Mesh Navigation Through Jumping

Nick Roumimper

March 7, 2017

Abstract

The motion planning problem has been applied to a wide variety of fields and contexts. However, very few solutions to this problem combine multiple modes of motion. This report explores the motion planning problem in a three-dimensional environment where the character can not only walk on all available surfaces, but also jump between different surfaces. We take a more fundamental approach than much of the practically oriented work in the field. Starting from a set of physics-based axioms, we establish the definitions of minimal and optimal jumps between any two points. By extending these concepts to three-dimensional line segments, we define the jump link minimal velocity and jump link minimal arc length, which are two concrete heuristics for connectivity and optimality. Combined with special case definitions for projected edges, this allows us to provide a practical implementation of a jump link. We then provide notes on the results' usage in practice. The usage of jump links in a path planning algorithm is left for future work.

Contents

1 Introduction

Path planning is a complex and important problem in many digital applications, such as computer games and pedestrian simulations. This problem takes the form of a character seeking a path through the areas it can traverse (the navigable space) from the starting position to the goal position. The subproblem where the navigable space is planar, and the character is constrained to walking (e.g. the section of a floor that is not occupied by obstacles) has been studied intensively, leading to a wide selection of different methods to represent the navigable space and to find paths within this representation. How humans plan their paths, and what constitutes a realistic path to the human eye, has been a field of particular interest.

So far, path planning has been applied to many types of navigable spaces, such as environments with layers, levels and slopes (multi-layered environments, e.g. a parking garage), higher-dimensional spaces, and in conjunction with time. This thesis studies the problem of navigation in virtual worlds for characters that can jump between locations. Jumping allows the character to surmount obstacles, or reach previously unreachable locations, and incorporating this ability greatly increases the complexity of the path planning algorithm. The ability to jump is especially common in computer games. As a result, practical implementations of characters jumping to reach their goals already exist, but this specific domain has not yet been investigated in detail, nor from a strongly founded theoretical standpoint.

Starting from a representation of the navigable space (i.e. a navigation mesh), in this thesis we develop methods to annotate the navigation mesh with jumping information, and apply them to a variety of environments. The following sections explain in greater detail the motivation for the thesis, its goals and how this report is structured.

1.1 Motivation

Our definition of a (suitable) multi-layered environment will be established in more detail in the upcoming sections of this report. However, to provide a practical example, a highway crossover is multi-layered by nature; both the road above and below the crossover can be seen as their own walkable environment (i.e. considered as a navigable polygon), and if both polygons are projected onto the ground plane, they overlap at the bridging section.

A navigation mesh encodes how characters can navigate between regions in an environment. A wide variety of navigation meshes can be computed automatically from the original three-dimensional environment. However, many of these meshes assume that characters are constrained to stay on the walkable surface. By jumping, previously unreachable locations can become reachable, and drastic shortcuts may be realized. The problem here is that jumping can not easily be considered a natural extension of walking; another (third) dimension comes into play, and physical rules of momentum and ballistics apply. Annotating a multi-layered environment with realistic jump links has not yet been researched without relying on predetermined trajectories or highly restrictive assumptions.

For practical applications (e.g. games), when provided with a set of discrete, point-to-point jump arcs, current state-of-the-art programs are known to add these point-to-point arcs in specific locations based on a sampled collision volume of jump trajectories. The main problem with this approach is that the trajectories then become predetermined, which may lead to very unnatural-looking paths. Game designers often simply add a limited number of jump arcs manually, directing the character's actions while simultaneously severely restricting any applied path planning algorithm.

This thesis examines jumping in multi-layered environments from a continuous standpoint. Starting from an axiomatic approach to the mechanism of jumping, we establish metrics and methods to quantify the quality of the jump between any two points. Subsequently, we

expand and derive these concepts to the greater context of any two line segments in three-dimensional space. As a result, we aim to provide the theoretical bedrock for advances in the quality of complex paths in computer games and advanced crowd simulations.

1.2 Contributions

This thesis has the following five main contributions. First of all, we present an in-depth discussion of the literature currently available on the subject. We hereby motivate the line of reasoning that allows us to approach this issue from a purely theoretical standpoint.

Secondly, we provide a number of jump-related metrics, derived from an exhaustive examination of this mode of transportation. These allow us to narrow down the range of valid possibilities between any two edges, and reduce even the valid possibilities within this small band to specific optimal trajectories.

Thirdly, we expand the definition of these metrics into the combination of any two line segments in three-dimensional space. In doing so, we find ourselves capable of determining the optimal jump between any two segments.

Fourthly, we draw the necessary conclusions to make these metrics usable for practical implementation, based on our own experiences. The latter three items constitute the primary contributions of this work.

1.3 Structure

Section 2 contains the theoretical background for this thesis; it provides an overview of related work on path planning and navigation through jumping, and sets up the approach of the rest of the report. Section 3 sets up the contributions of this report in point-to-point cases, which is subsequently used in sections 4 and 5 to expand to three-dimensional space. Section 6 is dedicated to addressing the practical implementation of these metrics as far as possible within the scope of this report. Finally, section 7 summarizes the contents of this thesis, and section 8 points out where there is still room for improvement within these methods and aids future research by suggesting avenues of exploration for new projects.

2 Related Work

This section contains the general theoretical background for this report, as well as an overview of the currently available methods for mesh navigation through jumping. Subsection 2.1, "Motion and path planning", provides a summarized definition of said problem, and a basic overview of the structures and algorithms commonly used to solve it. Readers familiar with this field may skip this subsection. Subsection 2.2, "Previous work in the field", discusses the state of the art pertaining to navigation through jumping. This section is also used to motivate the direction of this report.

2.1 Motion and path planning

The *motion planning problem* was first introduced in the domain of robotics, under the name of "the Piano Mover's Problem" [23]. Its name derives from the common problem of moving an unwieldy object from one side of a room to another without colliding with other obstacles. Motion planning refers to determining the actions required to move from a starting configuration to a goal configuration. Here, all degrees of freedom need to be considered, commonly resulting in high-dimensional configuration spaces. The problem is defined as follows [19], and has in this form been found to be PSPACE-hard to solve [24]:

1. Given a world $\mathcal{W}$, which is either $\mathbb{R}^2$ or $\mathbb{R}^3$;
2. a semialgebraic region representing all obstacles $\mathcal{O} \subset \mathcal{W}$;
3. a semialgebraic robot $\mathcal{X} \subset \mathcal{W}$ representing the object to be moved;
4. the configuration space $\mathcal{C}$ based on all possible transformations of $\mathcal{X}$, partitioned into the subspace of free configurations called the navigable space $\mathcal{N}$ and the subspace of blocked configurations called the obstacle space $\mathcal{O}$;
5. an initial configuration $q_I \subset \mathcal{N}$;
6. a goal configuration $q_G \subset \mathcal{N}$;
7. find a continuous path within $\mathcal{N}$ from q_I to q_G, or correctly report that such a path does not exist.

In this thesis, we focus on a subset of motion planning problems, where the configuration space is restricted to the spatial dimensions; i.e. we assume the object to be moved does not have any pertinent dimensions of orientation. This simplifies the planning result from a set of motions to a path, and hence, we refer to this restricted version as the *path planning problem*. Solving path planning problems is an integral part of creating convincing artificial intelligence, since characters are often required to move independently from one place to the next within their virtual world in an "efficient" manner. The definition of "efficiency" varies greatly per domain, but a common definition aims to minimize the Euclidean length of the total path. This version of the motion planning problem, known as the *shortest path problem*, is the backbone of this report. In the following sections, we provide a general summary of the representations and algorithms used in solving path planning problems. Note that we will focus this examination on planning problems in position-based configuration spaces with holonomic movement. This subset of problems is the most prevalent and well-researched.

2.1.1 Representations of the navigable space

In order to compute a path in any virtual environment, such as the two-dimensional environment seen in Figure 1a, we require a representation of the navigable space. This representation contains the legal configurations for the character (i.e. free locations). In many applications, the representation of the navigable space can (largely) be computed prior to execution, while the path planning itself is done at runtime.

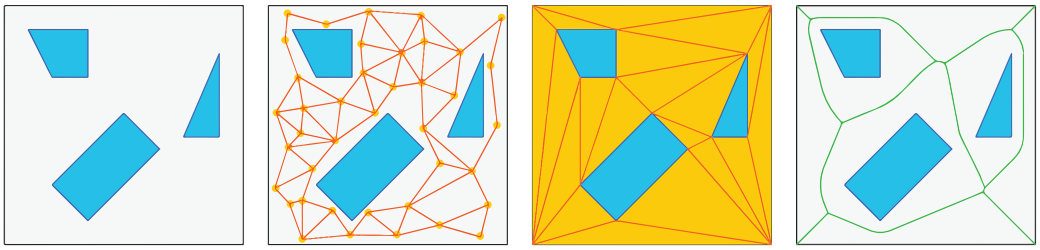

Figure 1: Representations of the navigable space. (a) A simple 2D environment with polygonal obstacles. (b) A sampling-based roadmap of the navigable space. (c) A triangulation. (d) The medial axis.

A common representation method involves sampling the available configurations, either regularly, (semi-)randomly, or incrementally (see [19] for a treatment). Samples are stored and linked to their neighbours in a *navigation graph*, which allows any graph planning algorithm to use it, presuming that the start and end position can be linked to the graph. One of the more commonly used methods is known as the *Probabilistic Roadmap* (PRM, [17], (b) in Figure 1), which branches out from a starting configuration into spatially close and valid configurations, and attempts to connect them to the navigation graph. Since these methods only approximate the navigable space, their navigation graphs may miss (connections between) configurations that are valid. Furthermore, the paths generated may be jagged and/or inefficient, and as a result, not look natural enough for virtual characters. [12]. However, due to their simple construction algorithms, their easy extension into higher dimensionalities and adjustable computational complexity, these methods can be applied to a wide variety of fields, a prime example being protein folding problems [1].

Oftentimes for configuration spaces limited to the spatial dimensions, we can use data structures which describe the navigable space in full, and can (preferably) be assembled automatically. For example, a *constrained Delaunay triangulation* (CDT, [6], see (c) in Figure 1 for a general triangulation example) subdivides the navigable space into triangles, allowing simple search operations on the triangulation's dual graph. Another strongly related structure is the *Voronoi diagram* ((d) in Figure 1), or its commonly used subset, the *medial axis* [3]. The medial axis of the navigable space consists of all points equidistant to two or more obstacles. Many of these translate to edges (for two obstacles) or vertices (for more obstacles) in the Voronoi diagram. These both fully cover the navigable space, and have been successfully applied to motion planning ([15], [2]).

Navigation meshes consist of a collection of regions that describe the space, combined with a graph that describes the connections between these regions. The *Explicit Corridor Map* (ECM, [11]), a prime example, consists of a Voronoi diagram annotated with event points that refer to the obstacles that shape that Voronoi edge. When a path is planned with depth-first search, a corridor (i.e. "hallway") is computed through the Voronoi diagram, which is triangulated and then planned inside with the highly efficient "funnel" algorithm.

However, since navigation meshes operate on a condensed representation rather than a complete one, they share the weakness that the resulting paths may still be inefficient. The *visibility graph* is a well-studied data structure (though explicitly not a navigation mesh) that can be used to determine the shortest path exactly (see [8] for a treatment). This graph connects all pairs of vertices of the obstacles that can be connected by a straight line that is entirely in the navigable space (i.e. vertices that can "see each other"). Then, when connecting the start and goal position to all visible vertices in the visibility graph, the shortest path in the graph is also the Euclidean shortest path.

The *visibility–voronoi complex* [28] is a hybrid approach consisting of a visibility-based complex, based on expanding all obstacles outward by any clearance distance value, and the Voronoi diagram in the original space combined. This complex can be used to determine

the shortest path with any desired clearance from the obstacles (by choosing paths from the visibility complex), and also allows for trade-offs when less clearance can lead to significant short-cuts (by choosing paths from the Voronoi diagram). However, this method requires $O(n^2)$ space during construction and exact number types, and may therefore not be feasible for the chosen application. There is a wide variety of meshes and mesh hybrids across the spectrum between precision and usability, showing that choosing a navigation mesh is hardly an uninvolved task. For a comparative study on navigation meshes, refer to the work by van Toll et al. [25]

2.1.2 Searching the navigation structure

When we have a representation of the navigable space available, we do not yet have any path leading through it, let alone the shortest one. A representational structure commonly contains a weighted, undirected graph that connects points in the navigable space, as seen in the previous section. Hence, graph search algorithms are applied to all of the above methods.

Dijkstra's algorithm [9], a classic result in graph search algorithms, finds the shortest path from the starting vertex to the goal vertex (or, in fact, any vertex in the graph). The search algorithm iteratively expands the connections from the vertex with the shortest distance to the starting vertex, and corrects encountered paths that are suboptimal. Over time, all vertices contain references that refer backwards along the shortest path to the starting vertex. The running time of this algorithm was later improved by storing and retrieving the yet to be explored vertices in a sorted Fibonacci heap, based on their distance to the start vertex [10].

Elaborating on the improvements made by ordering the vertices in a heap, Dijkstra's algorithm is extended to the *A*-algorithm* by changing the heuristic used to sort these vertices [14]. In the A*-algorithm (and similarly in many variants and extensions), the distance to the start vertex is added to a heuristically determined value, resulting in the sorting value. The heuristic can be manually chosen depending on the application, but as long as the added value does not overestimate the actual length of the path (i.e. it is "admissible"), A* will find the optimal path to the goal vertex. Choosing a suitable heuristic can result in far faster path finding, and even overestimating heuristics are used to find suboptimal paths quickly.

2.2 Previous work on navigation through jumping

In this section, we summarise the previous work in the field on navigation through jumping, both in practical applications and in scientific publications. While summarising, we highlight the gaps in the current research, and distill said gaps into the motivation of our report. We have separated the previous work according to whether or not the methods explicitly annotate the navigation mesh, in order to highlight the distinctions between the approaches.

2.2.1 Methods without jump annotations

We start by discussing the path planning algorithms that incorporate jumping without annotating their navigation structure. These methods do not consider jumping structurally as a method to significantly shorten path length, but rather as a tool to bridge gaps or obstacles of certain sizes. As a result, the jump capabilities are limited by factors such as the available animations or a preset jumping range.

This approach is most apparent in the work by Lau & Kuffner [18], which is one of the first path planning methods to address navigation through jumping. Here, a finite state machine (FSM) is manually assembled beforehand out of the available animations that describes what transitions between animations are valid, i.e. look as designed. The planning algorithm then iteratively expands a search tree from the starting position in the general direction of the goal area, until positions are found therein, and the shortest found path is selected. The provided example FSM includes a single jumping animation that can traverse

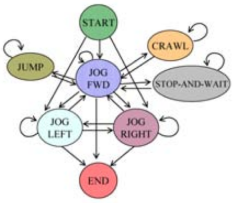

Figure 2: Image from [18], that displays the finite state machine that connects all moves.

obstacles up to a certain height, so jumping is possible, but very limited. More jumping animations could be incorporated, but the complexity of the search tree quickly becomes prohibitive. This method has been developed with smooth animation in mind, and is not suitable for dynamic character movement.

The method by Kapadia et al. [16] provides a more flexible representation of jumping. The concept of a planning hierarchy, where the path is refined gradually at different levels of detail, is the method's main contribution. The hierarchy suggested in the paper starts with an indicative route, based on the dual graph of the triangulation of the navigation mesh. This route is then altered based on dynamic unreachabilities on the second level, and the subpaths ("tunnels") are further refined on the third level. Finally, on the fourth level, the movement of dynamic obstacles is taken into account and final subpaths are determined.

It is there that jumping is used to surmount obstacles and gaps, adjusting the path slightly at a fairly high cost. Although this is one of the most dynamic and continuous implementations of jumping as of yet, the jumping adjustments do not alter the path significantly (i.e. change the homotopic class), and moving the jumping ability up in the hierarchy requires a completely different navigation structure.

