow problem is to use 64-bit floating-point numbers to calculate and store logarithms of path counts rather than representing path counts directly. This technique requires a few changes to the above algorithms, though the changes can be encapsulated by a custom data type if desired. On lines 2 and 12 of Algorithm 2, the default path count of 0 is replaced with the asymptotic limit of $\log(x)$ at $x = 0$, which is $-\infty$. On line 3, the path count of the source vertex is replaced with $\log(1)$, which is 0. On line 14, the sum of two path counts a and b must be replaced with an operation that combines $\log(a)$ and $\log(b)$ to obtain $\log(a+b)$. No part of the operation can be vulnerable to overflow. Assuming all logarithms are base 2, a mathematical expression that meets the requirements is derived in (5). To ensure the subexpression $2^{\log(b)-\log(a)}$ does not overflow, the two operands must be sorted such that $\log(a) \geq \log(b)$.

$$\begin{aligned} \log(a+b) &= \log(a(1+b/a)) \\ &= \log(a) + \log(1+b/a) \\ &= \log(a) + \log(1+2^{\log(b/a)}) \\ &= \log(a) + \log(1+2^{\log(b)-\log(a)}) \qquad \log(a) \geq \log(b) \end{aligned} \tag{5}$$

The operation on the last line of (5) suffices to compute log path counts from the goal and from the source. The next step is to obtain the traversal counts, which can also be handled using logarithms. Instead of multiplying a vertex's path counts a and b to yield its traversal count ab, as indicated on line 10 of Algorithm 1, one adds the two log path counts $\log(a)$ and $\log(b)$ to produce the log traversal count $\log(ab)$.

$$\log(ab) = \log(a) + \log(b)$$

Since taking the log of a set of numbers preserves their ordering, central paths can be generated by selecting the vertices with the highest log traversal counts. It is not necessary to produce the traversal counts themselves. Due to rounding errors, a central path computed using logarithms may differ slightly from one based on exact path and traversal counts.

3.3 Empirical Comparison

Intuition suggests that the central path approach could serve as a relatively simple upgrade to Dijkstra's algorithm or A* for applications demanding higher quality paths in exchange for modest increases in runtime. To investigate the extent to which the approach meets this expectation, at least for certain conditions, we retrieved a copy of the open source any-angle path planning C++ library by Uras and Koenig (2015), and extended it with a Central A* implementation. We then collected statistics for the four basic methods below:

- Regular A*: The existing grid-based A* implementation in which a shortest grid path is arbitrarily selected by way of various tie-breaking conventions.

- Smoothed A*: The Regular A* method followed by a smoothing operation that produces a shorter path with the same topology. We used the smoothing function included in the library, an implementation of the greedy algorithm reviewed in Section 2.5 that deletes path vertices as dictated by a succession of line-of-sight checks.

- Smoothed Central A*: The new Central A* implementation followed by smoothing.

- Regular Theta*: A basic any-angle path planning method included in the library.

To ensure an apples-to-apples comparison of runtimes, the all-paths A* search within the new Central A* implementation repurposes the existing data structures and Regular A* code where possible. Aside from details such as the library's use of floating-point numbers for distances, the subsequent path counting procedure closely reflects the algorithms and guidelines in Section 3.2. The A* and Central A* methods employ the standard grid distance heuristic reviewed in Section 2.3, which is also the octile distance since our study is limited to the 8-neighborhood.

The tests were conducted using benchmark sets from the repository of Sturtevant (2012). Each set is a collection of maps containing 2D grid-based obstacle geometry, and each map has an associated list of path planning problems specifying start and end vertices. We selected the following benchmarks: video game maps from *Baldur's Gate II*, scaled to 512×512 grid cells; game maps from *Dragon Age: Origins*, ranging from 22×28 to 1260×1104; game maps from *StarCraft*, ranging from 384×384 to 1024×1024; and randomly generated maps with 10%, 20%, 30%, and 40% blocked cells, at 512×512 resolution. These are the same maps tested by Uras and Koenig (2015) and Harabor et al. (2016), though the problem sets were updated in 2018. The C++ library processes the maps by placing vertices on the corners of cells and permitting passage through single-point gaps.

Table 1 reports the average suboptimality of the paths produced by each method. Path-length suboptimality, the quality metric employed by Uras and Koenig (2015) and others, is the difference in length between the computed paths and the shortest possible paths expressed as a percentage of the latter. Lower values indicate shorter, more desirable paths. We calculate suboptimality scores by averaging path lengths across all problems associated with the same map, then calculating the suboptimality for each map, then averaging suboptimalities across all maps in the benchmark set. The shortest possible paths are computed using the library's Anya implementation. Suboptimality scores for Central A* without smoothing are not shown, as they are identical to those of Regular A*. Shortest grid paths produced by any method are necessarily equal in length, and must therefore be smoothed before they can be compared with one another using this metric.

The results in Table 1 support the expectation that one can obtain shorter paths by smoothing a central grid path than by smoothing the output of a typical grid-based A* implementation. For the game maps, Smoothed Central A* yielded suboptimality averages roughly ten times lower than those of Smoothed A*, and only about 50% higher than those of Theta*. The path planning errors that arise with such maps are discussed further below. For the random maps, where the effectiveness of path counting is compromised by numerous single-cell passages and single-point gaps, Smoothed Central A* yielded suboptimalities closer to the geometric average between those of Smoothed A* and Theta*.

Benchmark Set	# of Maps	# of Problems	Average Suboptimality (%)			
			Regular A*	Smoothed A*	Smoothed Central A*	Regular Theta*
Baldur's Gate II	75	122600	4.8735	0.6686	0.0643	0.0421
Dragon Age: Origins	156	155620	4.6526	0.8935	0.1072	0.0642
StarCraft	75	211390	5.3018	1.1842	0.1091	0.0941
Random-10%	10	17970	4.5915	1.9175	0.4106	0.1446
Random-20%	10	19150	4.4357	2.2955	0.6336	0.2121
Random-30%	10	20780	4.4795	2.4990	0.7691	0.2421
Random-40%	10	36370	4.4051	2.1822	0.7092	0.2369

Table 1: Average path-length suboptimality for the four tested methods.

Benchmark Set	Average Runtime (ms)			Average # of Expansions		
	Smoothed A*	Smoothed Central A*	Regular Theta*	Smoothed A*	Smoothed Central A*	Regular Theta*
BG II	3.07	4.71	9.25	13226	13868	11945
DAO	1.24	1.69	3.39	5667	5870	5734
SC	12.55	15.96	43.42	50706	51450	50046
R-10%	3.41	7.85	2.88	12518	14335	5812
R-20%	3.37	5.71	4.45	11768	12863	9951
R-30%	3.82	4.76	6.30	13081	13516	15058
R-40%	3.71	4.18	5.78	14358	14404	15658

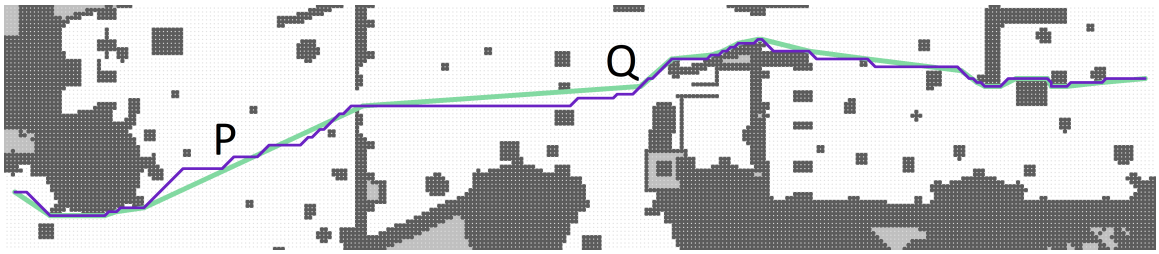
Table 2: Average runtime and average number of expansions per search.

Table 2 reports the average runtime for generating a single path, as well as the number of expansions during the main search procedure of each method. The runtime and expansion statistics are calculated by averaging first across all problems associated with the same map, and then across all maps in the benchmark set. Runtimes for A* and Central A* without smoothing are not shown, but smoothing grid paths took only about 1% as long as generating them. Tests were run on a 2.7GHz Intel Core i7 laptop with 16GB of RAM.