Finally, the method by Levine et al. [20], although it aims to solve planning in dynamic environments, provides the most meaningful implementation of jumping. This dynamic planning method is based on the assumption that the position of all objects (walkable spaces and obstacles) is known for any point in time. This knowledge allows the sampling of the planning space with landmarks, which are attached to their surface and move along with it. Based on the landmarks, the planning algorithm can assemble a search tree in space-time from the landmark closest to the start to the landmark closest to the goal. Since the distances and heights between landmarks may vary strongly, all of the character's movements are determined by motion controllers, whose parameters are determined through reinforcement learning in a preprocessing stage. Under the assumption that waiting is the lowest-cost motion controller, the search tree in space-time can be searched with A* for the optimal path up to sampling error.

Jumping is one of the motion controllers, used to overcome gaps and height differences. This capability is limited by the animation and the boundaries of the motion controller, but it can be adjusted within those bounds, making it flexible. Hence, it is very well-suited to predictable practical applications. Although the quality of the found path is provably high, the sampling error can be very large, and the planning complexity can become prohibitive when the sampling resolution is increased. Furthermore, it is left unclear how landmarks are handled that are over time intersected by the obstacle space.

2.2.2 Methods with jump annotations

We now move on to methods where jump annotations are added to the navigation structure, starting at simple methods and progressively discussing more refined representations. Here, the most basic representations are found in the field, implemented in game development kits such as Unity3D [27, 26] and CryEngine [7]. Both of these programs allow the user to

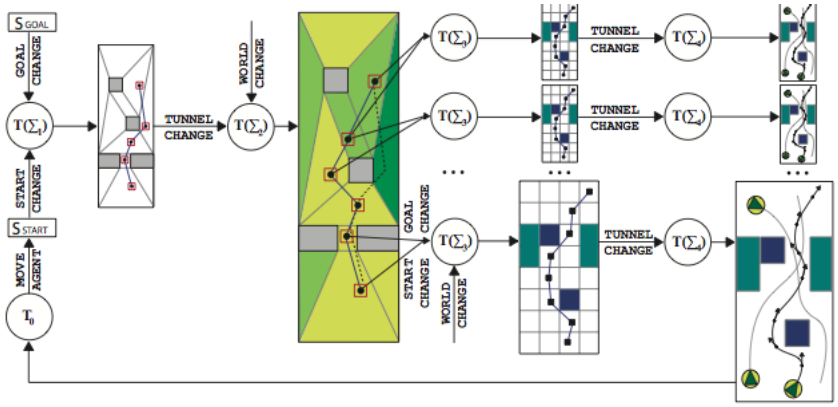

Figure 3: Image from [16], that illustrates the different path planning domains in its method.

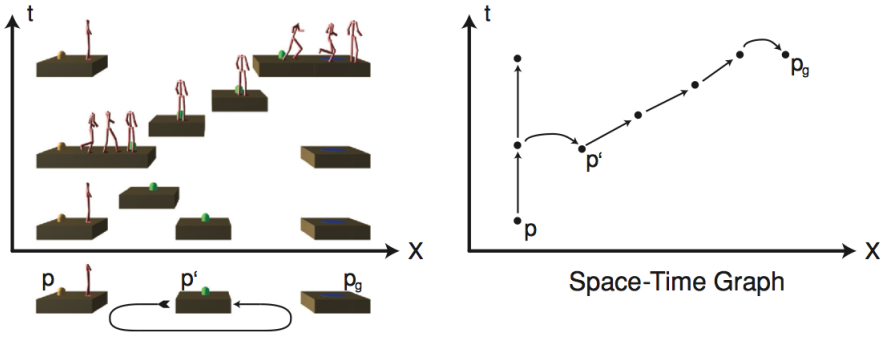

Figure 4: Image from [20], that illustrates its usage of space-time graphs in a two-dimensional setting.

add single jump arcs with a fixed start, end and trajectory to the navigation mesh. In a similar way, the CryEngine includes a specification of an AI Path [7], a path consisting of curves that can be added between spatially separated navigation meshes. These structures are intended for designers to manually increase the connectivity of their environments, and work well in practice; however, they have little value in terms of scientific rigor.

An example of a more flexible method can be found in the Smooth Movement Across Random Terrain (SMART) system, as implemented by Splash Damage for the 2011 shooter game Brink [13]. This game is centered around fast-paced free-running style action, and hence, the players have to be able to duck, jump over and jump across obstacles smoothly. To accomplish this, the navigation mesh is annotated in a preprocessing step with reachabilities, which connect two edges from separate sections that overlap when projected onto the ground plane. The players can subsequently run at these reachabilities, and based on factors such as the player's height and where the player is looking, the character jumps, mantles, vaults, wall hops or slides to move forward. This approach thusly adjusts the animation based on the appropriate action, allowing practical flexibility but indeterminate theoretical quality.

Returning to the realm of scientific publications, the jump representation takes a so far unexplored form in the work by Lopez et al. [21] This method is aimed at planning in changing environments, and represents the navigable space with collision volumes and walkable volumes that are recomputed on the fly. The primary volume used, the Accessibility Volume, connects the walkable surfaces between two objects, and is computed for every object by extruding the precomputed Accessibility Profile from the outer edges. The profile is constructed by sampling the limited range of jump trajectories from a single point, and taking the convex hull of these samples. To plan paths in the changing environment, probabilistic roadmaps (PRMs) are computed for the walkable surfaces; when the Accessi-

bility Volume of one object overlaps with the walkable surface of another, the path planning method seeks out two walkable nodes to connect and bridge the gap.

The advantage of this method is that the trajectory is dynamically specified from point to point, allowing for potentially very close to optimal paths containing natural-looking jumps. On the other hand, the usage of PRMs introduces all of their shortcomings: jagged and possibly inefficient paths, as well as missing valid connections, depending on the sampling density. The symptomatic jaggedness of the found paths is demonstrated quite clearly in the video accompanying this work. Also, the algorithm to determine the Accessibility Volume is particularly unwieldy, even for dynamic environments.

We find another PRM-based approach in the work by Campana et al. [5], wherein a point robot navigates a three-dimensional environment solely by jumping. In this system, each jump arc consists of a frictionless (i.e. symmetrical) parabola based on Newtonian physics. The set of admissible parabolas is bounded by four constraints: two constraints limiting the angles of takeoff and landing to prevent "slipping", one to limit the takeoff velocity to represent the robot's speed capacity, and one to limit the landing velocity to represent the robot's tolerance of impact forces. Any parabola in the band of trajectories that satisfies all four can be chosen during planning.

The jump representation for this method is created by randomly sampling a point from the environment. Based on the normal vector of the surface at that point, as well as the provided slipping constraint, we can construct a three-dimensional friction cone which applies to both takeoff and landing. As the paper proves, two of these points that can be connected with a valid parabola each have a neighbourhood of points that can be connected as well. As a result, a probabilistic roadmap planner that samples randomly from the environment will find a valid path between any two points if one exists, as running time tends to infinity.

The methods by Lopez et al. and Campana et al. find similar-looking paths. Whereas the former combines both walking and jumping motions through a fairly convoluted representation, the latter limits itself to jumping, but with a compact representation that lends itself to large environments. An extension of Campana's method with parallel walking connections between samples would be of particular interest. However, this would still not result in natural-looking motion. As the associated video demonstrates, many motions along any given path are extraneous, and thus, the paths found are of no demonstrable quality.

To conclude this survey, we discuss the most refined approach to adding and using jump annotations found so far. This approach is found in the Master's thesis by Sara Budde [4], as well as the Recast Navigation software by Mikko Mononen [22], and has likely been implemented in Unity3D [26]. (Licensing terms make this difficult to determine, but mr. Mononen is employed by Unity at the time of writing, and he has remarked on the similarities in an interview [4].) Note that the work by Sara Budde provides a more robust and flexible sampling approach, but the representation is structurally the same.

The algorithm iteratively selects and samples outer edges of the navigation mesh, where a predefined jump trajectory is projected as starting from each of these sample points. The trajectories that connect to the same walkable surface are considered together as a jump arc. To check the validity of such an arc, each of the separate trajectories is combined with the height of the character, and each of these "slices" is tested for collision with the surrounding obstacles. This may result in jump arcs being split up into any number of separate arcs. Every sufficiently wide jump arc that does not cause collisions becomes an annotation in the navigation mesh.

These jump annotations are highly suitable for practical purposes, can bridge any number of gaps, can easily cross obstacles and greatly increase mesh connectivity. However,

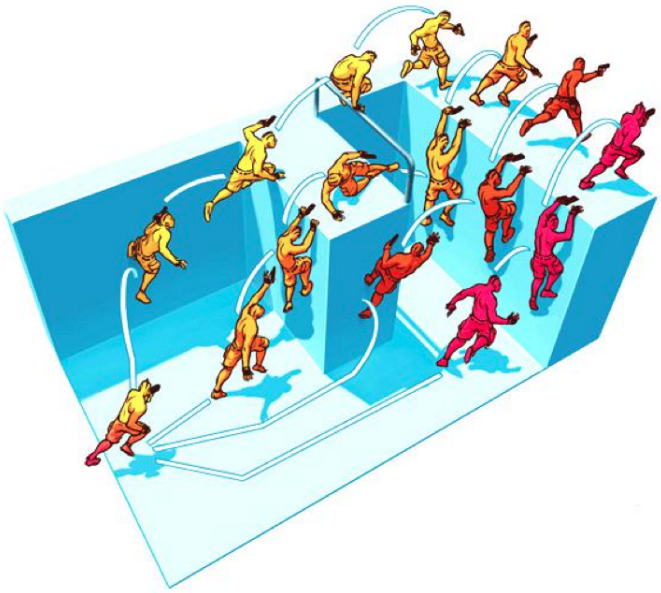

Figure 5: Image from [13], that shows the various SMART options for a geometrically simple environment.

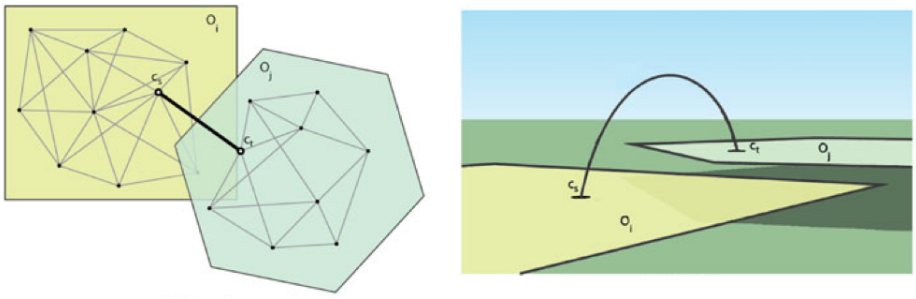

Figure 6: Image from [21], that illustrates how jump connections are established between PRMs in practice.

since they are essentially made up of collections of single jumping arcs, these annotations cannot be easily crossed diagonally or at varying heights. Similar to the single jump arcs, there are no guarantees on the quality of the paths planned.

2.2.3 Motivation of this research project

First of all, we here discuss the disadvantages of the methods that omit jump annotations for their path planning. If the mesh does not contain jump annotations, then a path planning algorithm that incorporates jumping can either use it as a highly local, and thereby small-scale and computationally manageable action, or attempt to plan (structurally significant) jumps while the algorithm is running.

Jumping at a local level, as found in the work by Kapadia et al. [16], has the distinct disadvantage that it does not meaningfully change the homotopic class of the found path. Although local jumps can be used to traverse small obstacles, which may be suitable for certain dynamic environments, the results are no different from planning when these obstacles are simply omitted. Essentially, such a path planning algorithm can choose to ”smooth over” small enough obstacles and gaps. Since jumping is essential rather than secondary to this report, this approach will not be sufficient here.

Planning larger jumps at runtime on a case-by-case basis, on the other hand, very quickly becomes computationally infeasible. Since the starting location of any jump should not

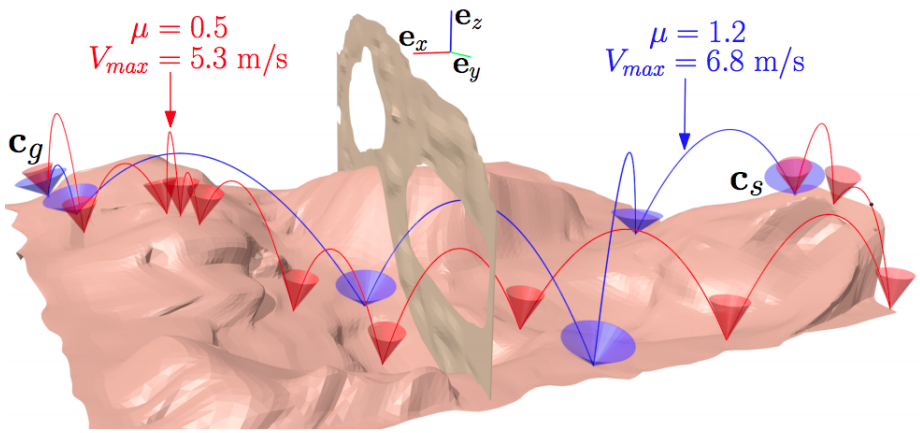

Figure 7: Image from [5], where the blue path has been planned with less stringent constraints than the red path.

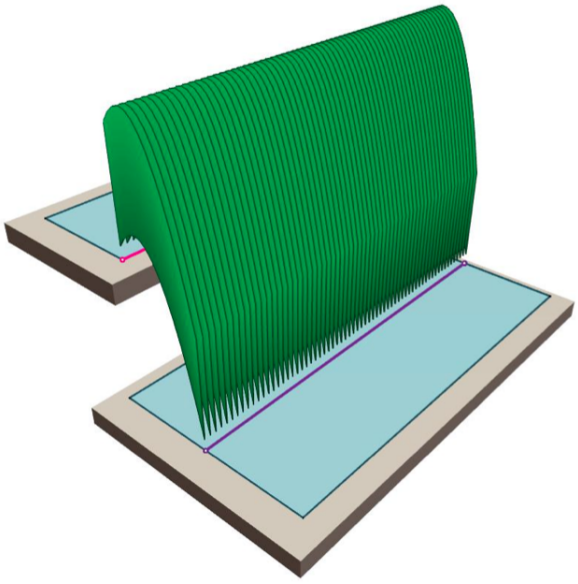

Figure 8: Image from [4], that shows how continuous representations are constructed from densely sampled collision slices.

reoccur in any remotely efficient path, reuse of previous results is very limited. Also, for the algorithm to connect all possible locations, each reachable location needs to be considered, which may well include the entire environment. Hence, each jump is a unique case that requires evaluation of (in the worst-case scenario) all navigable polygons. Attaching sufficiently advanced jump connections to the navigation mesh in an offline stage is simply a way to prepare these complex considerations for the online stage.

Secondly, from the examined methods that do use jump annotations, we can conclude that no representations have been defined as of yet that cover all (rationally) possible jump arcs. The closest contenders consist of the work by Lopez et al. [21] and the work by Budde [4], which use continuous annotations. We find that the representations generated by Budde are based on a single trajectory that is sampled with a predefined density. This keeps us from planning more efficient routes, regardless of context. On the other hand, the method by Lopez et al. may produce a fairly dense Accessibility Profile, but it is not guaranteed to be complete or have a smooth boundary. And even then, to establish actual jump connections, the method still requires random sampling of configurations that are within range, leading to further jagged and inefficient paths, or most damningly, valid paths not being found. An annotation structure that considers all valid trajectories is thus still sorely missing from the field.

Thirdly, although methods have been coined to place jump representations in the navigation mesh, these methods have not yet been applied to entirely continuous representations. The placement work by Mikko Mononen [22] and Budde provides an interesting theoretical base to work off of, but more exact placement methods are required in this case. Furthermore, the possible placement cases and degeneracies have so far not been formally described, so a description of such special cases contributes directly to this field.

3 Exploration of Jumping

Whereas the proceeding sections have explored the theory in and fundamental results of this field, we here move to providing our research.

In this section, we examine the idea of a single jump between two points, and apply it to connections between polygons in order to provide our definition of jump links. First, we make our assumptions relating to the mechanics of both moving and jumping. Secondly, we provide the equations from Newtonian physics we use to simulate jumps, which we treat as our mathematical axioms. Thirdly, we describe the jump space between a starting and a goal point, and define the concepts of minimal jumps, optimal jumps and the rational jump range. Fourthly, we interpolate this point-based definition between three-dimensional line segments, and examine the special cases that result from our axiomatic system. This provides us with our theoretical definition of jump links. Finally, we provide informal definitions of the heuristics we will define in the upcoming sections of this report.

3.1 Approach and terminology

Previously, we provided a number of broad-strokes overviews of the concepts central to this report. In order to provide the appropriate context for our work, we need to define the environment to which it applies, as well as the metrics that determine its success. Although many of the below definitions will not come directly into play in this section yet, we provide them here in advance and reference them when applicable.

Our chosen environment is a non-empty set of disjoint multi-layered walkable surfaces, where each surface is annotated with its own navigation mesh. To clarify, this means the following for any valid environment:

- The environment may contain only one walkable surface, or any number of walkable surfaces, as long as these surfaces don't (self-)overlap or (self-)intersect;
- When projected onto the ground plane, each walkable surface may intersect itself or any other surfaces any number of times;
- Each walkable surface consists of a collection of interconnected triangles in three-dimensional space;
- Each surface as a whole must be entirely walkable for a disk-shaped object with a non-zero radius, i.e. contains no perfectly vertical triangles or pairs of triangles connected by a single vertex;
- Each surface has been annotated with a complete navigation mesh that allows the planning of paths between any two points in that surface.

This latter constraint is particularly meaningful to our results; as seen in the related work, many different types of navigation meshes have been defined (or can be extended to become well-defined) for multi-layered walkable surfaces. This allows us to apply any of these methods to the subproblem of planning a walked path across a single surface. In this report, we provide the solution to the subproblem of meaningfully connecting walkable surfaces through jump links.

In order to remain consistent with most, if not nearly all available implementations of navigation meshes, as well as bound the complexity of the presented method, we restrict our path planning to holonomic motion. Although the jump trajectories we define in the next subsections are based on essential results from physics, we do not maintain or compute momentum explicitly when planning paths. Extending the method provided here to account for nonholonomic motion is left for future work.

As a consequence of the aforementioned, the metric used to gauge the quality of a path

is the sum of the Euclidean length of each of its component lines and curves, i.e. its total length from start to finish. As may be expected, a shorter path is considered better than a longer one. This suits both almost all navigation meshes, as well as holonomic motion, and applies to any piecewise assembled path curve (i.e. both line segments "on foot" and jump arcs "in the air").

We conclude this prerequisite section by clarifying certain terminology:

- The term "velocity" here applies to the three-dimensional vector of motion;
- The term "speed" here is used to refer to the scalar length of the velocity vector;
- The term "angle" here refers to the angle between the velocity vector and the ground plane.

The velocity vector for a jump between two given points is uniquely defined given both the speed and angle of the jump. In this situation, we therefore interchangeably refer to a jump's velocity and the combination of its speed and angle.

3.2 Mechanics

In order to describe and simulate jumps in a digital environment, we must first clearly define what "jumping" means in this context. Although it seems like an obvious concept, its simulated counterpart is limited by the simulation environment, as well as its purpose. We wish to find a definition that balances realism and computational complexity, with special emphasis on the latter, considering that we apply it across any number of ranges of jumps.

When jumping, we push ourselves away from the ground, launching from our start to our goal. In this thesis, we presume that the simulated environment includes a gravitational force, such that any character is drawn downward in the air at a constant rate of acceleration. (Hence, our results do not translate well to simulations of "extraterrestrial" environments, e.g. outer space.) From a physics standpoint, starting at "takeoff" the jumper traverses a trajectory based on said gravitational force, the jumping angle, the jumping force (which can also be described through the initial jumping speed) and the friction experienced along the trajectory, i.e. air drag.

Here, the air drag is by far the most complex force to simulate accurately; it depends on the density of the air (and therefore also on the temperature), as well as the speed of the jumper at any point in time, the cross sectional area of the jumper in the direction of motion and a drag coefficient value derived from the jumper's shape. If we were to include this full level of detail, the precomputation of paths would become computationally prohibitive. Even a limited representation has distinct disadvantages; this would prevent us from applying symmetry to simplify the representation, and/or require that the jump representations are recomputed depending on the character using them. This is not desirable behaviour for what is potentially already a computationally very expensive procedure. As a result, we will not be taking air drag into account when considering jump trajectories. This is left for a possible future implementation. To simulate how air drag strongly limits the distance traversed when jumping, the results can be made more realistic if required by reducing the maximal jump distance accordingly.

Since we omit air drag from our simulations, we also omit the possibility to alter one's trajectory in midair. Changes of configuration in the character might have influenced the cross sectional area and drag coefficient, changing the distance traversed, or even the angle of motion. Note that while changing the jump trajectory in midair is not in any way realistic, it is a commonly included mechanic in video games. Consider for example many platforming games, where players can interrupt or tweak the trajectory while moving. Although the method presented in this thesis could very well be applied to the domain of video games, we here choose realism over practicality of application, and indeed do not allow changes to the

trajectory after the start of the jump. This means that any jump trajectory, when projected onto the ground, must return a straight line segment.

We are now left with a trajectory influenced only by gravity, the jumping angle and the jumping speed. Since we presume that the gravitational constant has been provided, we can now exercise easily predictable control over the trajectory through the chosen angle and speed. In terms of physics, this method corresponds to the ballistic trajectory of a projectile when disregarding air drag. This situation has been well studied and can be easily typified through several standard formulas.

Our final observations provide reasonable limits on these two alterable parameters, as well as the range of trajectories. Concerning the speed, we must note that it is of course not possible to jump with negative speed, i.e. negative force. Also, for ease of reference, we assume that the maximum jump speed v_{max} has been defined for each character, as either a real value or "infinity". Considering the jumping angle, we note most importantly that we do not consider it possible to jump at a negative angle. With general ballistic trajectories, negative angles may apply when an object is launched downward. Although we do consider characters as being "launched" by their jump actions, we presume that they cannot push themselves downward off of the edge of a platform or through it, as this would be particularly unrealistic. We thereby limit the angle of jumps to values between 0 and π radians. (In upcoming sections, we will find that in practice only angles between 0 and $\frac{1}{2}\pi$ radians need to be considered.) It does remain possible to fall straight down off an edge - this is represented by "jumping" at any angle with zero speed.

Finally, we restrict the number of trajectories by requiring that any valid trajectory must have a zero or negative vertical speed when it reaches its intended goal. This means that the trajectory actually needs to "touch down" onto its goal, allowing the character to come to a standstill. In this simulation, we do not explicitly compute momentum, but we know that characters jumping up at a point with excessive force can in fact overshoot their target. To simply interrupt such a trajectory during the simulation at the desired goal point would unquestionably run counter to our goal of realism.

We thus conclude that we can represent jumping by a limited subset of ballistic trajectories without air drag or intermittent propulsion. This allows us to use the applicable equations in our simulations, and equally importantly, as the axiomatic basis for our definition of jump links. We therefore provide these equations in the next section.

3.3 Physics of jumping

In this section, we describe how we can create a jump trajectory, based on Newtonian physics equations. These describe ballistic trajectories in the plane from the starting position $(0, 0)$, where x refers to the horizontal direction and y refers to the vertical direction. Such a trajectory for a simple projectile (i.e. point mass) that is free from air drag is not resisted in the horizontal direction, and is only subject to the gravitational force g in the vertical direction. This is expressed as follows:

$$a_x = 0 \tag{1}$$

$$a_y = -g \tag{2}$$

where a_x is the horizontal acceleration, and a_y is the vertical acceleration. The velocity v along the trajectory is a combination of its horizontal and vertical component, v_x^2 and v_y^2 respectively, which can be expressed as:

$$v = \sqrt{v_x^2 + v_y^2} \tag{3}$$

The horizontal velocity does not change along the trajectory, but the vertical velocity is subject to the gravitational pull, as expressed in the following equations:

$$v_x = v_0 * cos(\theta) \tag{4}$$

$$v_y = v_0 * sin(\theta) - gt \tag{5}$$

where v_0 is the initial speed, θ is the angle of launch and t represents time. As a result, the horizontal and vertical displacement (x and y, respectively) are expressed as follows:

$$x = v_0 t * cos(\theta) \tag{6}$$

$$y = v_0 t * sin(\theta) - \frac{1}{2} g t^2 \tag{7}$$

Now, when we substitute out t with the formula based on x, the following equation is derived:

$$y = x * tan(\theta) - \frac{g x^2}{2(v * cos(\theta))^2} \tag{8}$$

The above equation is central to the following sections, where we will be utilizing a three-dimensional equivalent. Now, if we have both the starting point and (the relative position of) the goal point, we still require both v_0 and θ to define the trajectory. Through a separate derivation process, which we have omitted here, we find a formula that allows us to determine the angle required to reach a target coordinate (d_x, d_y) relative to the starting position $(0, 0)$:

$$\theta = arctan(\frac{v_0^2 \pm \sqrt{v_0^4 - g(g d_x^2 + 2 d_y v_0^2)}}{g d_x}) \tag{9}$$

We will also regularly be referring to the three-dimensional equivalent of the above equation in the upcoming sections. Finally, we provide several separately derived equations that we will only refer to once, but stem from the same background in physics. The following equation returns the range of the ballistic trajectory R, i.e. the horizontal distance the launched object has traversed when it once again reaches the height of its starting position, based on v_0, θ and g:

$$R = \frac{v_0^2 * sin(2\theta)}{g} \tag{10}$$

The following equation returns the maximum height of the ballistic trajectory h based on its range R:

$$h = \frac{R * tan(\theta)}{4} \tag{11}$$

Now that we have examined the base equations of ballistic physics in the plane, we can apply them to the simple case of jumping from one point to another, and derive the concepts crucial to our definition of the jump link representation.

3.4 Single jump concepts

In this section, we examine jumping from a predefined starting point to a predefined goal point in an unobstructed space, and define a minimal and optimal velocity (i.e. jumping speed and angle) for any two points. This provides us with rational bounds on the trajectory space for any two points, resulting in the definition of the rational jump space.

Note that from this section on, we provide the three-dimensional equivalents of the previously described functions, in a coordinate system where the positive y-direction is considered to be "up" (i.e. gravity is applied in the negative y-direction). This means we will be replacing the variable x with l_{xz}, where l_{xz} represents the current ground plane projected distance to the starting point, and the variable y with l_y, where l_y represents the y-value (i.e. height) at the corresponding point. When referring to the fixed location of the goal point relative to the starting point, we substitute d_x with d_{xz} and retain d_y.

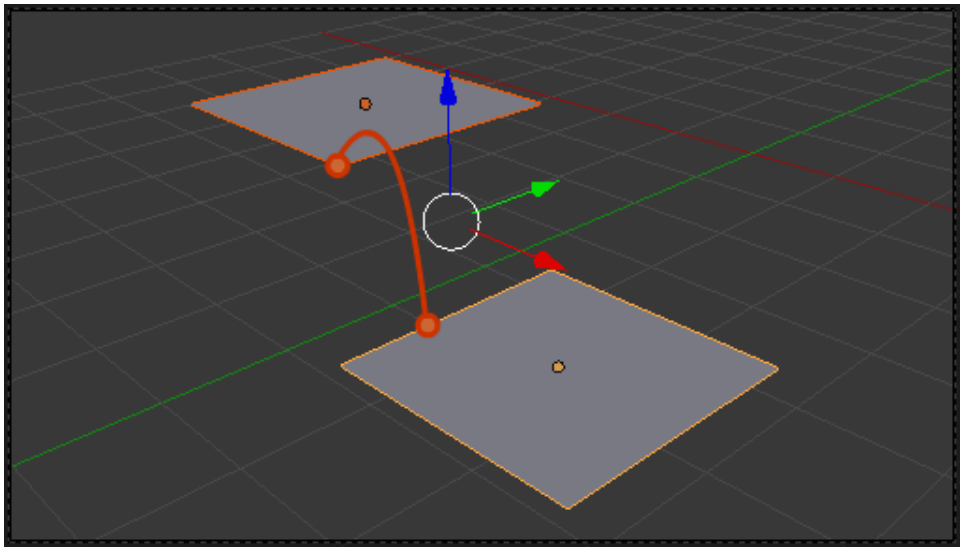

Figure 9: A simple sketch of a three-dimensional environment, with the axes included. The blue axis points in the positive y-direction, the red axis points in the positive x-direction and the green axis points in the positive z-direction. The provided trajectory is a sketch intended to clarify the perspective.

3.4.1 Minimal jump speed and associated angle

Given the locations of the start and goal of our desired jump, it is clear that not all chosen pairs of speeds and angles will result in a valid jump between the two. Any one pairing may overshoot or undershoot the target, and only very specific pairings relate to trajectories landing exactly at the goal point. Equation 9 provides us with a way to determine the associated angle when the speed has been provided. However, it also provides a way to define the minimal speed required to reach the goal from the starting position.

We refer back to Equation 9, and here provide its three-dimensional equivalent where v represents the jumping speed:

$$\theta = arctan(\frac{v^2 \pm \sqrt{v^4 - g(gd_{xz}^2 + 2d_y v^2)}}{gd_{xz}}) \tag{12}$$

For clarity, we note that d_{xz} represents the ground plane projected distance from the goal point to the starting point, and d_y represents the height difference between the goal and starting point. Consider, when we substitute $A = v^4 - g(gd_{xz}^2 + 2d_y v^2)$, the different cases corresponding to the different values of A. If A is negative, the above returns zero values; this means that the speed is insufficient to reach the target point, regardless of the chosen angle. If A is positive, the above returns two possible launch angles. The greater angle will result in a higher arc, but both trajectories reach (x_t, y_t).

However, the final case is of particular interest. When A is equal to zero, which occurs for a single specific speed, the above returns only one associated angle. There is, so to speak, no leeway in how the jump has to be made. This implies that the jump speed is minimal for that particular target coordinate. We refer to this minimal speed as v_{min}. If we equate A to zero, we find that we can determine the minimal speed directly from d_{xz} and d_y, as we demonstrate below:

$$v_{min}^4 - g(gd_{xz}^2 + 2d_y v_{min}^2) = 0$$

$$v_{min}^4 - 2gd_y v_{min}^2 - g^2 d_{xy}^2 = 0$$

substituting: $a = v_{min}^2$ $$a^2 - 2gd_y a - g^2 d_{xz}^2 = 0$$

application of the quadratic formula $$a = \frac{2gd_y \pm \sqrt{4g^2 d_y^2 + 4g^2 d_{xz}^2}}{2}$$

unsubstituting $$v_{min} = \sqrt{\frac{2gd_y \pm \sqrt{4g^2 d_y^2 + 4g^2 d_{xz}^2}}{2}}$$

$$v_{min} = \sqrt{gd_y \pm \frac{\sqrt{4g^2 * (d_y^2 + d_{xz}^2)}}{2}}$$

$$v_{min} = \sqrt{gd_y \pm \frac{2g * \sqrt{d_y^2 + d_{xz}^2}}{2}}$$

$$v_{min} = \sqrt{gd_y \pm g\sqrt{d_y^2 + d_{xz}^2}}$$

$$v_{min} = \sqrt{g}\sqrt{d_y \pm \sqrt{d_y^2 + d_{xz}^2}}$$

Although at first glance, this equation appears to be able to return two valid results, in practice it will return exactly one. Since we consider irrational speeds invalid in practice, the negative case above does not return a meaningful result in any case. We highlight the negative case to demonstrate this point:

$$v_{min} = \sqrt{g}\sqrt{d_y - \sqrt{d_y^2 + d_{xz}^2}}$$

Now, the second square root only returns a valid result when $d_y - \sqrt{d_y^2 + d_{xz}^2}$ is zero or positive, and similarly, $\sqrt{d_y^2 + d_{xz}^2}$ always returns a zero or positive value. In fact, whenever $d_{xz}^2 > 0$, $\sqrt{d_y^2 + d_{xz}^2} > |d_y|$, making $d_y - \sqrt{d_y^2 + d_{xz}^2}$ negative and the result invalid. We therefore only have to consider cases when $d_{xz}^2 = 0$, i.e. the line segment between the start and the goal position extends directly upward.