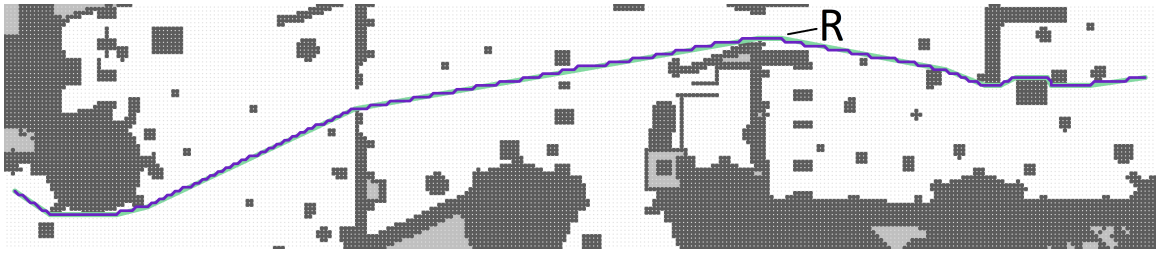
The results in Table 2 mostly support the expectation that Central A* imposes only modest increases in runtime compared with a typical implementation of A*. For the game maps, Central A* took roughly 30% to 50% longer than A* as a result of (a) a small increase in the average number of expansions, as shown in the table; (b) a small increase in the time required to process each expansion; and (c) the additional step of counting paths. The additional expansions can be explained by the stricter termination condition associated with the all-paths A* search, as described in Section 2.4. Compared with A*, the central path approach introduced only a 13% increase in average runtime for random maps at 40% obstacle density; however, the additional counting step became progressively more significant at 30% density, 20% density, and 10% density where path counting was as costly as the initial search. Central A* achieved shorter runtimes than Theta* for all benchmark

sets except for the 10% and 20% random maps. Runtimes were roughly 2-3 times shorter for the game maps.

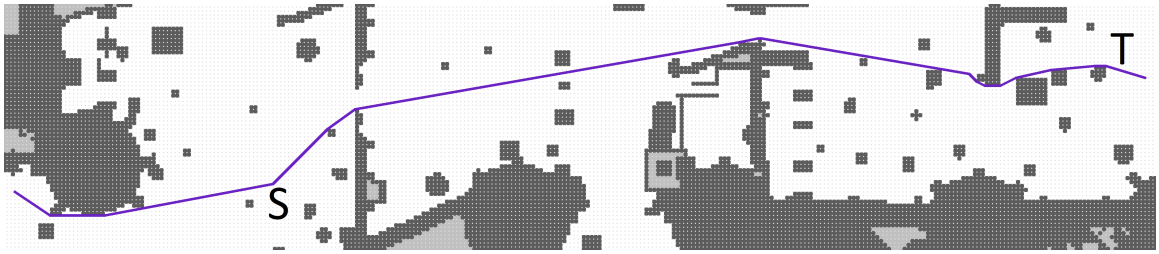
Several qualitative advantages and disadvantages of the central path approach are revealed by examining individual scenarios such as the one in Figure 9. For this problem, the A* solutions in Figure 9a clearly exhibit the flaws that central paths are intended to address. First, typical A* grid paths feature artifacts such as the pronounced terraces near label P, though these particular terraces are resolved by the smoothing operation. Second, it is relatively common for an arbitrarily selected grid path to pass an obstacle on a side that is unmistakably suboptimal, as seen near label Q. These topology errors cannot be resolved by smoothing. A third issue in the example, also near label Q, is that the greedy smoothing algorithm overshoots a corner of an obstacle and causes a conspicuous triangle-shaped detour to appear in the smoothed path.



(a) A* (thin purple line) and Smoothed A* (thick green line)



(b) Central A* (thin purple line) and Smoothed Central A* (thick green line)



(c) Theta*

Figure 9: A section of map "ost000a" from the *Dragon Age: Origins* benchmark set, distributed by the Moving AI Lab with permission from BioWare. The paths are solutions to instance 800 (numbering from 0) of the 2018 problem set. The source is on the left.

The Central A* grid path in Figure 9b avoids the artifacts near P and the topology error near Q. The greedy smoothing algorithm still overshoots obstacles, particularly near label R, but the relative directness of a central grid path limits the severity of these smoothing defects. Because central paths are nearly direct to begin with, they should accommodate relaxation- or interpolation-based smoothing techniques that avoid line-of-sight checks and are capable of producing more curved paths. Alternatively, applying the "string-pulling" algorithm of Han et al. (2020) to a central path should yield a smooth path that is taut, or nearly taut, and highly direct.

Although Theta* outperforms 8-neighbor Central A* in terms of average path length, we submit that defects in a basic Theta* path may be more noticeable when they do occur. One such defect appears in Figure 9c near label S, where an unfortunate confluence of factors leads Theta* to skip certain line-of-sight checks that would have revealed a more direct trajectory. With its natural tendency to prioritize vertices near sightlines, the central path approach avoids such conspicuous errors. Central paths can be suboptimal, however, when there are two plausible path topologies and no sightline to clarify the better option. A case in point occurs at label T, where the optimal topology found by Theta* has no shortest grid path and is hence inaccessible to A* and Central A*. Our impression is that topology errors of this nature are less obvious to the human eye than defects involving a severe deviation from a sightline. Nevertheless, one might consider using a 16-neighbor Central A* implementation for applications that demand even higher quality paths.

4. Theoretical Analysis

A curious feature of the central path approach is that it tends to produce grid paths with nearly straight sections spanning arbitrary distances at any angle, yet the only vectors or line-of-sight tests required are between neighboring vertices. Here we pursue a theoretical understanding of why the approach works. We begin by observing a relationship between path counting and grid-based visibility. Next, we confirm that central grid paths converge on clear sightlines. Finally, we conjecture that central paths converge on direct paths.

4.1 Visibility by Counting

The fact that central grid paths tend to approximate straight-line paths suggests that shortest grid paths are typically concentrated around sightlines. Intuitively, this implies that if a sightline from point A to point B is not blocked by an obstacle, then the number of shortest grid paths between A and B is likely close to the maximum possible number. On the other hand, if the sightline is blocked, then the number of shortest grid paths is likely closer to zero. This idea can be used to develop a visibility method based on path counting. The visibility by counting method turns out to be mathematically equivalent to the linear grid-based visibility approach adapted from Tsai et al. (2004) in Section 2.6, but we present it here to shed light on the theoretical properties of central grid paths.

Recall from Section 2.3 that on an obstacle-free grid, shortest grid paths are sequences of at most two distinct bracketing moves. In Section 3.1, we observed that the number of such paths from a source vertex to any other vertex is the number at the corresponding position of Pascal's triangle. As shown in Figure 10 below, counting the number of shortest grid paths heading outward from a source vertex on an λ-neighbor grid will generate λ

copies of Pascal's triangle. The triangles will overlap by one vertex in directions for which there is only one shortest grid path to each vertex.

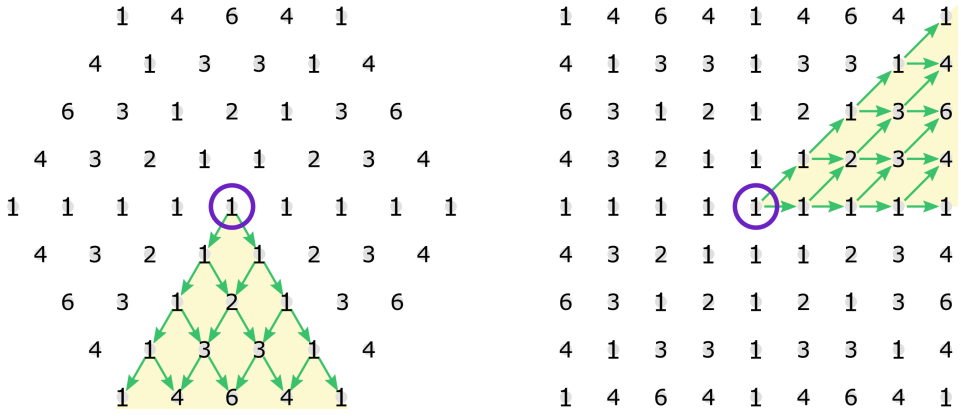


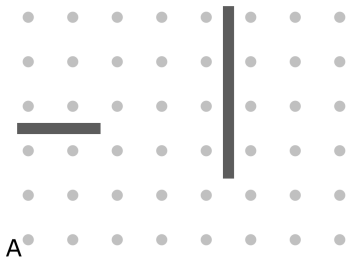
Figure 10: Vertices labeled with the number of shortest grid paths heading outward from a source vertex on a 6-neighbor grid (left) and an 8-neighbor grid (right). The numbers form overlapping copies of Pascal's triangle, one of which is highlighted in each example.

The idea is now to compute the fraction of shortest grid paths heading outward from a source vertex that are not obstructed by obstacles. This fraction is the visibility score for each vertex. The region visible from the source is approximated as the set of vertices with at least 50% of the paths unobstructed.

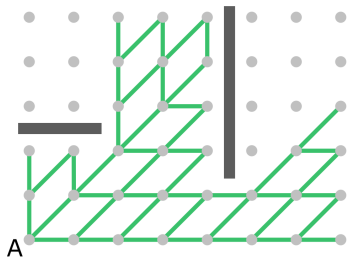
To illustrate, consider the scenario in Figure 11a. The task is to evaluate sightlines from the source vertex A to all other vertices in the environment. Figure 11b shows all of the grid paths heading outward from A that are not obstructed by obstacles. Observe that unlike the counting approach proposed for navigation, the paths used for visibility do not wrap around obstacles but continue to advance from the source according to the bracketing moves. Figure 11c shows the path counts for these outward grid paths. Because the paths always head outward from the source, the path counts can be computed using a standard array traversal (e.g. $[x, y] = [0, 0]$, $[1, 0]$, $[2, 0]$, ..., $[0, 1]$, $[1, 1]$, $[2, 1]$, ...). Figure 11d shows the path counts, or Pascal numbers, for all shortest grid paths ignoring any obstacles. Dividing the path counts in Figure 11c by those in Figure 11d yields the fractions, or visibility scores, in Figure 11e. Figure 11f highlights the vertices with visibility scores of at least 0.5, which happen to be exactly the set of vertices that are truly visible from A.