Assuming this is the case, when $d_y > 0$, the negative case returns exactly zero. We know this is invalid in practice, since the goal point is higher than the starting point, meaning we have to jump directly upward at some speed to reach it. On the other hand, when $d_y < 0$, the result is irrational, and finally, when $d_y = 0$, the returned value of zero is valid and even correct, but exactly the same as the value found in the positive case. This means we can safely ignore the negative case, resulting in the completed equation for the *minimal jump speed*:

$$v_{min} = \sqrt{g}\sqrt{d_y + \sqrt{d_y^2 + d_{xz}^2}} \tag{13}$$

This minimal jump speed is well-defined for every point pairing. As we have shown that this speed corresponds to exactly one angle, we can define the *minimal jump angle* θ_{min} accordingly. When we substitute $0 = v^4 - g(gd_{xz}^2 + 2d_y v^2)$ into Equation 12, we find that the minimal jump angle is easily computed when the minimal jump speed has been established:

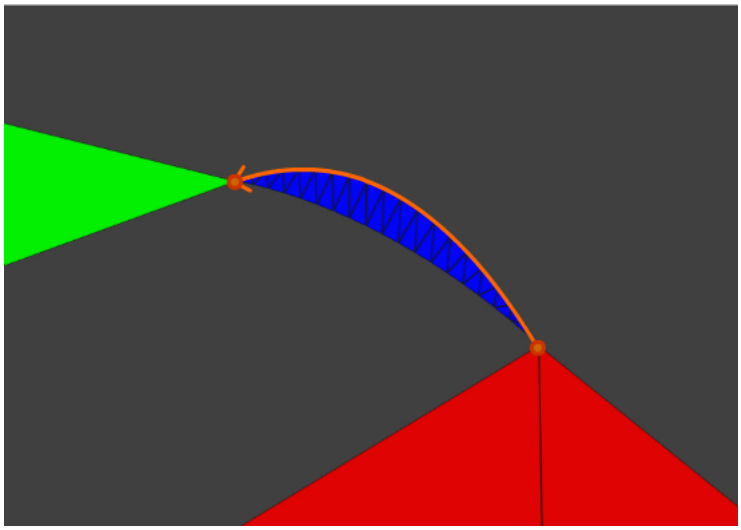

Figure 10: A rendered sketch of a three-dimensional environment, with a minimal velocity jump trajectory drawn from the starting polygon (red) to the goal polygon (green). Note that the other trajectories in the blue slice are also valid, and are in fact shorter than the minimal velocity jump. However, the lower the speed, the larger the range needs to be in order to compensate; the minimal velocity jump between two points also has the longest rational arc.

$$\theta_{min} = arctan(\frac{v_{min}^2}{gd_{xz}}) \tag{14}$$

The minimal jump velocity vector, then, derives naturally from the minimal jump speed and angle. The illustration provided by Figure 10 shows a minimal velocity jump trajectory, contrasted against other trajectories which require greater speeds.

3.4.2 Optimal jump velocity

As previously defined, we define the cost of any path as the sum of the Euclidean length of each of its component lines and curves. In upcoming sections, we motivate this choice further, but this means that for any two points, the trajectory with the lowest possible arc length is the best possible jump between them - in the context of the full path as well. Referring back to the illustration, this means that the lower arc is preferable to the upper arc.

As a result, we can define the *optimal jump velocity* by the speed and angle associated with the trajectory with the smallest arc length. To find these values for any given point pair, we do not need to compute the arc lengths explicitly; we only have to demonstrate that the resulting trajectory length is indeed the smallest possible. The definitions of the optimal jump speed and angle vary depending on the difference in height between the starting and goal point, and so these definitions are provided in separate paragraphs for clarity.

Optimal jump speed when jumping downward When jumping downward (i.e. when the y-value of the starting point is greater than the y-value of the goal point), the shortest trajectory is found when the jumping angle is equal to 0 radians/degrees. Figure 11 makes that evident; any other trajectory, even when the associated speed is greater, must maintain a greater angle and therefore arcs over the trajectory at 0 radians/degrees. As a result, the optimal jump angle when jumping downward θ_{opt_d} is always exactly zero.

By referring to Equation 12, we can determine the formula for the associated optimal speed

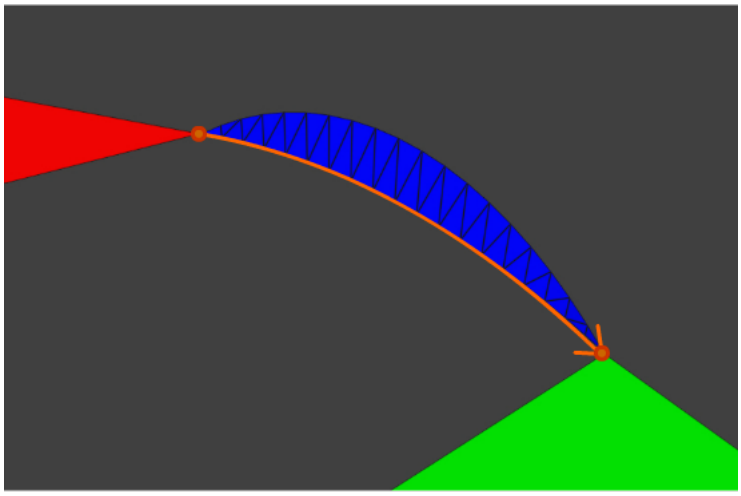

Figure 11: A similar rendered sketch of a three-dimensional environment, with the optimal velocity jump trajectory drawn from the starting polygon (red) to the goal polygon (green). Note that the starting angle here is zero, which is the lowest possible valid angle.

v_{opt_d} by deriving from the following equation:

$$0 = arctan(\frac{v_{opt_d}^2 \pm \sqrt{v_{opt_d}^4 - g(gd_{xz}^2 + 2d_y v_{opt_d}^2)}}{gd_{xz}})$$

Based on trigonometric functions we can state that $arctan(x) = 0 \rightarrow x = 0$, and we know that for the result of a fraction to be zero, the numerator needs to be 0. We can hence reduce the above to the following equation:

$$0 = v_{opt_d}^2 \pm \sqrt{v_{opt_d}^4 - g(gd_{xz}^2 + 2d_y v_{opt_d}^2)}$$

Now, we know that both $v_{opt_d}^2$ and $\sqrt{v_{opt_d}^4 - g(gd_{xz}^2 + 2d_y v_{opt_d}^2)}$ are zero or positive, so all solutions to this equation are found when we subtract the two terms from each other (when both are zero, they could also be added together, but this offers no separate result). Therefore, we can reduce the above to:

$$\begin{aligned}
v_{opt_d}^2 &= \sqrt{v_{opt_d}^4 - g(gd_{xz}^2 + 2d_y v_{opt_d}^2)} \\
v_{opt_d}^2 &= \sqrt{v_{opt_d}^4 - g^2 d_{xz}^2 - 2gd_y v_{opt_d}^2} \\
\sqrt{v_{opt_d}^4} &= \sqrt{v_{opt_d}^4 - g^2 d_{xz}^2 - 2gd_y v_{opt_d}^2} \\
\text{further reducing the equation} \qquad g^2 d_{xz}^2 &= -2gd_y v_{opt_d}^2 \\
gd_{xz}^2 &= -2d_y v_{opt_d}^2 \\
\frac{gd_{xz}^2}{-2d_y} &= v_{opt_d}^2 \\
\sqrt{\frac{gd_{xz}^2}{-2d_y}} &= v_{opt_d}
\end{aligned}$$

This leads us to the following definitions for the optimal jump speed and angle when jumping downward:

$$\begin{aligned}
v_{opt_d} &= \sqrt{\frac{gd_{xz}^2}{-2d_y}} \\
\theta_{opt_d} &= 0
\end{aligned} \tag{15}$$

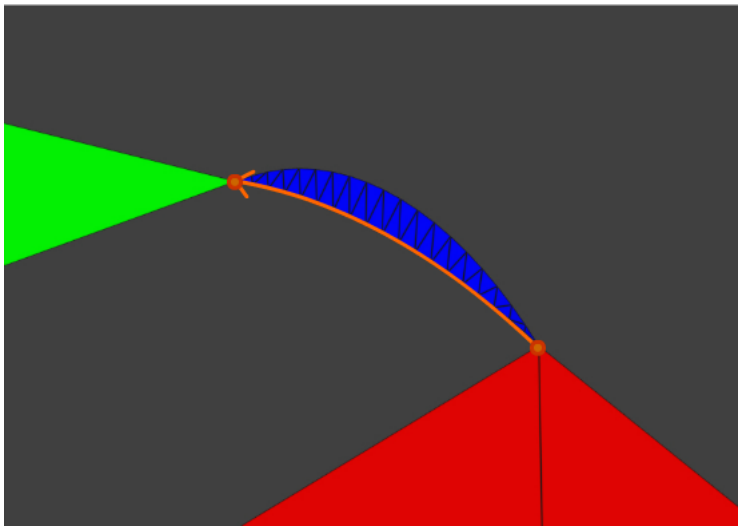

Figure 12: A rendered sketch of a three-dimensional environment, with the optimal velocity jump trajectory drawn from the starting polygon (red) to the goal polygon (green). Note that the landing angle of the trajectory is zero, being the exact converse of the downward trajectory between the two points.

Optimal jump velocity when jumping upward As stated in our assumptions, all valid trajectories must land on their goal point rather than simply cross it, i.e. have a zero or negative upward speed when the trajectory reaches the goal point. Since we consider the navigable space to contain all walkable surfaces as floors, a trajectory might otherwise only cross through the goal point before being propelled further away by momentum. Thus, when jumping upward (i.e. when the y-value of the goal point is greater than the y-value of the starting point), aiming directly at the goal point with an infinitely great speed does not result in a valid trajectory. In a manner mirroring jumping downward, the trajectory is as short as possible when the goal point coincides with its apex, implying that the "angle of motion" at the goal point is always zero.

For this particular case, the launch angle is not a constant, but varies depending on the locations of the start and goal points. Contrary to the preceding derivations, we first find the optimal jump angle θ_{opt_u} and use its value to determine the optimal jump speed v_{opt_u}. We refer back to Equation 11:

$$h = \frac{R * tan(\theta)}{4}$$

Here, h is the highest point of the trajectory, and R is the range of the projectile, i.e. the distance traversed when the parabola reaches the same height as that of its starting point. Since the positions are measured relative to the starting point, the range corresponds to the parabola's second root. In the above equation, we can substitute h with d_y (our desired apex height) and R with $2d_{xz}$ (twice our desired traversed distance). After performing these substitutions, we can rewrite the equation as follows:

$$\begin{aligned} d_y &= \frac{2d_{xz} * tan(\theta_{opt_u})}{4} \\ 4d_y &= 2d_{xz} * tan(\theta_{opt_u}) \\ \frac{4d_y}{2d_{xz}} &= tan(\theta_{opt_u}) \\ arctan(\frac{2d_y}{d_{xz}}) &= \theta_{opt_u} \end{aligned} \tag{16}$$

We now only need to determine the optimal jump speed. We refer back to Equation 8, which describes the height of the trajectory as a function of the distance traversed, and provide

its three-dimensional equivalent:

$$l_y = l_{xz} * tan(\theta_{opt_u}) - \frac{g l_{xz}^2}{2(v_{opt_u} * cos(\theta_{opt_u}))^2} \tag{17}$$

The variable l_{xz} represents the increasing ground plane projected distance from the starting point, while the variable l_y represents the associated height for that point. (The choice of the letter l holds no specific significance, other than being distinct enough to hopefully prevent confusion.) When we substitute l_{xz} with d_{xz} and l_y with d_y, we find the following:

$$d_y = d_{xz} * tan(\theta_{opt_u}) - \frac{g d_{xz}^2}{2(v_{opt_u} * cos(\theta_{opt_u}))^2}$$

By applying the results from Equation 16, we can simplify this function significantly:

$$d_y = d_{xz} * tan(arctan(\frac{2d_y}{d_{xz}})) - \frac{g d_{xz}^2}{2(v_{opt_u} * cos(arctan(\frac{2d_y}{d_{xz}})))^2}$$

$$d_y = d_{xz} * \frac{2d_y}{d_{xz}} - \frac{g d_{xz}^2}{2(v_{opt_u} * cos(arctan(\frac{2d_y}{d_{xz}})))^2}$$

$$d_y = 2d_y - \frac{g d_{xz}^2}{2(v_{opt_u} * cos(arctan(\frac{2d_y}{d_{xz}})))^2}$$

Furthermore, since the relationships between trigonometric functions and their inverses state that $cos(arctan(x)) = \frac{1}{\sqrt{x^2+1}}$:

$$d_y = 2d_y - \frac{g d_{xz}^2}{2v_{opt_u}^2 * \frac{1}{\sqrt{\frac{2d_y}{d_{xz}}^2+1}}^2}$$

$$-d_y = -\frac{g d_{xz}^2}{2v_{opt_u}^2 * \frac{1}{\frac{4d_y^2}{d_{xz}^2}+1}}$$

$$d_y = \frac{g d_{xz}^2 * (\frac{4d_y^2}{d_{xz}^2}+1)}{2v_{opt_u}^2}$$

$$v_{opt_u}^2 = \frac{4g d_y^2 + g d_{xz}^2}{2d_y}$$

$$v_{opt_u}^2 = 2g d_y + \frac{1}{2}\frac{g d_{xz}^2}{d_y}$$

$$v_{opt_u} = \sqrt{2g d_y + \frac{1}{2}\frac{g d_{xz}^2}{d_y}}$$

$$v_{opt_u} = \sqrt{g}\sqrt{2d_y + \frac{1}{2}\frac{d_{xz}^2}{d_y}}$$

This leads us to the following definitions for the optimal jump speed and angle when jumping upward:

$$\theta_{opt_u} = arctan(\frac{2d_y}{d_{xz}})$$

$$v_{opt_u} = \sqrt{g}\sqrt{2d_y + \frac{1}{2}\frac{d_{xz}^2}{d_y}} \tag{18}$$

As found previously, the associated velocity vector is uniquely defined by the optimal jump speed and angle. Figure 12 illustrates that the trajectory traversed is the same as the optimal trajectory in the converse case; we will demonstrate and expand on this property in upcoming sections.

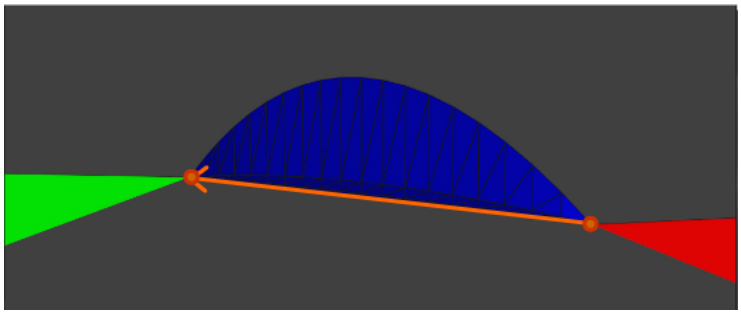

Figure 13: A rendered sketch of a three-dimensional environment, where the starting and goal points are at the same level. The blue slice of possible trajectories approaches the marked straight line as the maximum speed increases, meaning that a valid trajectory is available for any speed above the minimum.