Like all grid-based visibility methods, it is possible for a vertex with a visibility score of 0.5 or higher to be occluded, perhaps only by a small obstacle. Similarly, it is possible for a vertex with a score below 0.5 to be visible, perhaps only through a small opening between obstacles. The accuracy of the method can be improved by choosing a larger neighborhood or by decreasing the grid spacing.

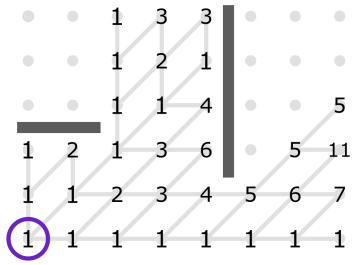
Similar to the central path approach, path counts computed for visibility can be stored and manipulated using logarithms to avoid overflow. Whereas the final counting step of the navigation approach involved adding two log path counts to obtain a log traversal count, here we must subtract two log path counts to obtain a log visibility score. If needed, it is always safe to exponentiate a log visibility score to obtain the fraction itself.



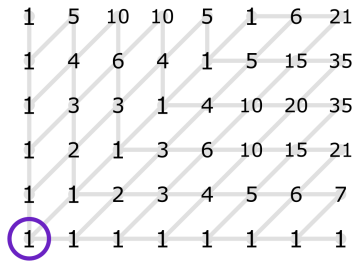
(a) A grid with obstacles



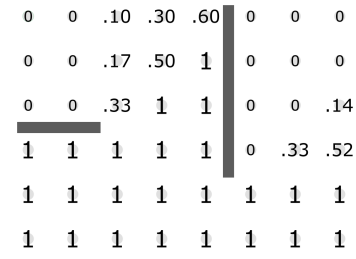
(b) All outward grid paths from A



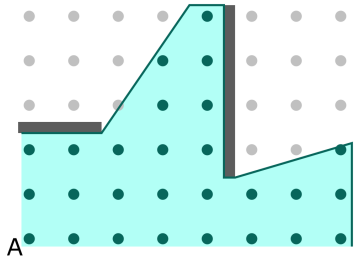
(c) The path counts from A



(d) The Pascal numbers from A



(e) The visibility scores

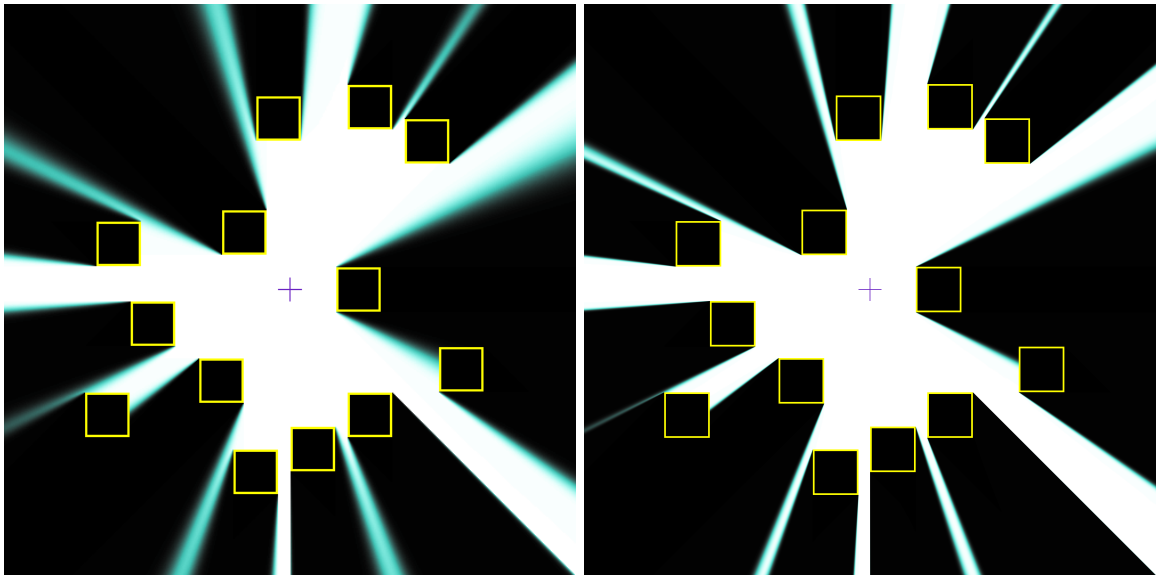


(f) The points visible from A

Figure 11: The visibility by counting approach.

Another way to avoid overflow is to restrict the size of any visibility analysis to fewer than 1030 grid moves from the source. When represented using 64-bit floating-point numbers, path counts for visibility will never overflow on 4-neighbor grids smaller than 515 vertices in both dimensions, or 8-neighbor grids smaller than 1030 vertices. Note that we cannot make the same guarantee for the navigation approach, where paths meander around obstacles.

The apparent convergence of the approach can be observed in Figure 12, where each pixel is colored based on the visibility score of the associated vertex. Fully obstructed vertices with scores of 0 are colored black, perfectly visible vertices with scores of 1 are colored white, and areas of uncertainty with scores between 0 and 1 are colored on a dark-to-light, cyan-colored spectrum. The solution on the 501×501 grid exhibits a certain degree of uncertainty around the edges of the shadows. When the resolution is increased to 5001×5001 vertices, these uncertain regions are reduced to thin slivers.



(a) 501×501 vertices (b) 5001×5001 vertices

Figure 12: Plots of the visibility scores produced by the path counting approach at two grid resolutions. The yellow boxes are obstacles and the source vertex is in the middle.

Observe that both solutions in Figure 12 feature two almost perfectly sharp shadow edges near the bottom center and bottom right of the field. This is due to the fact that two obstacles have been deliberately placed so that their corners line up with the source vertex in a cardinal or diagonal direction. In these exact directions there is one possible outward grid path, and in the vicinity of these locations there are not enough outward paths to produce the same tapering effect that occurs throughout most of the environment. This lack of isotropy is typical for grid-based visibility methods with explicit geometry, though view points and obstacles will not normally align as precisely as in Figure 12.

We now show that visibility by counting is mathematically equivalent to the linear grid-based visibility approach described in Section 2.6. The first step is to formally define the outward path counts, taking obstacles into account. We employ the notation established in Section 2.6 for any standard 2D grid neighborhood, where m and k are the numbers of bracketing moves required to get to any vertex from the source. In this case, however, m and k are not explicitly used in the calculations and need not be evaluated. The path count $c_{m,k}$ of each vertex is given by (6) below.

$$\begin{aligned} c_{0,0} &= 1 \\ c_{m,0} &= c_{m-1,0}\mathcal{V}^{m,0}_{m-1,0} && m > 0 \\ c_{0,k} &= c_{0,k-1}\mathcal{V}^{0,k}_{0,k-1} && k > 0 \\ c_{m,k} &= c_{m-1,k}\mathcal{V}^{m,k}_{m-1,k} + c_{m,k-1}\mathcal{V}^{m,k}_{m,k-1} && m, k > 0 \end{aligned} \tag{6}$$

Recall from Section 2.6 that $\mathcal{V}_{\mathbf{q}}^{\mathbf{r}} = 1$ means there is a straight-line path between neighboring vertices $\mathbf{q}$ and $\mathbf{r}$, and $\mathcal{V}_{\mathbf{q}}^{\mathbf{r}} = 0$ means the path is blocked by an obstacle. Ignoring all obstacles ($\mathcal{V}_{\mathbf{q}}^{\mathbf{r}} = 1$ everywhere), the same procedure generates the Pascal numbers $C_{m,k}$. Dividing each path count $c_{m,k}$ by the corresponding Pascal number $C_{m,k}$ produces the visibility score $\psi_{m,k}$.

$$\psi_{m,k} = \frac{c_{m,k}}{C_{m,k}} \tag{7}$$

Having listed the equations of the path counting approach, we now seek an alternative way of calculating the same scores. Instead of generating two sets of path counts and later dividing them, the division operations could be performed on the fly. This would allow one to store only the final visibility score for each vertex. Additionally, instead of computing the Pascal numbers $C_{m,k}$ recursively, one could treat them as if they were calculated using the explicit formula in (8). The formula expresses the number of ways to reorder a sequence of m items of one type and k items of another, which is equivalent to the number of sequences of bracketing moves comprising shortest paths on an obstacle-free grid.

$$C_{m,k} = \frac{(m+k)!}{m!k!} \tag{8}$$

To derive the alternative way of calculating $\psi_{m,k}$, we begin with (7), then substitute in the expression of (8), then incorporate the recursive step of (6), then simplify to remove all factorials. At the end of the analysis, we arrive at an alternative expression for $\psi_{m,k}$ that is identical to the calculation in (4) from Section 2.6, the formula for the linear grid-based visibility method. Thus we find that, ignoring rounding errors, the linear and counting approaches will produce the same results.

$$\begin{aligned}
\psi_{m,k} &= \tfrac{c_{m,k}}{C_{m,k}} \\
&= \left(\tfrac{m!k!}{(m+k)!}\right) c_{m,k} \\
&= \left(\tfrac{m!k!}{(m+k)!}\right) \left(c_{m-1,k}\mathcal{V}_{m-1,k}^{m,k} + c_{m,k-1}\mathcal{V}_{m,k-1}^{m,k}\right) \\
&= \left(\tfrac{m!k!}{(m+k)!}\right) c_{m-1,k}\mathcal{V}_{m-1,k}^{m,k} + \left(\tfrac{m!k!}{(m+k)!}\right) c_{m,k-1}\mathcal{V}_{m,k-1}^{m,k} \\
&= \tfrac{m}{m+k}\left(\tfrac{(m-1)!k!}{(m+k-1)!}\right) c_{m-1,k}\mathcal{V}_{m-1,k}^{m,k} + \tfrac{k}{m+k}\left(\tfrac{m!(k-1)!}{(m+k-1)!}\right) c_{m,k-1}\mathcal{V}_{m,k-1}^{m,k} \\
&= \tfrac{m}{m+k}\tfrac{c_{m-1,k}}{C_{m-1,k}}\mathcal{V}_{m-1,k}^{m,k} + \tfrac{k}{m+k}\tfrac{c_{m,k-1}}{C_{m,k-1}}\mathcal{V}_{m,k-1}^{m,k} \\
&= \left(m\psi_{m-1,k}\mathcal{V}_{m-1,k}^{m,k} + k\psi_{m,k-1}\mathcal{V}_{m,k-1}^{m,k}\right)/(m+k)
\end{aligned}$$

Although the two approaches are mathematically equivalent, it is worth noting that each involves a particular set of operations that may be more convenient for certain applications. The linear approach requires a division operation to be performed for each vertex during the grid traversal. With the counting approach, the division operations are deferred to the end of the method, and can sometimes be avoided entirely. If one requires only the classification of each vertex as visible or not visible, then the final calculation $c_{m,k}/C_{m,k} \geq 0.5$ can be

replaced with $2c_{m,k} \geq C_{m,k}$. On the other hand, the counting approach requires that (a) two floating-point numbers are calculated and stored for each vertex, instead of just one; or (b) the Pascal numbers $C_{m,k}$ are precomputed, then retrieved.

Aside from the minor practical trade-offs outlined above, the significance of visibility by counting is that it reveals a relationship between grid-based visibility and the proposed counting approach for grid-based navigation. Consider the scenarios in Figure 13. In the first scenario, a sightline between points A and B passes through a gap in a wall, clearing the obstacles by some small yet positive distance δ. It appears that the wall will block most of the possible shortest grid paths between A and B, thereby producing a visibility score less than 0.5. However, the work of Kao and Tsai (2008) suggests that the linear grid-based visibility score will converge on the correct result of 1 as the grid spacing approaches zero. It follows that the visibility by counting score will also converge on 1, and this in turn implies that nearly all possible shortest grid paths will go through the gap at a sufficiently fine grid resolution. In the second scenario, the sightline passes through the middle of a small circular obstacle of positive radius δ. Although the obstacle may appear to block only a small fraction of possible shortest grid paths, the fact that the visibility score converges on 0 assures us that nearly all of these paths will become blocked by the obstacle at a sufficient resolution. These insights about grid-based visibility support a theory that central grid paths converge on sightlines where possible and direct paths in general.

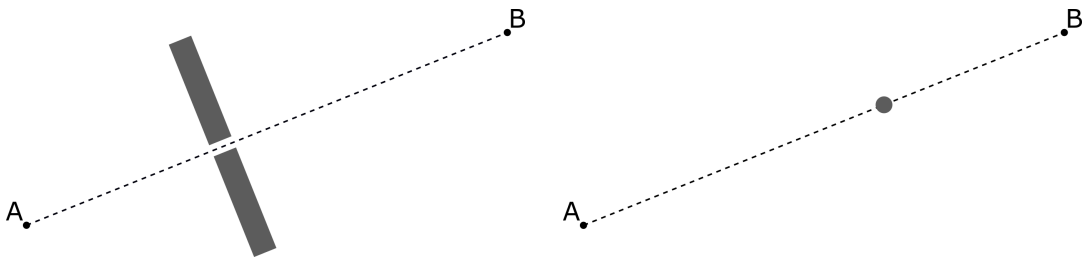


Figure 13: Two challenging scenarios in which grid-based visibility methods will nevertheless evaluate the sightlines correctly if the grid spacing is sufficiently small.

4.2 Convergence on Clear Sightlines

The relationship between path counting and grid-based visibility suggests that shortest grid paths become increasingly concentrated around sightlines as the grid spacing approaches zero. We now provide an explanation for this effect by analyzing the distribution of path and traversal counts between endpoints A and B on an obstacle-free grid. We first relate these path counts to the binomial distribution, then approximate this distribution with a normal distribution, then appeal to the central limit theorem to argue that central grid paths converge on the A-B sightline as the grid resolution is increased. We then observe that the convergence property holds even if obstacles are introduced into the environment, provided these obstacles neither contact nor block the sightline.

Our analysis can be extended to all of the standard 2D grid neighborhoods using the transformation in (1), but for simplicity we consider an 8-neighbor grid with start vertex A

located at the origin and end vertex B at $[N, K]$. Given $0 < K < N$, all shortest grid paths from A to B are sequences of the two bracketing moves $\mathbf{u} = [1, 0]$ and $\mathbf{v} = [1, 1]$. We are interested in the relative proportion of these shortest grid paths that, after n moves, pass through some vertex $[n, k]$. This formulation of the problem is illustrated in Figure 14.

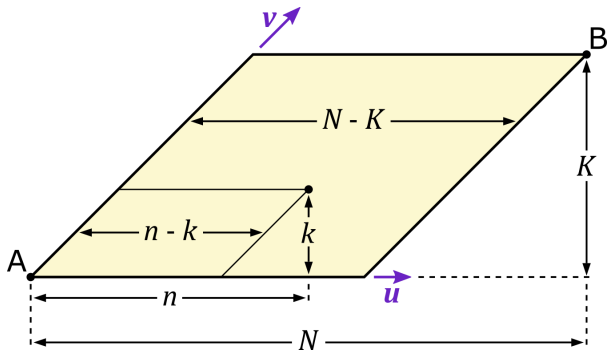


Figure 14: Endpoints A and B are separated by N moves, of which $N - K$ are in direction $\mathbf{u}$ and K are in direction $\mathbf{v}$. A point en route is reached after n moves from A, of which $n - k$ are in direction $\mathbf{u}$ and k are in direction $\mathbf{v}$. The illustration depicts an 8-neighbor grid with $\mathbf{u} = [1, 0]$ and $\mathbf{v} = [1, 1]$, but the analysis can be extended to other neighborhoods.

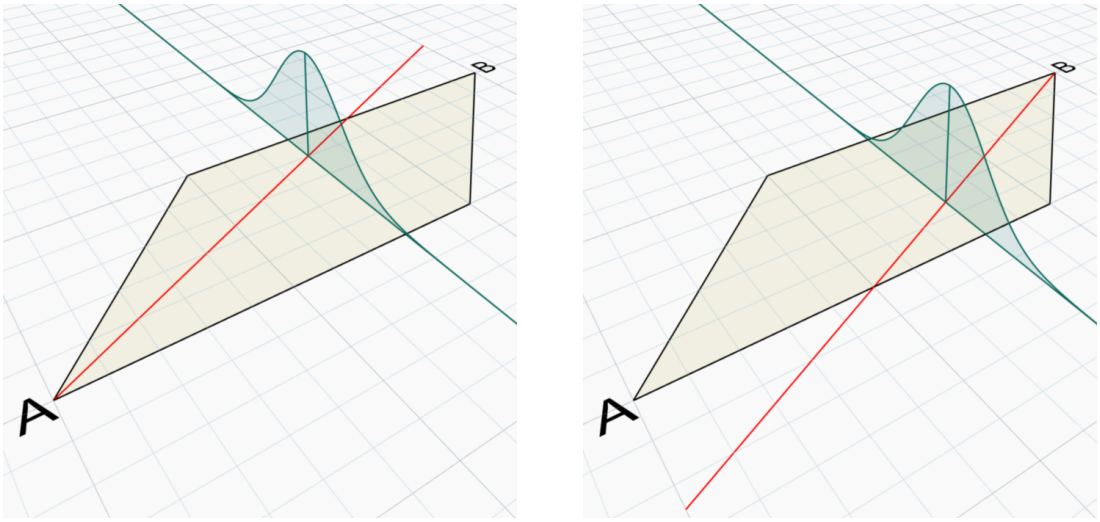
The number of ways to travel from A to $[n, k]$ is $\frac{n!}{k!(n-k)!} = 2^n f_n(k)$, where $f_n(k)$ is the probability mass function of a binomial distribution with success probability $p = 1/2$. Since we are interested in relative proportions, we drop the constant 2^n and focus on the binomial distribution. The distribution is defined for $0 \leq k \leq n$, but what matters for our analysis is that it accurately captures the relative proportions of path counts from A to any $[n, k]$ within the parallelogram of shortest grid paths. For each $n \in \{1, 2, 3, ..., N - 1\}$, we now approximate the binomial distribution using a normal distribution with matching mean μ_A and variance σ_A^2, as defined in (9). Depicted in Figure 15a, this normal distribution approximates the relative proportion of shortest grid paths that end at $[n, k]$ after n moves forward from A.