Optimal jump velocity on the level plane In the final remaining case, we consider jumping between two points at the same height. This case is very likely to occur in manually modelled environments, as well as extracted environments where the coordinates have been rounded to an appropriate precision. Here, the straight line segment between the two points is in fact a valid trajectory, if the character is capable of jumping at an infinitely high speed. Depending on the application, this is either valid, or the speed is bounded to a predefined value; regardless, we can always find a valid trajectory associated with the maximum speed v_{max}. As a result, the optimal jump speed on the level plane v_{opt_l} is equal to the v_{max} for that particular character.

This leaves us to find the associated angle θ_{opt_l}. We refer back to Equation 10, where we substitute R with d_{xz}, v with v_{max} and θ with θ_{opt_l}:

$$d_{xz} = \frac{v_{max}^2 * sin(2\theta_{opt_l})}{g}$$

By rewriting this function as follows, we can determine the value for θ_{opt_l}:

$$\begin{aligned} d_{xz} &= \frac{v_{max}^2 * sin(2\theta_{opt_l})}{g} \\ gd_{xz} &= v_{max}^2 * sin(2\theta_{opt_l}) \\ \frac{gd_{xz}}{v_{max}^2} &= sin(2\theta_{opt_l}) \\ \arcsin(\frac{gd_{xz}}{v_{max}^2}) &= 2\theta_{opt_l} \\ \frac{1}{2}\arcsin(\frac{gd_{xz}}{v_{max}^2}) &= \theta_{opt_l} \end{aligned}$$

This leads us to the following definitions for the optimal jump angle and speed on the level plane, which are demonstrated in Figure 13:

$$\begin{aligned} v_{opt_l} &= v_{max} \\ \theta_{opt_l} &= \frac{1}{2}\arcsin(\frac{gd_{xz}}{v_{max}^2}) \end{aligned} \tag{19}$$

Considering all possible cases has provided us with a clear definition of the optimal jump speed and angle between two points, regardless of how they relate to each other spatially. We will therefore be referring to v_{opt} and θ_{opt} respectively, to represent the values appropriate for the starting and goal points.

3.4.3 Rational jump range

Now that we have defined the minimal and optimal jumps for point-to-point jumps, we can introduce the concept of *rational trajectories*. Provided the maximum jump speed of

a character v_{max}, we consider the chosen trajectory between two points rational if it is valid, and has the lowest arc length of the trajectories associated with $min(v_{max}, v_{opt})$. We elaborate on this definition by examining the different cases of how v_{max} relates to v_{min} and v_{opt}:

- If $v_{max} < v_{min}$, we know that there are no valid trajectories, and hence there is no rational trajectory;
- If $v_{max} = v_{min}$, the only valid trajectory between the starting and goal point is the minimal jump trajectory, making this the rational trajectory by extension;
- If $v_{max} > v_{min}$ and $v_{max} \leq v_{opt}$, v_{max} is associated with two possible trajectories, where one has a lower launch angle than the other. The trajectory with a lower launch angle results in a shorter arc length. We therefore consider this trajectory to be rational in this case;
- If $v_{max} > v_{opt}$, we find either a longer trajectory than the one associated with v_{opt} and θ_{opt} or an invalid trajectory; hence, the rational trajectory for this case is the optimal jump trajectory.

Furthermore, any rational jump trajectory will by necessity be located between the minimal and optimal jump, meaning that these are the bounds to a range of trajectories which we hereby refer to as the *rational jump range*. Any jump outside of this range can be considered irrational, since there is always either a jump associated with $min(v_{max}, v_{opt})$ that has a lower arc length, or no valid trajectory available. This gives us a surprisingly compact range of jump trajectories that require consideration in the unobstructed jump space.

3.5 Jump links

In this section, we abstractly define the concept of a jump link, in the context of a three-dimensional multi-layered navigable space. We start by motivating our approach, and further limit the set of trajectories we consider valid in this context. This allows us to strictly define valid edge-to-edge connections, as well as partial and projected connections.

3.5.1 Approach

In this report, we use jumping as a method to navigate a multi-layered navigable space by traversing directly through the three-dimensional space enveloping it. Since this type of environment can contain any configuration of navigable polygons, it could be argued that any valid point-to-point connection in the mesh needs to be considered. In fact, path planning with all potential jumps can find more efficient paths than when jumps are constrained to a subset.

However, accurately representing any possible jump between even just two walkable polygons and selecting a provably efficient arc presents a problem with a complexity exceeding the scope of this report. We could choose to apply sampling methods or imposing limitations on the parameters (such as the shapes of either polygon, the number of permitted jump arcs, the existence of holes and/or the relative placement of the polygons); however, we believe that in many instances, an exact exploration of the edge-to-edge case will be more valuable than a severely limited exploration of the polygon-to-polygon case.

Furthermore, as we are about to demonstrate, certain jumps into polygons can be simulated by choosing virtual edges inside the polygon. Through this approach, we maximize the number of valid connections while limiting the complexity of the associated problem. Using trajectories that include any positions inside polygons can then, in future work, also be approached as a path-smoothing operation on the paths found by this report. Hence, we here only consider jumps between two line segments that are part of polygons. In practice, the approach is in line with the one used in the foremost reference, namely the Master's

thesis by Sara Budde [4].

Finally, it must be noted that any jump link is considered to be bidirectional. In the upcoming sections, we will not explicitly specify the jump direction, especially since the trajectories are not affected by direction of traversal (which would have been the case, had we incorporated air drag). Any concept we define thereby has a counterpart in the other direction.

3.5.2 Assumptions

Succinctly summarized, the assumptions in section 3.2 imply the following requirements for a valid point-to-point jump:

- The launch speed of the jump needs to be equal to or greater than zero;
- The landing speed needs to be equal to or less than zero (e.g. landing "downward");
- The jumping velocity needs to point upward or remain level with the horizon.

Since trajectories directly upward or downward adhere to these requirements, we can conclude that any two points in space can be connected through a valid jump. However, when planning in an environment populated with polygons, we find that not every jump arc is necessarily possible or sensible.

For our following examples, we choose two triangles and a point on one of the edges for each triangle, between which we wish to plan a jump arc. We assume that neither of these polygons is degenerate, which is to say their three points are not collinear and neither triangle is perpendicular to the ground plane. (We can safely exclude these latter cases, since we could not consider such a triangle part of a walkable surface.)

Depending on the placement of these polygons, we find that certain jump arcs intersect the starting polygon or goal polygon. If a jump trajectory intersects with inaccessible obstacles or other layers (”floors”) in the navigable space, we cannot consider it valid, for apparent reasons of realism. Although we do not yet consider polygons other than the aforementioned triangles, we already find that the surrounding context limits the extent to which we can connect any two line segments. (See also Figure 14 for an example.)

We can also construct cases where the valid jump arcs occur away from the direction of the polygon's edge (i.e. at greater than $\frac{1}{2}\pi$ radians). Although these are not necessarily incorrect, we here choose to consider these arcs invalid as well, for two reasons. First of all, almost every jump arc with a launch angle greater than $\frac{1}{2}\pi$ radians (outside of degenerate cases) has a more efficient counterpart with a starting point from another location in the polygon (see also Figure 15 for an example).

Secondly, although these jumps adhere to our previously constructed point-to-point definitions, we find that they appear unrealistic in an environment populated with walkable surfaces. In the context of a larger path, such a jump could only be viewed as a physically impossible heel-face turn, inconsistent with a common viewer's interpretation of momentum (even if we do not apply non-holonomic motion in this report). This limits the valid range of launch angles to $[0, \frac{1}{2}\pi]$.

3.5.3 Valid jump links

In the remainder of this section, we use the term ”edges” to refer to line segments that are specifically part of the boundary of a polygon; ”line segments” is used for the more general case, where they may also be part of a polygon's interior.

Abstractly speaking, we can define a *jump link* as a connection between two valid line

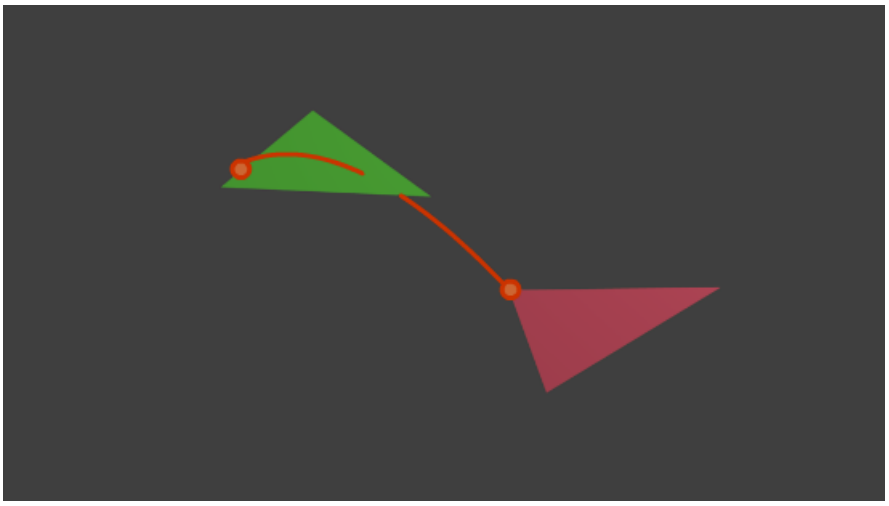

Figure 14: A sketch of a trajectory blocked by two same-sized source polygons. This trajectory is considered invalid. Similarly, trajectories passing through non-source polygons are also considered invalid; the representation of the obstacle space is discussed in upcoming sections.

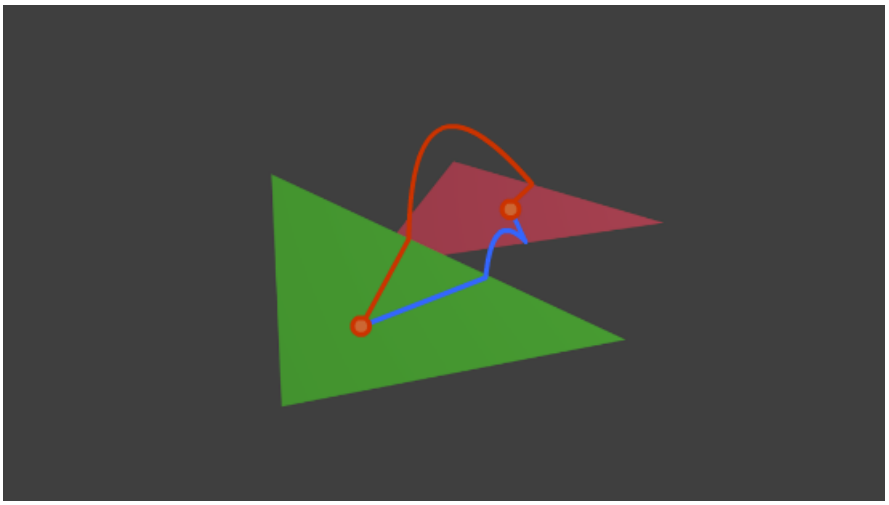

Figure 15: A sketch of two paths, where the red polygon contains the starting point and the green polygon contains the goal point. The orange path contains an arc with a launch angle greater than $\frac{1}{2}\pi$ radians; as we can see, this trajectory has a more efficient counterpart, and appears unrealistic in a larger context. The blue path presents a preferable alternative.

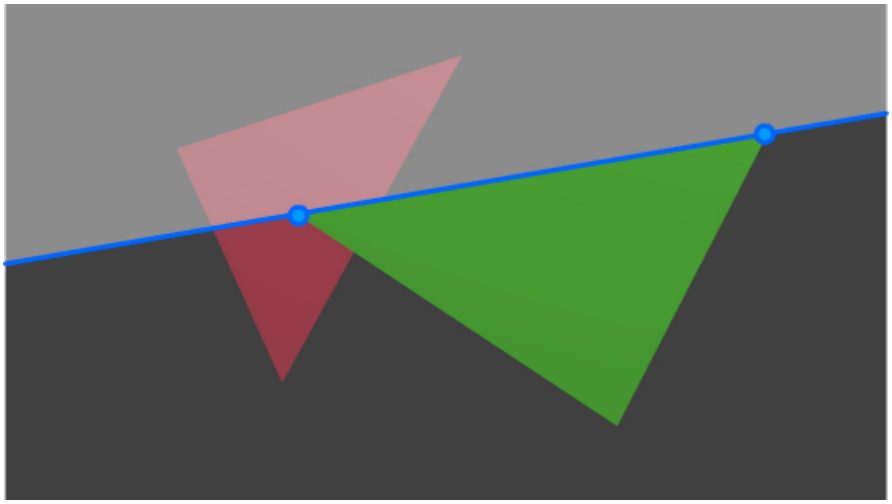

Figure 16: A top-down view sketch of two polygons. The blue line represents the dividing plane between the halfspaces of the edge, whose endpoints are marked with the blue dots. The lightly shaded space represents the outer halfspace for said edge.

segments. Naturally, this definition is meaningless if we do not also provide the definition of valid line segments. Based on the additional constraints, we can now define the combinations of line segments which we consider to be valid. The core principle is that for a jump link to be valid between two line segments, each point on each line segment must have at least one valid jump trajectory to the other segment.

For many combinations of polygonal edges, we find that there may not be a single valid jump arc between them. Using a simple metric to filter the (potentially) valid from the invalid combinations goes a long way towards limiting the computational load of the jump link creation algorithm. Hence, we introduce the concept of *facing edges* to separate the two.

For any edge that belongs to a triangle, we can define two inclusive halfspaces whose bounding plane both entirely contains the edge and is perpendicular to the ground plane. Combined, they contain the entirety of the three-dimensional space. The third point, by necessity, must be in at least one of these halfspaces; since we consider vertical polygons to be degenerate in walkable surfaces, we can state that it resides in strictly one. We call the halfspace containing the triangle's third point the *inner halfspace*, and the other the *outer halfspace*. This is illustrated in Figure 16.

Due to our newly introduced constraints with regards to the jumping angle, the outer halfspace of such an edge contains all trajectories we consider valid. (Since the halfspaces are inclusive, the inner halfspace may also contain valid trajectories, but the outer halfspace must contain them all in any case.) We consider one edge to be *facing* the other if the one edge is entirely contained in the outer halfspace of the other edge, and at most a single endpoint is contained in the halfspace's bounding plane. This latter condition filters degenerate combinations, but allows the case where two edges are connected by an endpoint, i.e. jumping around corner gaps. We consider two edges to be facing if each is facing the other. Although in practice no trajectories may be found, any two facing edges have the potential to be connected by a valid jump arc. Note that the edges need not be directly opposite each other in order to be facing; in fact, we can assume that in practice they often will not be.