$$\mu_{\mathrm{A}} = \frac{n}{2} \qquad \sigma_{\mathrm{A}}^2 = \frac{n}{4} \tag{9}$$

Similarly in the other direction, the number of ways to travel from B to $[n, k]$ is $2^{N-n} f_{(N-n)}(K - k)$. Dropping the constant 2^{N-n}, we are left with a transformed binomial distribution that accurately captures the relative proportions of path counts from B to any $[n, k]$ in the parallelogram. For each $n \in \{1, 2, 3, ..., N - 1\}$, we once again approximate the binomial distribution using a normal distribution. The transformed mean μ_B and variance σ_B^2 are given in (10). Depicted in Figure 15b, this normal distribution approximates the relative proportion of shortest grid paths that end at $[n, k]$ after $N - n$ moves backward from B.

$$\mu_{\mathrm{B}} = K - \frac{N - n}{2} \qquad \sigma_{\mathrm{B}}^2 = \frac{N - n}{4} \tag{10}$$

As indicated in Section 3.1, the number of ways to travel from A to B passing through $[n, k]$ is the product of the number of paths from A to $[n, k]$ and the number of paths from $[n, k]$ to B. For each $n \in \{1, 2, 3, ..., N - 1\}$, we therefore approximate the proportion of



(a) Normal distribution for paths from A (b) Normal distribution for paths from B

Figure 15: Visualizations of the normal distributions which approximate the proportion of shortest grid paths (a) forward from A and (b) backward from B. The red lines indicate where the distribution peak would be located for various n between 0 and N. The distributions extend into the region outside the parallelogram where there are no relevant paths. However, since we are only interested in relative proportions within the parallelogram, the unused portions of the distributions do not impact the quality of the approximation.

A-B paths that traverse $[n, k]$ using the pointwise product of the probability density functions of the two normal distributions described above. Given any two normal distributions $\mathcal{N}(\mu_A, \sigma_A^2)$ and $\mathcal{N}(\mu_B, \sigma_B^2)$, it is known that the pointwise product of their density functions is proportional to the density function of a new normal distribution $\mathcal{N}(\mu_{AB}, \sigma_{AB}^2)$, the parameters of which are defined as follows (Bromiley, 2014).

$$\mu_{\mathrm{AB}} = \frac{\mu_{\mathrm{A}}\sigma_{\mathrm{B}}^2 + \mu_{\mathrm{B}}\sigma_{\mathrm{A}}^2}{\sigma_{\mathrm{A}}^2 + \sigma_{\mathrm{B}}^2} \qquad \sigma_{\mathrm{AB}}^2 = \frac{\sigma_{\mathrm{A}}^2\sigma_{\mathrm{B}}^2}{\sigma_{\mathrm{A}}^2 + \sigma_{\mathrm{B}}^2} \tag{11}$$

If we substitute the parameters of (9) and (10) into the formula for μ_{AB} in (11), most of the terms cancel and we are left with the following.

$$\mu_{\mathrm{AB}} = \frac{K}{N}n \tag{12}$$

The result in (12) reveals that the mean value of the combined normal distribution, which is depicted in Figure 16, falls exactly on the straight line between vertex A at $[0, 0]$ and vertex B at $[N, K]$. Since the density function of this distribution is approximately proportional to the traversal counts computed by the central path approach, we observe that the highest traversal counts should occur relatively close to the A-B sightline. This explains why central grid paths tend to adhere to line-of-sight trajectories, at least in the absence of obstacles. We further observe that, according to the central limit theorem, the

corresponding binomial and normal distributions converge as $N \to \infty$. It follows that if the grid resolution is successively refined, the proportion of paths traversing any region within the parallelogram will become increasingly consistent with the normal approximation. We therefore claim that a central path between two points in a 2D obstacle-free environment will converge on the sightline as the grid spacing approaches zero.

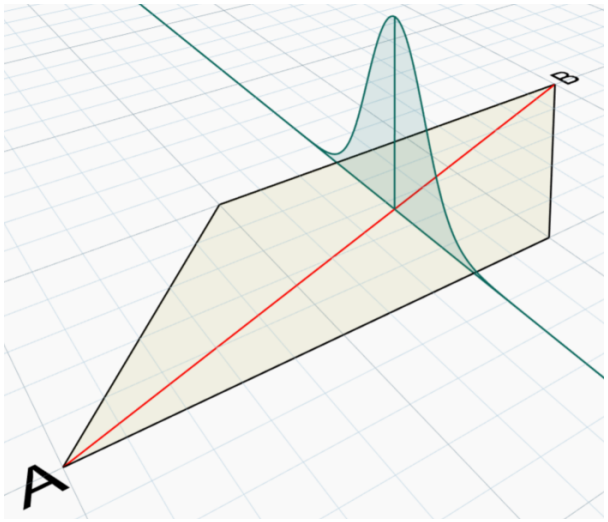


Figure 16: Visualization of the normal distribution which approximates the proportion of shortest grid paths between A and B. The red line indicates that the distribution peak coincides with the A-B sightline for all n between 0 and N.

We now consider the case of an obstacle somewhere within the parallelogram of shortest grid paths, but neither contacting nor blocking the A-B sightline. By cutting off some of the paths on one side of the sightline, the obstacle may cause the highest traversal counts to shift toward the opposite side. Figure 17 depicts an extreme case of this effect, where an obstacle spanning the full length of the path is offset from the sightline by only some small positive distance δ. As illustrated, the obstacle will induce a bulge in the central path.

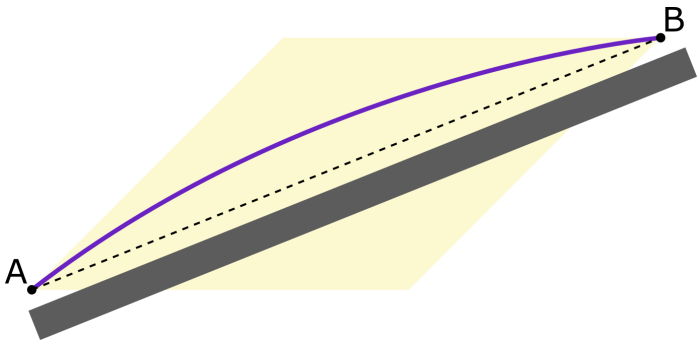


Figure 17: An illustration of an obstacle inducing a bulge in a central path.

To better understand the effect of obstacles on central paths, we now derive the standard deviation σ_{AB} of the combined normal distribution. This parameter characterizes the extent to which all possible A-B grid paths are spread out on either side of the sightline. Substituting the parameters of (9) and (10) into the formula for σ_{AB}^2 in (11), and taking the square root of both sides, we obtain the result in (13) below. The last step is based on the fact that $n(N-n)$ reaches its maximum value at $n = N/2$.

$$\sigma_{\mathrm{AB}} = \sqrt{\frac{\sigma_{\mathrm{A}}^2 \sigma_{\mathrm{B}}^2}{\sigma_{\mathrm{A}}^2 + \sigma_{\mathrm{B}}^2}} = \frac{1}{2}\sqrt{\frac{n(N-n)}{N}} \leq \frac{\sqrt{N}}{4} \tag{13}$$

Suppose now that the grid spacing is repeatedly halved by inserting new vertices among the existing vertices. After η such subdivisions, the grid resolution is increased by a factor of $\rho = 2^{\eta}$. Point B is then found at $[\rho N, \rho K]$, and all features of the map grow linearly with ρ when measured in grid spacings. In particular, there are now ρ times as many grid spacings between the clear sightline and some obstacle on one side. Yet the maximum standard deviation in (13) increases only from $\sqrt{N}/4$ to $\sqrt{\rho N}/4$ grid spacings, or by a factor of $\sqrt{\rho}$. This implies that after a sufficient number of subdivisions, only an arbitrarily small faction of all possible A-B grid paths will deviate from the sightline to the extent that they are blocked by the obstacle. Thus in the limit, obstacles at any distance from the sightline have no impact on the resulting central path. This theory is tested in Figure 18, where a set of central paths can be seen converging on a clear sightline despite a nearby obstacle spanning its length.