3.5.4 Partial jump links

Edges need not be entirely facing or non-facing. One edge may be facing the other but not vice versa; however, accepting these cases would violate our approach of bidirectional jump links. This leaves us with the possibility that the dividing plane of one edge's halfspace

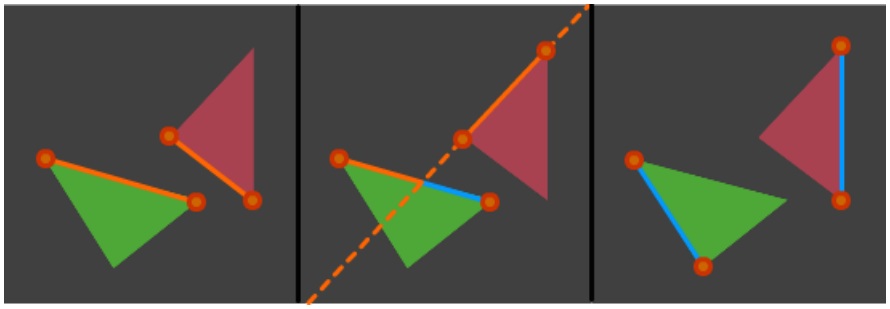

Figure 17: Three top-down sketches of facing, partially facing and non-facing edges. In the first sketch, both edges are facing, which allows us to connect them with a jump link. In the second sketch, only the orange sections can be connected, since the green polygon intersects the dividing plane of the other edge. In the third sketch, neither edge faces the other, so we do not connect these two.

intersects with the other edge, vice versa, or both. According to the preceding definition, we would have to reject these possibilities out of hand. However, valid jump arcs may still exist between parts of the edges, bounded by endpoints and the intersections with the bounding planes. In order to create a complete representation, we wish to consider the combination of valid line segments separately from the remainder of the edge.

This is why we disconnect the concept of a jump link from specific edges. Although pairings of edges are still used to generate links, the result may be any combination of line segments. This allows us to include valid partial cases that would be excluded in a stricter implementation. Such a valid partial case is presented in the second example of Figure 17.

3.5.5 Projected jump links

We include one final possible case that is not directly related to relations between multiple edges, but rather, the relation between an edge and a polygon. As specified in the above constraints, falling down and jumping directly upward constitute valid trajectories. Although we do not perform direct polygon-to-polygon comparisons, this added category allows a wide variety of added connections between one higher edge and one lower polygon. (Note that the spatially converse case is invalid in any configuration.)

These positions can be determined by subdividing the walkable polygons into layers and projecting the outer edges of these layers onto the underlying ones. Wherever the edges project onto an underlying polygon, we can establish a virtual edge directly at the projection site. These virtual edges are used similarly to the partial jump links. However, since the polygon projected onto can have any number of deformities and holes, for this thesis, we only introduce virtual edges when the entirety of the projection lands within the target polygon. Otherwise, partial approaches might be generated which don't combine well with our direct edge to edge mapping. A valid projected jump link is illustrated in Figure 18.

In summary, based on the target environment and restrictions to the trajectories, we have defined three types of jump links we will generate to connect the inner and disjunct areas of walkable surface in any given multi-layer environment. Although the treatise so far has been particularly theoretical, such a fundamental underpinning is required to motivate the concrete concepts we will be providing in future sections. With these definitions out of the way, we can focus on how to create jump link representations that are of use to the path planning algorithm, which is the topic of the next section.

3.6 Jump link concepts

Now that we have identified when we can connect two line segments through a jump link, the question remains exactly what this concept entails. Essentially, a jump link is a representa-

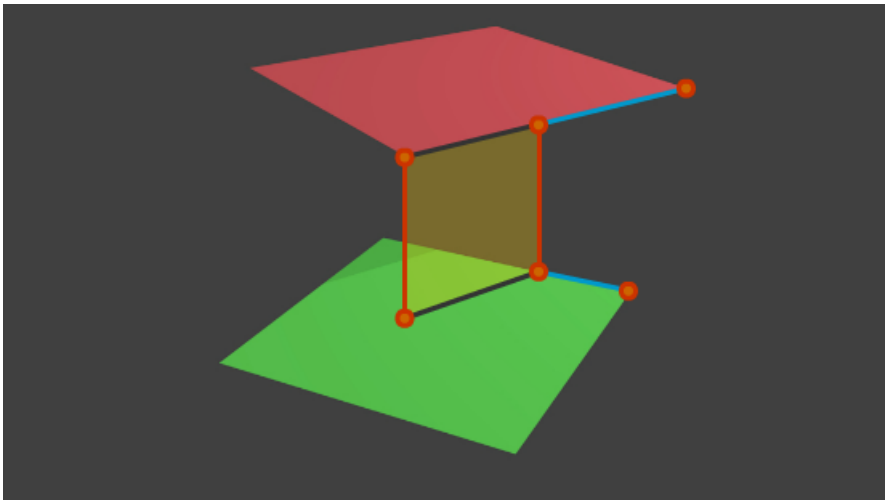

Figure 18: A side view sketch of the two possible jump links between two edges. Projected onto the ground plane, the edges overlap at the middle vertical line. The yellow shaded plane displays the partial projection of the red polygon onto the green polygon, which allows us to add a projected jump link through a virtual edge. The remaining line segments of the edge (colored light blue) can also be connected by a facing jump link.

tion of the geometry between its line segments, and the information we store in these links only needs to be what we will be using in our path planning algorithm. Within this section, we introduce our path planning requirements and provide the methods used to compute them.

As we have seen previously, we can establish any number of trajectories between two line segments, and the remaining geometry may interfere in any number of ways. We wish to make these complex situations more tractable by providing bounds on the trajectories across, as well as a description of the space that the jump link traverses. This section is dedicated to defining the two central concepts that make up our jump link representation and motivating their importance.

First of all, when planning a path across a jump link, we need to know if it is possible for the character to cross it. Although we don't prescribe what applications our method is to be used for, within many contexts (e.g. video game development, navigation based on human locomotion) it is reasonable to assume that the reach of the character is bounded in some way. We would not expect a human character to make a jump of one hundred meters across, but such jump links might very well be added to an environment when our method is applied. Simultaneously, we expect the memory load of our preprocessed stage to be fairly great, since we also need to describe the three-dimensional obstacles for every jump link. This makes it inefficient to store separate sets of jump links for any situation.

We define the *jump link minimal velocity* as the jump vector associated with the lowest possible jump speed, amongst all pairings of starting and goal points on the respective edges. As we have seen previously, such a minimal jump velocity exists for any valid point pairing, and since both line segments are bounded by their endpoints, there must be a lowest minimal jump velocity across any jump link. With this minimal velocity precomputed, we can efficiently prune the search tree during path planning if we define a maximum speed for the character.

Secondly, the "lowest cost" jump across a given jump link is of particular computational interest to us. The A*-algorithm, as described in Chapter 2, uses a search heuristic to focus the exploration of the search tree; it retains optimality if this search heuristic is admissible, i.e. it never overestimates the cost of the full path to the goal. The closer to the lower bound

this heuristic is, the faster the goal is found. For regular locomotion, these heuristics are usually easy to define (e.g. the straight-line distance to the goal), but when incorporating jumping physics, it becomes far more complicated to find efficient heuristics.

Our current path planning approach defines the cost of a path as the sum of the Euclidean length of each of its component lines and curves. When traversing a polygonal environment (efficiently), this cost is restricted to the sum of the length of the separate line segments across each of the polygons. As we introduce jumping, we find that each jump separately can be considered as a segment of that path, curved in three-dimensional space. To maintain a consistent measure of cost, the arc length of the jump should define the cost of that particular segment. As a result, we find a lower bound on the cost across a given jump link if we can compute the arc length of the "smallest trajectory" across a jump link. We define the *jump link minimal arc length* as the shortest possible arc length of the trajectory amongst all pairings of starting and goal points on the respective edges. This heuristic allows us to speed up the path finding process significantly.

These two central concepts form the basis of the jump link representation we will be introducing. In the upcoming sections, we derive the equations necessary to determine these metrics for any two three-dimensional line segments.

4 Jump Link Minimal Velocity

In this section, we will be providing our derivation for the jump link minimal velocity, demonstrating that we can compute it in linear time for any two line segments. Based on the preceding section, we can consider this process completed when we have derived the two points, across the line segments, that have the jump link minimal speed; the associated jump vector follows logically from those two points.

For many combinations of edge segments, especially when we are dealing with a manually constructed environment, the minimal speed required to cross the gap is fairly apparent. Two line segments that are ground-plane parallel and level, for example, are best connected through either a right-angle jump arc or an arc between two endpoints; on the other hand, two line segments that share an endpoint have a lowest possible speed of zero. Here, we start by assuming we know nothing about the line segments we wish to plan between - and interpolate across them as lines rather than bounded segments. Not only does this give us an exact result for the jump link minimal velocity, it also provides us with important conclusions as to the extrema of this function.

The derivation to reach this conclusion is quite elaborate, involves many division and square root operations, and requires a great deal of substitution to keep the result legible. This is why we will first show the full process of finding the equation for the jump link minimal speed, assuming that the contents of square roots are not negative and denominators are not zero; afterwards, we discuss all cases that need to be considered separately under its own heading.

4.1 Interpolated minimal speed

We refer back to Equation 13, which describes the minimal speed between two points as a function of their ground plane projected distance d_{xz} and their height difference d_y:

$$v_{min} = \sqrt{g}\sqrt{d_y + \sqrt{d_y^2 + d_{xz}^2}}$$

When we wish to find the minimal speed between two edges $v_{min}^{s \to g}$, the values for d_{xz} and d_y are dependent on both the point chosen to start from and the point chosen to land on. The valid starting and goal points are each between the first and second endpoint of the respective line segment; as a temporary shorthand, we refer to these points as p_1^s and p_2^s for the starting edge and p_1^g and p_2^g for the goal edge.

Since we know that the edges are line segments in three-dimensional space, we can apply linear interpolation to describe the position of all valid starting and goal points, as well as describe the distance between the selected points. We introduce c_s as the interpolation scalar between p_1^s and p_2^s, where a value of 0 refers to p_1^s and a value of 1 refers to p_2^s. Similarly, we introduce c_g as the interpolation scalar between p_1^g and p_2^g, where a value of 0 refers to p_1^g and a value of 1 refers to p_2^s.

In order to describe the distance vector d between the starting and goal points, which we split in its components d_x, d_y and d_z, as a function of c_g and c_s, we introduce a number of new vectors to our terminology as follows:

- $\bar{d}$, the initial distance vector, is computed by subtracting p_1^s from p_1^g;
- Δ^s, the rate of change for the starting edge, is computed by subtracting p_1^s from p_2^s;
- Δ^g, the rate of change for the goal edge, is computed by subtracting p_1^g from p_2^g.

Note that by the above definitions, we essentially shift the frame of reference to that of p_1^s. These new vectors, which for any two predefined edges are constant values, allow us to define d_x, d_y and d_z as follows:

$$\begin{aligned} d_x &= \bar{d}_x + \Delta_x^g c_g - \Delta_x^s c_s \\ d_y &= \bar{d}_y + \Delta_y^g c_g - \Delta_y^s c_s \\ d_z &= \bar{d}_z + \Delta_z^g c_g - \Delta_z^s c_s \end{aligned}$$

where Δ_x^g refers to the x-component of the vector Δ^g, and all other subscripts are considered similarly. The above already provides us with our desired interpolated definition of d_y, but the definition for d_{xz} has to be composed from the preceding according to the Euclidean definition of distance as follows:

$$\begin{aligned} & d_{xz} = \sqrt{d_x^2 + d_z^2} \\ \text{substituting} \quad & d_{xz} = \sqrt{(\bar{d}_x + \Delta_x^g c_g - \Delta_x^s c_s)^2 + (\bar{d}_z + \Delta_z^g c_g - \Delta_z^s c_s)^2} \\ \text{after expansion} \quad & d_{xz} = \sqrt{\begin{aligned}&(\Delta_x^{s\,2} + \Delta_z^{s\,2})c_s^2 + (\Delta_x^{g\,2} + \Delta_z^{g\,2})c_g^2 + (-2\Delta_x^s - 2\Delta_z^s)c_s c_g \\ &+(-2\bar{d}_x\Delta_x^s - 2\bar{d}_z\Delta_z^s)c_s + (2\bar{d}_x\Delta_x^g + 2\bar{d}_z\Delta_z^g)c_g + (\bar{d}_x^2 + \bar{d}_z^2)\end{aligned}} \end{aligned} \tag{20}$$

The expanded version of the above expression introduces a great number of constants, which we can compute directly from our frame of reference, but which take up a great deal of space in the expression, making it almost illegible. We will encounter this particular problem a few times over the course of the derivation, which is why we will perform quite a few substitutions. The primary purpose of the derivation here is to elucidate the process, rather than provide the equation in its entirety.

We see this in practice when we substitute our newly found descriptions of d_y and d_{xz} into the minimal speed equation. We can rewrite its term $d_y^2 + d_{xz}^2$ as seen below:

$$\begin{aligned} & d_y^2 + d_{xz}^2 \\ & d_y^2 + (\sqrt{d_x^2 + d_z^2})^2 \\ & d_y^2 + d_x^2 + d_z^2 \end{aligned}$$

which when expanded returns a large expression, on par with that of Equation 20. This is why we provide our interpolated minimal speed function with several substitutions performed, which are provided directly afterwards:

$$v_{min}^{s\to g} = \sqrt{g}\sqrt{\bar{d}_y + \Delta_y^g c_g - \Delta_y^s c_s + \sqrt{\alpha c_s^2 + \beta c_g^2 + \gamma c_s c_g + \epsilon c_s + \zeta c_g + \eta}} \tag{21}$$

where

$$\begin{aligned} \alpha &= \Delta_x^{s\,2} + \Delta_y^{s\,2} + \Delta_z^{s\,2} \\ \beta &= \Delta_x^{g\,2} + \Delta_y^{g\,2} + \Delta_z^{g\,2} \\ \gamma &= -2\Delta_x^g\Delta_x^s - 2\Delta_y^g\Delta_y^s - 2\Delta_z^g\Delta_z^s \\ \epsilon &= -2\bar{d}_x\Delta_x^s - 2\bar{d}_y\Delta_y^s - 2\bar{d}_z\Delta_z^s \\ \zeta &= 2\bar{d}_x\Delta_x^g + 2\bar{d}_y\Delta_y^g + 2\bar{d}_z\Delta_z^g \\ \eta &= \bar{d}_x^2 + \bar{d}_y^2 + \bar{d}_z^2 \end{aligned}$$

All of the above substitutions are fairly simple, and each is a constant with regards to the selected edges. We will henceforth be working with Equation 21, but in a number of special cases, we will unsubstitute these expressions.

4.2 Relation between the interpolation variables

Now that we have an expression in Equation 21 that defines the minimal speed as a function of two variables, we can attempt to find its minimum amongst the function's extrema. We know that in the extrema of such a function, both partial derivatives are equal to zero. These partial derivatives are straightforward to find by application of the chain rule and the standard derivative of square root functions. To elucidate the equation's origin, we start by providing the partial derivative of Equation 21 with respect to c_s split up into three distinct parts, separated by applications of the chain rule:

$$\begin{aligned}\frac{\partial v_{min}^{s\to g}}{\partial c_s} = &\frac{\sqrt{g}}{2\sqrt{\bar{d}_y + \Delta_y^g c_g - \Delta_y^s c_s + \sqrt{\alpha c_s^2 + \beta c_g^2 + \gamma c_s c_g + \epsilon c_s + \zeta c_g + \eta}}} * \\ &(-\Delta_y^s + \frac{1}{2\sqrt{\alpha c_s^2 + \beta c_g^2 + \gamma c_s c_g + \epsilon c_s + \zeta c_g + \eta}} * \\ &(2\alpha c_s + \gamma c_g + \epsilon))\end{aligned} \tag{22}$$

We alter it slightly to view the equation as two separate parts:

$$\begin{aligned}\frac{\partial v_{min}^{s\to g}}{\partial c_s} = &\frac{\sqrt{g}}{2\sqrt{\bar{d}_y + \Delta_y^g c_g - \Delta_y^s c_s + \sqrt{\alpha c_s^2 + \beta c_g^2 + \gamma c_s c_g + \epsilon c_s + \zeta c_g + \eta}}} && \text{Part 1} \\ &* \\ &(-\Delta_y^s + \frac{(2\alpha c_s + \gamma c_g + \epsilon)}{2\sqrt{\alpha c_s^2 + \beta c_g^2 + \gamma c_s c_g + \epsilon c_s + \zeta c_g + \eta}}) && \text{Part 2}\end{aligned}$$

At the extrema, the above function needs to be equal to zero; since the two parts are multiplied, this implies that at least one of them has to be zero for this to be true. However, Part 1 in the above equation can only return a value of zero if $g = 0$, and according to our assumptions, this would not constitute a valid simulation. As a result, we know that Part 2 must be equal to zero at the extrema. After some slight rewriting, we can conclude that the following must hold:

$$2\sqrt{\alpha c_s^2 + \beta c_g^2 + \gamma c_s c_g + \epsilon c_s + \zeta c_g + \eta} = \frac{2\alpha c_s + \gamma c_g + \epsilon}{\Delta_y^s} \tag{23}$$

Through a comparable procedure, we find the partial derivative of Equation 21 with respect to c_g:

$$\begin{aligned}\frac{\partial v_{min}^{s\to g}}{\partial c_g} = &\frac{\sqrt{g}}{2\sqrt{\bar{d}_y + \Delta_y^g c_g - \Delta_y^s c_s + \sqrt{\alpha c_s^2 + \beta c_g^2 + \gamma c_s c_g + \epsilon c_s + \zeta c_g + \eta}}} && \text{Part 1} \\ &* \\ &(\Delta_y^g + \frac{(2\beta c_g + \gamma c_s + \zeta)}{2\sqrt{\alpha c_s^2 + \beta c_g^2 + \gamma c_s c_g + \epsilon c_s + \zeta c_g + \eta}}) && \text{Part 2}\end{aligned} \tag{24}$$

Since a similar line of reasoning holds for this partial derivative, the following must hold at the extrema of Equation 21:

$$2\sqrt{\alpha c_s^2 + \beta c_g^2 + \gamma c_s c_g + \epsilon c_s + \zeta c_g + \eta} = \frac{2\beta c_g + \gamma c_s + \zeta}{-\Delta_y^g} \tag{25}$$

Seeing as both Equation 23 and 25 have to be true in the extrema, we find that the following must also hold there:

$$\frac{2\alpha c_s + \gamma c_g + \epsilon}{\Delta_y^s} = \frac{2\beta c_g + \gamma c_s + \zeta}{-\Delta_y^g} \tag{26}$$

This crucial insight allows us to relate c_s to c_g for all extrema. By reducing c_s to an expression containing c_g, we can substitute for c_s, allowing us to find a global minimal velocity for the edges. The above can be rewritten to such an expression:

$$\begin{aligned}
\frac{2\alpha}{\Delta_y^s}c_s + \frac{\gamma}{\Delta_y^s}c_g + \frac{\epsilon}{\Delta_y^s} &= -\frac{2\beta}{\Delta_y^g}c_g - \frac{\gamma}{\Delta_y^g}c_s - \frac{\zeta}{\Delta_y^g} \\
\frac{2\alpha}{\Delta_y^s}c_s + \frac{\gamma}{\Delta_y^g}c_s &= -\frac{2\beta}{\Delta_y^g}c_g - \frac{\gamma}{\Delta_y^s}c_g - \frac{\zeta}{\Delta_y^g} - \frac{\epsilon}{\Delta_y^s} \\
\frac{2\alpha\Delta_y^g + \gamma\Delta_y^s}{\Delta_y^s\Delta_y^g}c_s &= \frac{-2\beta\Delta_y^s - \gamma\Delta_y^g}{\Delta_y^s\Delta_y^g}c_g + \frac{-\zeta\Delta_y^s - \epsilon\Delta_y^g}{\Delta_y^s\Delta_y^g} \\
(2\alpha\Delta_y^g + \gamma\Delta_y^s)c_s &= (-2\beta\Delta_y^s - \gamma\Delta_y^g)c_g + (-\zeta\Delta_y^s - \epsilon\Delta_y^g) \\
c_s &= \frac{-2\beta\Delta_y^s - \gamma\Delta_y^g}{2\alpha\Delta_y^g + \gamma\Delta_y^s}c_g + \frac{-\zeta\Delta_y^s - \epsilon\Delta_y^g}{2\alpha\Delta_y^g + \gamma\Delta_y^s}
\end{aligned}$$

As demonstrated, the relation between c_s and c_g is linear; only the constants involved consist of many different terms. This is why we perform another substitution to simplify our terminology:

$$c_s = \kappa c_g + \lambda \tag{27}$$

where

$$\begin{aligned}
\kappa &= \frac{-2\beta\Delta_y^s - \gamma\Delta_y^g}{2\alpha\Delta_y^g + \gamma\Delta_y^s} \\
\lambda &= \frac{-\zeta\Delta_y^s - \epsilon\Delta_y^g}{2\alpha\Delta_y^g + \gamma\Delta_y^s}
\end{aligned}$$

4.3 Reduction to extrema

Now that we can describe the relation between c_s and c_g, we can replace c_s with $\kappa c_g + \lambda$ in Equation 23 and attempt to simplify it to a solvable form. We present this process below, providing succinct notes as we go:

$$\begin{aligned}
\frac{2\alpha c_s + \gamma c_g + \epsilon}{\Delta_y^s} &= 2\sqrt{\alpha c_s^2 + \beta c_g^2 + \gamma c_s c_g + \epsilon c_s + \zeta c_g + \eta} \qquad \text{(Equation 23)} \\
\frac{2\alpha(\kappa c_g + \lambda) + \gamma c_g + \epsilon}{\Delta_y^s} &= 2\sqrt{\alpha(\kappa c_g + \lambda)^2 + \beta c_g^2 + \gamma c_g(\kappa c_g + \lambda) + \epsilon(\kappa c_g + \lambda) + \zeta c_g + \eta} \\
\frac{2\alpha\kappa c_g + 2\alpha\lambda + \gamma c_g + \epsilon}{2\Delta_y^s} &= \sqrt{\alpha\kappa^2 c_g^2 + 2\alpha\kappa\lambda c_g + \alpha\lambda^2 + \beta c_g^2 + \gamma\kappa c_g^2 + \gamma\lambda c_g + \epsilon\kappa c_g + \epsilon\lambda + \zeta c_g + \eta}
\end{aligned}$$

We perform a substitution in the above, resulting in the following:

$$\frac{\mu c_g + \nu}{2\Delta_y^s} = \sqrt{\alpha\kappa^2 c_g^2 + 2\alpha\kappa\lambda c_g + \alpha\lambda^2 + \beta c_g^2 + \gamma\kappa c_g^2 + \gamma\lambda c_g + \epsilon\kappa c_g + \epsilon\lambda + \zeta c_g + \eta} \tag{28}$$

where

$$\begin{aligned}
\mu &= 2\alpha\kappa + \gamma \\
\nu &= 2\alpha\lambda + \epsilon
\end{aligned}$$

Continuing on from Equation 28, we find:

$$\frac{(\mu c_g + \nu)^2}{4\Delta_y^{s\,2}} = \alpha\kappa^2 c_g^2 + 2\alpha\kappa\lambda c_g + \alpha\lambda^2 + \beta c_g^2 + \gamma\kappa c_g^2 + \gamma\lambda c_g + \epsilon\kappa c_g + \epsilon\lambda + \zeta c_g + \eta$$

$$\frac{\mu^2 c_g^2 + 2\mu\nu c_g + \nu^2}{4\Delta_y^{s\,2}} = (\alpha\kappa^2 + \beta + \gamma\kappa)c_g^2 + (2\alpha\kappa\lambda + \gamma\lambda + \epsilon\kappa + \zeta)c_g + (\alpha\lambda^2 + \epsilon\lambda + \eta)$$

$$\frac{\mu^2}{4\Delta_y^{s\,2}} c_g^2 + \frac{2\mu\nu}{4\Delta_y^{s\,2}} c_g + \frac{\nu^2}{4\Delta_y^{s\,2}} = (\alpha\kappa^2 + \beta + \gamma\kappa)c_g^2 + (2\alpha\kappa\lambda + \gamma\lambda + \epsilon\kappa + \zeta)c_g + (\alpha\lambda^2 + \epsilon\lambda + \eta)$$

$$(\frac{\mu^2}{4\Delta_y^{s\,2}} - (\alpha\kappa^2 + \beta + \gamma\kappa))c_g^2 + (\frac{2\mu\nu}{4\Delta_y^{s\,2}} - (2\alpha\kappa\lambda + \gamma\lambda + \epsilon\kappa + \zeta))c_g + (\frac{\nu^2}{4\Delta_y^{s\,2}} - (\alpha\lambda^2 + \epsilon\lambda + \eta)) = 0$$

which, despite seeming daunting, is simply a quadratic equation, which can be solved through the quadratic formula:

$$c_g = \frac{-\bar{b} \pm \sqrt{\bar{b}^2 - 4\bar{a}\bar{c}}}{2\bar{a}}$$

where

$$\bar{a} = (\frac{\mu^2}{4\Delta_y^{s\,2}} - (\alpha\kappa^2 + \beta + \gamma\kappa))$$

$$\bar{b} = (\frac{2\mu\nu}{4\Delta_y^{s\,2}} - (2\alpha\kappa\lambda + \gamma\lambda + \epsilon\kappa + \zeta))$$

$$\bar{c} = (\frac{\nu^2}{4\Delta_y^{s\,2}} - (\alpha\lambda^2 + \epsilon\lambda + \eta))$$

This finally provides us with the values for c_s and c_g at all possible extrema of Equation 21, which are found as follows:

$$\begin{aligned} c_g &= \frac{-\bar{b} \pm \sqrt{\bar{b}^2 - 4\bar{a}\bar{c}}}{2\bar{a}} \\ c_s &= \kappa c_g + \lambda \end{aligned} \tag{29}$$

And when we can use this to determine the possible values for c_s and c_g, we can substitute their d_{xz} and d_y into Equation 13 to determine the jump link minimal speed, as well as the jump link minimal angle and by extension, the jump link minimal velocity.

It must be noted that the values for c_s and c_g are not restricted to the bounds of the line segments they are derived from; they may well extend under 0 or beyond 1. However, this result does allow us insight into the structure of the function, which we will refer back to in the upcoming discussion of the special cases and the structural observations.

4.4 Special cases

Having seen the entire derivation process, we now examine in chronological order the as of yet undiscussed special cases where the above derivation returns no result at all. Note that we have already established in Section 3.4.1 that the point-to-point minimal speed is well-defined for any two points. For legibility's sake, we here repeat its definition:

$$v_{min} = \sqrt{g}\sqrt{d_y + \sqrt{d_y^2 + d_{xz}^2}}$$

We therefore find the first of these cases in Equation 22; the partial derivative of the interpolated minimal speed with respect to c_s. (Note that the case presented here applies to the partial derivative with respect to c_g as well.) The result of this equation becomes invalid in two different cases. The first case occurs when the minimal speed between c_g and c_s is actually zero. Now, based on its definition, we know this situation occurs when the following two conditions both apply:

- d_{xz}, i.e. the ground plane projected distance between the two points is zero;
- d_y, i.e. the height difference between the start and the goal point is negative.

In other words, this case occurs when there is at least one point on the starting edge that lies directly above the goal edge. As we have previously discussed, falling directly downward with zero speed actually constitutes a valid jump. As a result, we can simply define the minimal jump speed in such a case as this as zero.

The second case occurs when the denominator found in Part 2 of Equation 22 is zero. This would occur when $\sqrt{d_x^2 + d_y^2 + d_z^2}$ is actually zero; as we can observe directly, this implies that the distance between the two points is zero. We know that this case can occur in practice, if we select two line segments that overlap in exactly one point. Especially manually constructed environments are likely to have such cases occur, where characters can jump across corners or over gaps in walkable polygons. However, it must be apparent that the minimal jump speed in this case is zero as well.

We encounter our next potentially problematic derivation when deriving the relation between c_s and c_g. Specifically, Equation 26 cannot be used directly when either Δ_y^s or Δ_y^g is zero - and especially when both are zero. Since this means that either edge is level, we can be assured that these cases will occur in the intended environments, either by design or by sampling coordinate rounding.

For any of these cases, we find that we can establish a valid definition of κ and λ (and thereby, a valid relation between c_s and c_g) regardless. The previously demonstrated derivation results in the most complicated definition of κ and λ. We will here provide a separate explicit derivation of κ and λ for the case when Δ_y^s is zero, but Δ_y^g is not. This means we start from the following relation between the partial derivatives:

$$
\begin{aligned}
2\alpha c_s + \gamma c_g + \epsilon &= \frac{2\beta c_g + \gamma c_s + \zeta}{-\Delta_y^g} \\
2\alpha c_s + \gamma c_g + \epsilon &= -\frac{2\beta}{\Delta_y^g} c_g - \frac{\gamma}{\Delta_y^g} c_s - \frac{\zeta}{\Delta_y^g} \\
(2\alpha - \frac{\gamma}{\Delta_y^g}) c_s &= (-\frac{2\beta}{\Delta_y^g} - \gamma) c_g + (-\frac{\zeta}{\Delta_y^g} - \epsilon) \\
(2\alpha\Delta_y^g - \gamma) c_s &= (-\gamma\Delta_y^g - 2\beta) c_g + (-\epsilon\Delta_y^g - \zeta) \\
c_s &= \frac{-\gamma\Delta_y^g - 2\beta}{2\alpha\Delta_y^g - \gamma} c_g + \frac{-\epsilon\Delta_y^g - \zeta}{2\alpha\Delta_y^g - \gamma}
\end{aligned}
$$

And thereby we find the alternative definitions for κ and λ:

$$
\begin{aligned}
\kappa &= \frac{-\gamma\Delta_y^g - 2\beta}{2\alpha\Delta_y^g - \gamma} \\
\lambda &= \frac{-\epsilon\Delta_y^g - \zeta}{2\alpha\Delta_y^g - \gamma}
\end{aligned}
$$

Since the other two cases do not result in more complicated derivations, we do not explicitly provide them here. We also observe that when either edge is level, we can still perform the reduction to extrema without the associated denominator problems.

Finally, we must conclude that this function, while in itself valuable in proving that the jump link minimal velocity does indeed exist, may not always result in us directly finding a minimum, for example when the two line segments are parallel. We address why we do not explicitly need to address these cases, since they do not directly interfere with our usage of this derivation, in Section 6.

5 Jump Link Minimal Arc Length

In addition to finding the jump link minimal speed, which during planning aids to prune unreachable jump connections from the search tree, we also aim to find the jump across any given link where the length of the jump arc (i.e. the cost of that connection) is minimal. When performing a search, this allows us to prune reachable, but suboptimal jump connections. We start by demonstrating how we can simplify this problem by applying symmetry, before deriving the minimal arc length between set starting and goal points. By examining this function and proving certain crucial properties, we motivate finding the jump link minimal arc length through practical methods.

5.1 Arc length symmetry

For predefined starting and goal points, we can disregard any speed and angle pair that is not optimal, since this pair by definition has a longer trajectory. Since we have three different definitions for the optimal speed, i.e. when jumping downward, jumping upward and on the level plane, one would assume we require three separate functions to describe any trajectory's arc length. However, we will show here that we only require one arc length integral for all three cases.

The central observation is that, given the position of the starting and goal point, the optimal trajectory when jumping upward from the start to the goal $t^{s\to g}_{opt_u}$ is the same as the optimal trajectory when jumping downward from the goal to the starting point $t^{g\to s}_{opt_d}$. We will demonstrate this by proving that the angle of $t^{g\to s}_{opt_d}$ at d_{xz} is the negative of the starting angle for $t^{s\to g}_{opt_u}$. Since trajectories can be uniquely defined by their relative position to the goal and the starting angle (note how there is only one valid speed in Equation 12 when θ, d_{xz} and d_y are given), we hereby show that $t^{s\to g}_{opt_u}$ and $t^{g\to s}_{opt_d}$ traverse the same arc, only in the opposite direction.