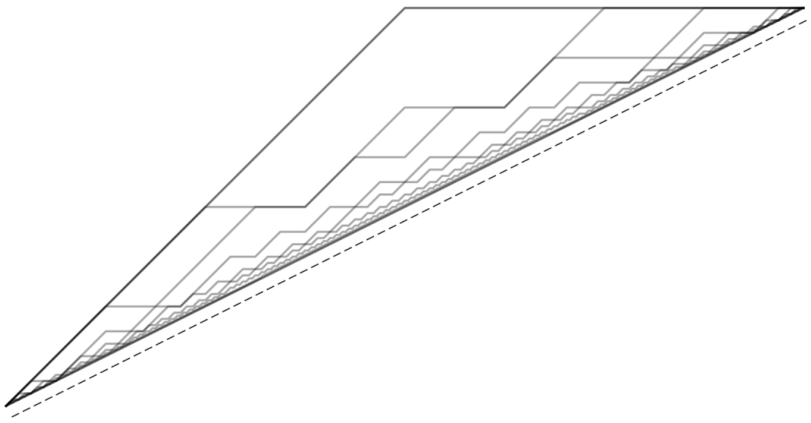


Figure 18: A plot of 11 central paths (solid lines) between $[0, 0]$ and $[64\delta, 32\delta]$, where δ is the distance to an obstacle (dashed line) running parallel to the sightline (not rendered). The grid spacing is 32δ for the first path, and is divided by 2 for each subsequent path. The final path (closest to the sightline) has a grid spacing of $\delta/32$. At resolutions for which there were multiple central paths, only the furthest from the sightline is plotted.

Since our theory pertains to the limiting behavior of central paths as the grid spacing approaches zero, we should also consider how obstacles may affect central paths in realistic scenarios where the grid spacing is fixed. The example in Figure 19 attempts to recreate the worst-case scenario in Figure 17, but using an actual architectural model with a reasonable

grid spacing of 25 cm. As expected, a wall that is not aligned with the grid induces a bulge in a central path running alongside it. Yet the bulge is arguably quite subtle. By contrast, a grid path produced by a typical Dijkstra or A* implementation using the same endpoints might well pass through point P in the figure. The central path is significantly more direct even in this example, and in general the bulging effect tends to go unnoticed.

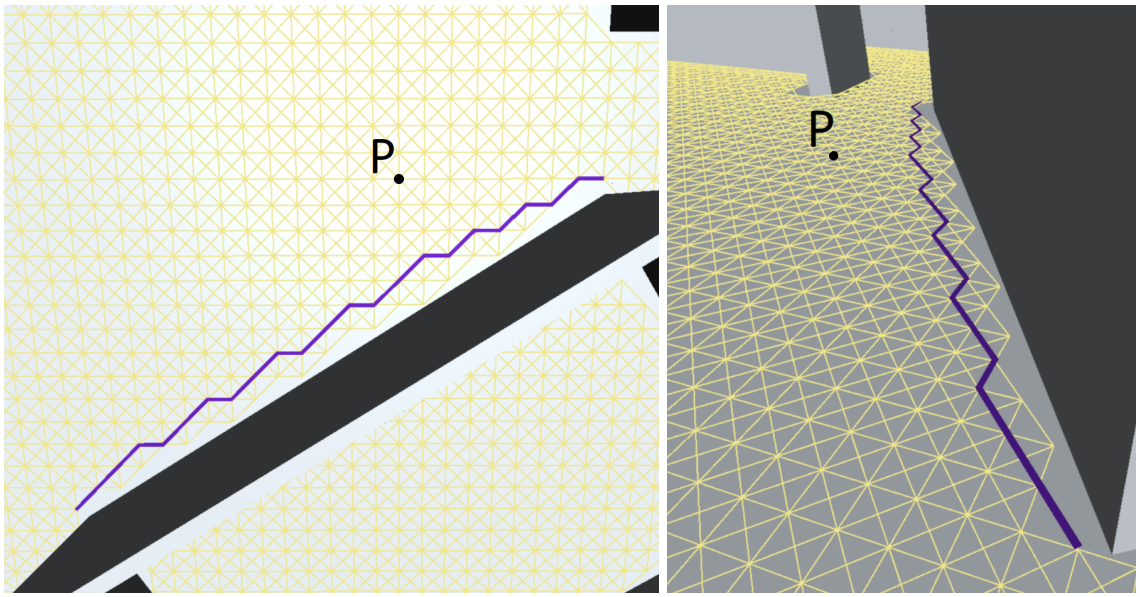


Figure 19: A central path traveling alongside a wall. Shown from different perspectives, the obstacle induces only a subtle bulge. Vertex P indicates how far a shortest grid path with the same endpoints can stray from the line-of-sight trajectory.

4.3 Convergence on Direct Paths

The convergence theory in Section 4.2 pertains only to scenarios in which a sightline exists between the endpoints of a desired path. Even in those cases, the sightline must be clear of obstacles within some positive distance δ on both sides. We now propose a generalization of the theory that overcomes these limitations and applies when paths are forced to veer around obstacles. Recall from Section 2.1 that a direct path is one that must follow any sightline between any pair of points on the path. We conjecture that if the central path approach is applied to the general 2D path planning problem, then reapplied using successively shorter grid spacings, the resulting central paths will converge on a direct path. In essence, central paths are direct in the limit.

Since the notion of convergence can have multiple interpretations in the context of paths, we base our conjecture on a measure of path similarity devised by Fréchet (1906). The Fréchet distance $\delta_{\mathrm{F}}(s,t)$ between paths s and t can be understood as the minimum length of leash needed to connect a human traversing s to a dog traversing t (Alt & Godau, 1995). The human and the dog are allowed to independently vary their speeds, but are not allowed to go backwards. We say that a sequence of paths s_n converges on path t if

$\delta_{\mathrm{F}}(s_n, t) \to 0$ as $n \to \infty$. We also introduce a deviation-from-directness metric that accounts for the fact there can be multiple direct paths between a pair of endpoints. Defined in (14), the directness deviation $\delta_{\mathrm{D}}(s)$ of path s is the minimum Fréchet distance between s and any path $t \in \mathcal{D}_s$, where $\mathcal{D}_s$ is the set of all direct paths with the same endpoints as s.

$$\delta_{\mathrm{D}}(s) = \min_{t \in \mathcal{D}_s}(\delta_{\mathrm{F}}(s, t)) \tag{14}$$

We conjecture that central paths converge on direct paths regardless of whether obstacle geometry is continuous, as described in Section 2.1, or grid-based, as described in Section 2.3. In either case, we assume that the endpoints A and B of the desired path are vertices on a standard rectangular or triangular 2D grid of spacing D, which is overlaid on the environment. To induce convergence, the grid spacing is repeatedly halved by inserting new vertices among the existing vertices. After η such subdivisions, the grid spacing is $D/2^{\eta}$. The navigation grid at each resolution level is constructed as outlined in Section 2.3, where moves between neighboring vertices are allowed if and only if there is a straight-line path between them. We assume that for sufficiently large η, any topology that contains a direct path also contains at least one grid path.

The above sets up a path planning problem for each $\eta \in \{0, 1, 2, 3, ...\}$. Applying the central path approach will yield a solution $s \in \mathcal{C}_\eta$, where $\mathcal{C}_\eta$ is the set of all possible central grid paths between A and B at resolution level η. The upper bound on the directness deviation of a central path is therefore the maximum value of $\delta_{\mathrm{D}}(s)$ over all of these possible solutions. The conjecture that central paths converge on direct paths can thus be formally stated as the property that this upper bound approaches zero as $\eta \to \infty$.

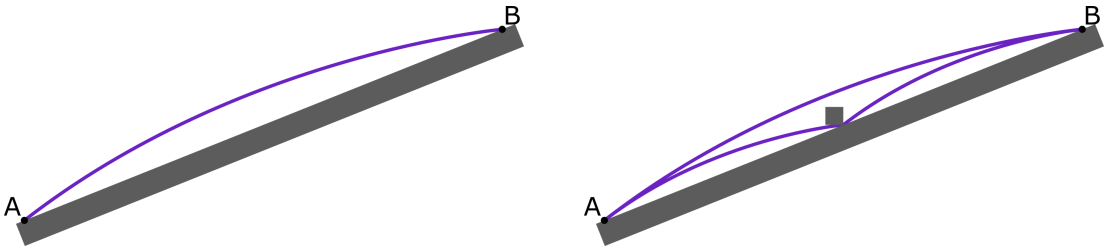
$$\lim_{\eta \to \infty} \left(\max_{s \in \mathcal{C}_\eta}(\delta_{\mathrm{D}}(s)) \right) = 0 \tag{15}$$

While a formal proof of (15) is a challenge for future research, we believe the conjecture holds for all 2D path planning scenarios that satisfy our assumptions. Our reasoning is that if there is a sightline between any two points P and Q on a central path, and if that sightline is not part of the path, then that portion of the central path should prove unstable as the grid spacing decreases. Specifically, we expect at least one of the following three effects to occur:

- Centralization effect: If P and Q are on an obstacle-free stretch of the path, then the distribution of traversal counts along the entire stretch should increasingly conform with the normal approximation based on the central limit theorem; as a result, that stretch of the central path will converge on a sightline.
- Attraction effect: If P and Q are on a stretch of the path that veers around a convex corner, a convex curve, or in some cases a straight surface of an obstacle, then a concentration of sightlines on the inside of the turn will pull the central path closer to the obstacle.
- Shortcutting effect: If P and Q are mutually visible through an unutilized shortcut between obstacles, then an increase in the traversal counts along the sightline will cause the path to be rerouted through the shortcut at a sufficiently fine grid resolution.

The centralization and attraction effects cause central paths to converge on taut paths, whereas the shortcutting effect causes central paths to transition to topologies for which these taut paths are also direct paths. Note that central paths do not necessarily converge on shortest paths. Even at extremely fine grid resolutions, shortest grid paths and shortest smooth paths may have different topologies. Figure 2 shows a case where central paths on an 8-neighbor grid may converge on a direct path that is not a shortest path.

The remainder of this section focuses on the special cases illustrated in Figure 20, which serve to illuminate the nuances of the conjecture. We use the $[n, k]$ notation of Section 4.2, where vertex A is at $[0, 0]$ and vertex B is at $[N, K]$. We assume $0 < K < N$.



(a) An obstacle spanning a sightline (b) Two obstacles creating a single-point gap

Figure 20: Two scenarios in which obstacles contacting a sightline will block most of the possible shortest grid paths between A and B. The curves illustrate possible topologies for central paths.

We first consider the case in Figure 20a, where the A-B sightline is clear on one side and blocked on the other. As illustrated, the obstacle will induce a bulge in any central path from A to B. The scenario is similar to the ones discussed in Section 4.2, except in this case there is no clearance between the obstacle and the sightline. The question is whether central paths will still converge on the sightline as the grid spacing approaches zero.

It was argued in Sections 4.1 and 4.2 that the fraction of unobstructed shortest grid paths, or equivalently the grid-based visibility score, converges on 1 if the sightline is clear of obstacles within some positive distance δ on both sides. Similarly, the visibility score converges on 0 if any obstacle crosses the sightline. If a sightline is merely contacted by an obstacle, however, the theory in Sections 4.1 and 4.2 does not predict how the visibility scores will converge. Such scenarios must instead be examined on a case-by-case basis.

The grid-based visibility score for Figure 20a can be estimated by observing its relationship to the generalized ballot problem (Takács, 1962). In simplified terms, ballot problems are concerned with the probability ψ that the ratio of votes for two candidates never falls below some constant κ at any point during the ballot counting process. It is known that this probability can be interpreted as the fraction of shortest grid paths that never pass below a line L (Humphreys, 2010). If κ in the ballot problem is exactly the ratio of the total number of votes for each candidate, then L is the sightline in our scenario and ψ is the visibility score. Meng (2009) reports upper and lower bounds for ψ in the generalized ballot problem, and one can use the lower bound as an estimate $\hat{\psi}_N$ for the visibility score in our scenario.

$$\hat{\psi}_N = \frac{1}{N} \tag{16}$$

The proof of (16) is based on a technique first applied to the ballot problem by Dvoretzky and Motzkin (1947). Consider any shortest grid path s from A to B, ignoring the obstacle. If s passes below the sightline at any point, then select some vertex $[n, k]$ on the path that is at the maximum distance below the sightline in the perpendicular direction. Now construct a cyclic permutation of s by (1) translating the entire path such that the point at $[n, k]$ is moved to $[0, 0]$; (2) trimming off the first n moves that are now between $[-n, -k]$ and $[0, 0]$; and (3) inserting those same n moves in order onto the end of the path. It is easy to see that this cyclic permutation is also a shortest grid path from A to B, but one that never passes below the sightline. It follows that at least one of the N cyclic permutations of any possible shortest grid path s will not be blocked by the obstacle, and hence $1/N$ is a lower bound of ψ. Meng (2009) also provides an upper bound of ψ and an associated proof, and it can be shown that the upper bound is strictly less than double the lower bound for our conditions. It is therefore clear that if the sightline from the source to $[N, K]$ is completely blocked on one side, the visibility score converges on 0.

Because the obstacle in Figure 20a blocks the vast majority of possible shortest grid paths at fine resolutions, we may not invoke the argument of Section 4.2 that the obstacle becomes increasingly irrelevant as the grid spacing approaches zero. Yet the conjecture in this section applies to all direct paths, including any sightline that is contacted but not blocked by an obstacle. As the grid spacing approaches zero, the attraction effect should steadily reduce the bulge illustrated in Figure 20a and cause central paths to converge on the sightline.

Finally, we consider the special case illustrated in Figure 20b. Here the previous example is augmented with a square obstacle of small but positive width δ, placed such that one of its corners creates a single-point gap at the midpoint of the sightline.

It was stated in Section 2.1 that paths may or may not be permitted to pass through single-point gaps, and that the theory presented in this paper should apply regardless. If passage through these gaps is prohibited, then all paths follow the outer topology illustrated in Figure 20b. The straight line from A to B is neither a path nor a sightline in that case, and consequently the only direct path is the taut path with a sharp bend at the outermost corner of the square obstacle. As the grid spacing is repeatedly halved, the attraction effect will cause the distance between successive central paths and this convex corner to converge on zero. At the same time, the centralization effect will cause these central paths to converge on the sightlines on either side.

If paths are permitted to traverse single-point gaps, then the straight line from A to B is both a direct path and a sightline. The taut path on the outside of the square is then disqualified from being a direct path, since it does not utilize the sightline. A technicality arises here as to whether the single-point gap is positioned such that any grid paths are able to go through. Due to its location, the gap in this scenario is accessible to grid-based solutions, and so the conjecture applies. The conjecture suggests that as the grid spacing is repeatedly halved, the number of shortest grid paths that pass through the gap will grow large compared with the number that go around the outside of the square obstacle. Thus even if central grid paths initially follow the outer topology, the shortcutting effect will eventually reroute them through the gap. The attraction effect will thereafter cause the central paths to converge on the sightline. We believe these effects will indeed occur, and submit that the generalized ballot theorem may be useful in proving this point.

5. Discussion and Conclusions

We introduced central grid path planning, a grid-based navigation approach in which Dijkstra's algorithm or A* is extended with a path counting step to produce relatively direct shortest grid paths. The approach avoids line-of-sight checks between non-neighboring vertices, yet still approximates straight path trajectories of arbitrary length and direction. We also proposed the notion of a direct path, and argued that central paths converge on clear sightlines and likely converge on direct paths as the grid spacing approaches zero.

The central path approach represents a natural solution to the problem of selecting a relatively direct path from the multitude of shortest grid paths that typically exist between two endpoints. Even if a smooth path is ultimately desired, generating and then smoothing a central grid path will yield more optimal results on average than generating and then smoothing an arbitrarily selected shortest grid path. We submit that for a typical application involving navigation on a grid-based environment, a logical strategy for upgrading a conventional A* implementation is as follows:

- If A* paths are noticeably indirect, and slightly longer runtimes can be tolerated, one can adopt Central A* by switching to an all-paths search and computing logarithms of path counts from both ends. Empirical results suggest that Central A* followed by smoothing could be an attractive alternative to the Theta* any-angle method. For the game maps included in our experiment, Smoothed Central A* achieved path lengths nearly as short as those of Theta* with runtimes closer to A*.

- If A* paths are of acceptable quality, but runtimes are somewhat too long, one can adopt a canonical path method such as Canonical A* or Jump Point Search by restricting vertex visits and expansions. An open question is whether a preliminary canonical path search could accelerate the computation of a central path.

- If A* is found lacking in both speed and quality, one might consider any-angle methods like Subgoal Graphs or Block A* that perform a precomputation on the environment to accelerate subsequent searches. Additional research would be needed to incorporate subgoals or blocks into a central path solver.

Future empirical studies could compare the central path approach to alternative path planning methods using a wider selection of maps, smoothing algorithms, and grid neighborhoods. The 16-neighborhood deserves particular attention, as it would improve the quality of the resulting central grid paths while also reducing the number of vertices for which log traversal counts need to be computed. Rather than using path length suboptimality as the sole measure of path quality, it is worth considering additional metrics to more fully capture the notion of relative directness. The directness deviation in Section 4.3 may be difficult to compute, but alternative metrics could include (a) the area or maximum deviation between the path being evaluated and the taut path with the same topology, or (b) the fraction of scenarios in which the topology of the evaluated path contains a shortest path. Whereas the conventional path length suboptimality metric requires shortest grid paths to be smoothed before they can be compared, these alternative metrics could be applied to grid paths and smooth paths alike.

Future theoretical work could include a formal proof of (15), the conjecture that central grid paths converge on direct paths provided the direct paths have topologies accessible to grid-based solutions. A first step might be to prove the conjecture for grid-based obstacle geometry. Another line of investigation would be to generalize the theory of central grid paths to 3D environments and higher dimensional spaces.

The finding that linear grid-based visibility scores can also be computed by path counting reveals that the existing visibility approach and the proposed navigation approach share a common theoretical basis. The relationship between the approaches is striking, as linear grid-based visibility was originally (a) based on implicit level set geometry instead of explicit geometry; (b) derived from differential equations instead of graph theory; (c) formulated using cardinal neighbors only instead of a variety of grid neighborhoods; (d) specified using normalized direction vectors instead of integer coefficients; and (e) validated using numerical analysis instead of probability theory. Differences in conventions may obscure a number of relationships between level set methods and methods derived from the concept of a grid path. For example, it is common practice in the level set community to perform path planning by solving the Eikonal equation on a 4-neighbor grid. When the method is modified to use the 8-neighborhood (Danielsson & Lin, 2003), the resulting formula is equivalent to one of the interpolations performed by Field D*.

We close with a few remarks about the use of the word "central" in naming the new grid-based navigation approach and the resulting paths. First, there is no intended connection between the central paths described here and those in linear programming and optimization (Nemirovski & Todd, 2008). The central paths in this work should therefore be referred to as "central grid paths" if the context is not clear. Our work is related, however, to the concept of centrality in graph theory (Borgatti & Everett, 2006), particularly *betweenness centrality* that quantifies the extent to which a vertex is between all others (Freeman, 1977). Betweenness centrality is computed using traversal counts similar to those in this paper, except that the shortest paths are between all pairs of vertices in a graph. Our use of the word "central" was also motivated by the central limit theorem, which clarifies how the simple counting procedure of Pascal's triangle can make straight lines, of any angle, appear on a grid.

Acknowledgments

We sincerely thank the anonymous reviewers, whose insightful feedback and tireless attention to detail helped us significantly improve the quality and correctness of this paper. Any remaining errors are of course our own. We also thank Nigel Morris, our colleague at Autodesk Research who introduced us to the level set method for computing visibility.

References

Alt, H., & Godau, M. (1995). Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, *5*(1), 75–91.

Bailey, J., Tovey, C., Uras, T., Koenig, S., & Nash, A. (2015). Path planning on grids: The

effect of vertex placement on path length. In *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE)*, pp. 108–114.

Borgatti, S. P., & Everett, M. G. (2006). A graph-theoretic perspective on centrality. *Social Networks*, *28*(4), 466–484.

Botea, A., Bouzy, B., Buro, M., Bauckhage, C., & Nau, D. (2013). Pathfinding in games. In *Artificial and Computational Intelligence in Games*, pp. 21–31. Dagstuhl Publishing.

Botea, A., Müller, M., & Schaeffer, J. (2004). Near optimal hierarchical path-finding. *Journal of Game Development*, *1*, 7–28.

Bromiley, P. A. (2014). Products and convolutions of gaussian probability density functions. Tech. rep., University of Manchester.

Cohen-Or, D., Chrysanthou, Y. L., & Silva, C. T.and Durand, F. (2003). A survey of visibility for walkthrough applications. *IEEE Transactions on Visualization and Computer Graphics*, *9*(3), 412–431.

Daniel, K., Nash, A., Koenig, S., & Felner, A. (2010). Theta*: Any-angle path planning on grids. *Journal of Artificial Intelligence Research*, *39*, 553–579.

Danielsson, P.-E., & Lin, Q. (2003). A modified fast marching method. In *Proceedings of the Scandinavian Conference on Image Analysis*, pp. 1154–1161.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, *1*, 269–271.

Dvoretzky, A., & Motzkin, T. (1947). A problem of arrangements. *Duke Mathematical Journal*, *14*(2), 305–313.

Ferguson, D., & Stentz, A. (2006). Using interpolation to improve path planning: The Field D* algorithm. *Journal of Field Robotics*, *23*(2), 79–101.

Fisher-Gewirtzman, D., Shashkov, A., & Doytsher, Y. (2013). Voxel based volumetric visibility analysis of urban environments. *Survey Review*, *45*(333), 451–461.

Fréchet, M. (1906). Sur quelques points du calcul fonctionnel. *Rendiconti del Circolo Matematico di Palermo*, *22*, 1–72.

Freeman, L. C. (1977). A set of measures of centrality based on betweenness. *Sociometry*, *40*(1), 35–41.

Ghosh, S. K. (2007). *Visibility Algorithms in the Plane*. New York, NY: Cambridge University Press.

Goldstein, R., Breslav, S., Walmsley, K., & Khan, A. (2020). SpaceAnalysis: A tool for pathfinding, visibility, and acoustics analyses in generative design workflows. In *Proceedings of the Symposium on Simulation for Architecture and Urban Design (SimAUD)*.

Han, J., Uras, T., & Koenig, S. (2020). Toward a string-pulling approach to path smoothing on grid graphs. In *Proceedings of the Symposium on Combinatorial Search (SoCS)*, pp. 106–110.

Harabor, D., & Grastien, A. (2011). Online graph pruning for pathfinding on grid maps. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 1114–1119.

Harabor, D., Grastien, A., Öz, D., & Aksakalli, V. (2016). Optimal any-angle pathfinding in practice. *Journal of Artificial Intelligence Research*, *56*, 89–118.

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, *4*(2), 100–107.

Humphreys, K. (2010). A history and a survey of lattice path enumeration. *Journal of Statistical Planning and Inference*, *140*(8), 2237–2254.

Kao, C.-Y., & Tsai, R. (2008). Properties of a level set algorithm for the visibility problems. *Journal of Scientific Computing*, *35*, 170–191.

Kraft, J. S., & Washington, L. C. (2014). *Elementary Number Theory*. Boca Raton, FL: CRC Press.

Lozano-Pérez, T., & Wesley, M. A. (1979). An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, *22*(10), 560–570.

Meng, D. (2009). Nice bounds for the generalized ballot problem. arXiv:0912.1999 [math.CO].

Morini, E., Rocchi, F., Avizzano, C. A., & Bergamasco, M. (2010). Visibility techniques applied to robotics. In *Proceedings of the International Symposium in Robot and Human Interactive Communication (ROMAN)*, pp. 367–372.

Nagy, D., Lau, D., Locke, J., Stoddart, J., Villaggi, L., Wang, R., Zhao, D., & Benjamin, D. (2017). Project Discover: An application of generative design for architectural space planning. In *Proceedings of the Symposium on Simulation for Architecture and Urban Design (SimAUD)*.

Nemirovski, A. S., & Todd, M. J. (2008). Interior-point methods for optimization. *Acta Numerica*, *17*, 191–234.

Noreen, I., Khan, A., & Habib, Z. (2016). Optimal path planning using RRT* based approaches: A survey and future directions. *International Journal of Advanced Computer Science and Applications*, *7*(11), 97–107.

Oh, S., & Leong, H. W. (2017). Edge N-level sparse visibility graphs: Fast optimal any-angle pathfinding using hierarchical taut paths. In *Proceedings of the International Symposium on Combinatorial Search (SoCS)*, pp. 64–72.

Pelechano, N., Allbeck, J. M., & Badler, N. I. (2008). *Virtual Crowds: Methods, Simulation, and Control (Synthesis Lectures on Computer Graphics and Animation)*. Morgan & Claypool Publishers.

Ravankar, A., Ravankar, A. A., Kobayashi, Y., Hoshino, Y., & Peng, C.-C. (2018). Path smoothing techniques in robot navigation: State-of-the-art, current and future challenges. *Sensors*, *18*(9), 3170.

Richardson, A., & Olson, E. (2011). Iterative path optimization for practical robot planning. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3881–3886.

Rivera, N., Hernández, C., Nicolás, H., & Baier, J. A. (2020). The 2^k neighborhoods for grid path planning. *Journal of Artificial Intelligence Research*, *67*, 81–113.

Roth, S. D. (1982). Ray casting for modeling solids. *Computer Graphics and Image Processing*, *18*(2), 109–144.

Sturtevant, N. R. (2012). Benchmarks for grid-based pathfinding. *Transactions on Computational Intelligence and AI in Games*, *4*(2), 144–148.

Sturtevant, N. R., & Rabin, S. (2016). Canonical orderings on grids. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 683–689.

Takács, L. (1962). A generalization of the ballot problem and its application in the theory of queues. *Journal of the American Statistical Association*, *57*(298), 327–337.

Tsai, Y.-H. R., Cheng, L.-T., Osher, S., Burchard, P., & Sapiro, G. (2004). Visibility and its dynamics in a PDE based implicit framework. *Journal of Computational Physics*, *199*(1), 260–290.

Turner, A., Doxa, M., & O'Sullivan, D. Penn, A. (2001). From isovists to visibility graphs: A methodology for the analysis of architectural space. *Environment and Planning B: Planning and Design*, *28*(1), 103–121.

Uras, T., & Koenig, S. (2015). An empirical comparison of any-angle path-planning algorithms. In *Proceedings of the Symposium on Combinatorial Search (SoCS)*, pp. 206–210, code available at: http://idm-lab.org/anyangle.

Uras, T., Koenig, S., & Hernandez, C. (2013). Subgoal graphs for optimal pathfinding in eight-neighbor grids. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 224–232.

Yap, P., Burch, N., Holte, R., & Schaeffer, J. (2011). Block A*: database-driven search with applications in any-angle path-planning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 120–125.