To find the angle of $t^{g\to s}_{opt_d}$ at d_{xz}, we need to find the optimal trajectory's slope at d_{xz}. For this, we need the trajectory function's derivative, a result that will come in handy in upcoming sections. Equation 17 for this particular case is as follows:

$$l_y = l_{xz} * tan(\theta^{g\to s}_{opt_d}) - \frac{g l^2_{xz}}{2(v^{g\to s}_{opt_d} * cos(\theta^{g\to s}_{opt_d}))^2}$$

When jumping downwards, θ_{opt_d} is always equal to zero, and since $tan(0) = 0$ and $cos(0) = 1$, this reduces to:

$$l_y = -\frac{g l^2_{xz}}{2v^{g\to s\,2}_{opt_d}}$$

The optimal speed can be described in terms of d_{xz} and d_y, as demonstrated in Equation 15, leading to the following derivation:

$$l_y = -\frac{g l^2_{xz}}{2\sqrt{\frac{g d^2_{xz}}{-2d_y}}^2}$$

$$l_y = -\frac{g l^2_{xz}}{2\frac{g d^2_{xz}}{-2d_y}}$$

$$l_y = -\frac{g l^2_{xz}}{\frac{g d^2_{xz}}{-d_y}}$$

$$l_y = \frac{d_y g}{g d^2_{xz}} l^2_{xz}$$

The final simplification step results in a particularly succinct trajectory function:

$$l_y = \frac{d_y}{d_{xz}^2} l_{xz}^2 \tag{30}$$

Its derivative is equally succinct, and easily found:

$$l_y' = \frac{2d_y}{d_{xz}^2} l_{xz} \tag{31}$$

We can hence determine that the slope of the trajectory at d_{xz} is equal to $\frac{2d_y}{d_{xz}}$. This slope signifies that every unit of ground plane projected movement corresponds to $\frac{2d_y}{d_{xz}}$ units of vertical movement; ergo, it is the ratio between vertical and ground plane motion. To find the angle associated with this slope, we consider that in this scenario the vertical motion is the opposite side with regards to the line through d_{xz}, where the ground plane motion is the adjacent side with regards through the line through d_{xz}; the angle is therefore found by computing

$$arctan(\frac{2d_y}{d_{xz}})$$

which is equal to $\theta_{opt_u}^{s \to g}$, according to Equation 18.

Finally, we observe that the arc length of the optimal speed jump on the level plane theoretically depends on the maximum speed. However, since we are looking for an admissible (i.e. underestimating) heuristic, we choose an infinitely high maximum speed. The associated "arc" is a straight line, and so on the level plane, the minimal arc length is simply d_y.

5.2 Arc length integral

Based on the previously demonstrated symmetry, we now only have to determine the arc length for downward jumps to cover all cases. Referring to elementary calculus, we find that the arc length l_{arc} of a given curve $f(x)$ is found by solving the following integral:

$$l_{arc} = \int_a^b \sqrt{1 + (f'(x))^2} dx$$

where a and b are the upper and lower bounds respectively of the measured curve section. Substituting in the downward trajectory derivative established in Equation 31, the integral becomes:

$$l_{arc} = \int_a^b \sqrt{1 + \left(\frac{2d_y}{d_{xz}^2} l_{xz}\right)^2} \, dl_{xz}$$

The curve section we're interested in, in terms of l_{xz} (i.e. distance from the starting point), runs from zero to d_{xz}, which results in the following value for the arc length l_{arc}:

$$l_{arc} = \int_0^{d_{xz}} \sqrt{1 + \frac{4d_y^2}{d_{xz}^4} l_{xz}^2} \, dl_{xz} \tag{32}$$

The required integral is difficult to determine, but when given a possible solution, it is simple to verify through derivation. So for simplicity's sake, we here provide the resulting integral first, and then demonstrate how it derives to the above equation. The arc length can be defined as follows:

$$l_{arc} = \sqrt{\frac{4d_y^2}{d_{xz}^4} l_{xz}^2 + 1} * \frac{1}{2} l_{xz} + \frac{1}{2\sqrt{\frac{4d_y^2}{d_{xz}^4}}} sinh^{-1}\left(\sqrt{\frac{4d_y^2}{d_{xz}^4}} l_{xz}\right) \Big|_0^{d_{xz}} \tag{33}$$

where $sinh^{-1}$ is the inverse hyperbolic sine function. Recall that since for two set points, d_y and d_{xz} are constants, when determining the derivative, we can substitute:

$$\alpha = \frac{4d_y^2}{d_{xz}^4}$$

for considerable simplification. The derivation process is provided below:

$$(\sqrt{\alpha l_{xz}^2+1}*\frac{1}{2}l_{xz}+\frac{1}{2\sqrt{\alpha}}sinh^{-1}(\sqrt{\alpha}l_{xz}))'$$

$$\frac{1}{2\sqrt{\alpha l_{xz}^2+1}}*2\alpha l_{xz}*\frac{1}{2}l_{xz}+\frac{1}{2}\sqrt{\alpha l_{xz}^2+1}+\frac{1}{2\sqrt{\alpha}}*\frac{\sqrt{\alpha}}{\sqrt{\alpha l_{xz}^2+1}}$$

$$\frac{\alpha l_{xz}^2}{2\sqrt{\alpha l_{xz}^2+1}}+\frac{1}{2}\sqrt{\alpha l_{xz}^2+1}+\frac{1}{2\sqrt{\alpha l_{xz}^2+1}}$$

$$\frac{\alpha l_{xz}^2+1}{2\sqrt{\alpha l_{xz}^2+1}}+\frac{1}{2}\sqrt{\alpha l_{xz}^2+1}$$

$$\frac{1}{2}\frac{\alpha l_{xz}^2+1}{\sqrt{\alpha l_{xz}^2+1}}+\frac{1}{2}\sqrt{\alpha l_{xz}^2+1}$$

$$\frac{1}{2}\sqrt{\alpha l_{xz}^2+1}+\frac{1}{2}\sqrt{\alpha l_{xz}^2+1}$$

$$\sqrt{\alpha l_{xz}^2+1}$$

unsubstituting: $$\sqrt{1+\frac{4d_y^2}{d_{xz}^4}l_{xz}^2}$$

Note that when we choose a value of zero for the arc length function, we always find zero. The arc length function can as a result be reduced to a single equation:

$$l_{arc}=\sqrt{\alpha d_{xz}^2+1}*\frac{1}{2}d_{xz}+\frac{1}{2\sqrt{\alpha}}sinh^{-1}(\sqrt{\alpha}d_{xz}) \tag{34}$$

And when we unsubstitute α in this equation, as well as harken back to its original definition to consider the following (note that we do not need to take the absolute value, since both d_y and d_{xz} will always be positive):

$$\sqrt{\alpha}=\sqrt{\frac{4d_y^2}{d_{xz}^4}}=\sqrt{(\frac{2d_y}{d_{xz}^2})^2}=\frac{2d_y}{d_{xz}^2} \tag{35}$$

We can further reduce the arc length equation as follows:

$$l_{arc}=\sqrt{\frac{4d_y^2}{d_{xz}^4}d_{xz}^2+1}*\frac{1}{2}d_{xz}+\frac{1}{2\frac{2d_y}{d_{xz}^2}}sinh^{-1}(\frac{2d_y}{d_{xz}^2}d_{xz}) \tag{36}$$

$$l_{arc}=\sqrt{\frac{4d_y^2}{d_{xz}^2}+1}*\sqrt{\frac{1}{4}d_{xz}^2}+\frac{d_{xz}^2}{4d_y}sinh^{-1}(\frac{2d_y}{d_{xz}}) \tag{37}$$

$$l_{arc}=\sqrt{d_y^2+\frac{1}{4}d_{xz}^2}+\frac{d_{xz}^2}{4d_y}sinh^{-1}(\frac{2d_y}{d_{xz}}) \tag{38}$$

Sadly, we must conclude that applying interpolation to this function does not result in a neat, derivable function like the one we have seen for the minimal velocity. Hence, a formal definition of the jump link minimal arc length remains out of our reach. However, the steps we have taken towards a practical implementation show how we do not necessarily need this formal definition to proceed. Therefore, we conclude this train of thought in Section 6.

6 Practical Implementation

In this section, we discuss the limitations of the results we have achieved so far in a practical setting, and address how we can fill the remaining gaps. We start by examining the minimal jump speed function.

As shown in Section 3.4.1, the minimal jump speed function is well-defined for any two points. This means that for any input points, the minimal jump speed will return a valid value, and as a result, the interpolated minimal jump speed function must be continuous in nature. Knowing this, we refer to the result of Equation 29. We find that the interpolated minimal jump speed equation has at most two extrema.

Under stricter circumstances, we might require a second derivative test to categorize each of these points as either a minimum, maximum or saddle point. This would allow us to prove exactly which category these extrema belong to, although notably, they cannot both be minima or both be maxima. However, practical concerns come into play here. During the implementation of the aforementioned method in a double-type precision environment, we found that the steps involved in this first-derivative procedure (in particular, the great numbers of square root operations) led to a systematic inaccuracy in our results. We were able to retrace these inaccuracies by grid sampling point pairs around a number of single-minimum edge pairings.

The inaccuracies we encountered occurred at the earliest around the fifth decimal. For environments computed with up to the centimeter or millimeter precision, such inaccuracies can certainly affect the results, even if the difference is fairly minute. And based on the complexity of the preceding derivation, we can be assured that the second derivatives will suffer from even greater inaccuracies. We must therefore consider that this test may not even be sufficiently reliable in practice.

The prevalence of numerical inaccuracy in most available frameworks requires us to accept, to a degree, an imperfection in our results. The preceding has been entirely exact; one would hope that the results reflect this. Luckily, we have shown that there are at most two extrema in the jump link minimal velocity function, which cannot logically both be minima or both be maxima. As a result, even when the two extrema are sufficiently close together, iterative grid sampling of this continuous function will lead to increasingly accurate results. In practice, we apply sampling pairs to find the minimal arc to within acceptable thresholds.

This sampling approach also applies to the jump link minimal arc length. Through practical testing, we have found that although attempts at explicit derivation have not had the desired effect, iterative grid sampling shows that for any one point, there is a single minimum amongst all possible target locations, and (logically) vice versa. We are thereby able to perform two-dimensional sampling to find the desired value up to the precision deemed meaningful. Given that any explicit derivation would result in similar numerical inaccuracies, we can consider the jump link minimal arc length found through sampling methods.

Finally, a gap still very much remaining is the case we have identified on jump links projected down. However, we have omitted this case from our preceding derivations, since the minimal velocity and arc length are easy to determine. The minimal velocity is either zero, or the velocity required to jump up between the two vertically closest points; the arc length is the same in both cases, namely the vertical distance between the aforementioned points. We do however note that the definition of a jump link is not complete without this case being covered in terms of connectivity. The challenge then becomes to find these instances - by projecting all edges onto surfaces where applicable.

7 Conclusion

Within this report, we have approached jumping between layers in a three-dimensional walkable environment from a theoretical standpoint. Through analysis of the related work, we have identified that an exact approach, free from predefined jump trajectories and sampling, was lacking from the field. Starting from axiomas from Newtonian physics and select assumptions for realism, we defined the central concepts of this report: the minimal jump and optimal jump between any two points, which together bound the rational jump range for any point robot.

Further based on our axiomas and careful analysis, we examined in great detail the necessity for a jump link structure, as well as the cases when these jump links are actually applicable in practice. This has allowed us to clearly define valid jump links, that can be used in any simulation requiring a realistic approach. We then find that any jump link that is to be used in path planning would require some manner of heuristics for planning algorithms.

We extend our existing base of definitions to two three-dimensional line segments, which we refer to as the jump link minimal velocity and the jump link minimal arc length, respectively. These values provide heuristics for jump minimality and optimality, which can be directly used by A* algorithms in planning a path across any complex environment. We explicitly derive the equations used to compute them, as far as has proven necessary to determine how we can implement them.

When applying these definitions to practical situations, we found that the accuracy of these methods could be improved by iteratively sampling results rather than attempting a direct derivation. We also require separate handling of the vertical cases which naturally followed from our axiomatic system.

The results of this report lay the foundation for more efficient, accurate and flexible path planning algorithms that take the full dimensionality of their environment into account. As addressed in the discussion, further results of practical implementation are omitted; we leave it to future researchers to expand on the metrics provided here.

8 Discussion and Future Work

From the theoretical framework of this report, a great number of possible avenues of research remain. We here describe these potential approaches, in the hope that future colleagues expand this work to progress the field even further.

Obstacle space representations - One avenue of research is including representations of the obstacle space between two edges as part of their jump link. This would eliminate the need for examining jumps on a case-by-case basis during path planning, allowing algorithms to plan paths in any polygonal environment in a single stretch. When the scope of this report was originally determined, this item had been considered up to the point of example calculations under simple circumstances (e.g. a vertical line obstacle blocking part of the rational jump range between two points). Despite its merit, it had to be concluded that the potential complexity of such a representation would necessitate a report of similar size to this one all on its own.

Practical A* algorithm implementation results - Arguably the greatest omission in this report is the number of practical results from applying these metrics in the A* algorithm, or similar path planning methods. Although the provided derivations stand on their own, a number of paths planned by connecting a suitable navigation mesh to the jump links would give instantaneous insight into the applicability of this method. Aside from structurally simple tessellations, this would require additional research into the annotated navigation structure as a whole.

Simulations with improved realism - Within this report, we have chosen a simplified set of axiomatic equations governing motion in order to strike a balance between realism and computational cost - two aspects which are commonly at odds. The resulting separation between the two modes of motion and the jump arc symmetry have been invaluable to achieving our results, but they by no means result in realistic effects. Within a nonholonomic system of motion, i.e. where the run-up to the point of jumping affects the available speeds and/or angles, the paths found by any planning algorithm (under the appropriate settings) will approximate reality far more closely, both mathematically and to the human eye. To a lesser extent, reintroducing air drag would have a similar effect, although one imagines the cost of reconsidering the results presented here would outweigh the value of such a small-scale improvement.

Additional modes of motion - We have here considered jumping as a mode of motion secondary to walking. As seen in the Related Work section, there is a significant amount of additional research pertaining to this approach. However, other modes of motion may also warrant further investigation. Notably, the SMART system employed in Brink [13] employs parkour-style animation controllers to scale the world geometry; its informal approach leaves room for more exact path planning with climbing and scaling options. Any approach that expands the range of motion of (human or robot) characters can contribute to the quality of their simulations.

References

[1] Nancy M Amato and Guang Song. Using motion planning to study protein folding pathways. *Journal of Computational Biology*, 9(2):149–168, 2002.

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

[3] Harry Blum. A transformation for extracting new descriptors of shape. *Proc. Symp. Models for Perception of Speech and Visual Form*, pages 362–380, 1967.

[4] Sara Budde. Automatic generation of jump links in arbitrary 3d environments. MSc Thesis, 2013.

[5] Mylène Campana and Jean-Paul Laumond. Ballistic motion planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2016)*, Daejeon, South Korea, October 2016.

[6] L Paul Chew. Constrained delaunay triangulations. *Algorithmica*, 4(1-4):97–108, 1989.

[7] CryTek. Cryengine documentation on off-mesh navigation, from the multi-layer navigation section. `http://docs.cryengine.com/display/SDKDOC2/Off-mesh+Navigation`. Accessed: 08-07-2015.

[8] Mark De Berg, Marc Van Kreveld, Mark Overmars, and Otfried Cheong Schwarzkopf. *Computational Geometry*. Springer, 2000.

[9] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[10] Michael L Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, 34(3):596–615, 1987.

[11] Roland Geraerts. Planning short paths with clearance using explicit corridors. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 1997–2004. IEEE, 2010.

[12] Roland Geraerts and Mark H Overmars. The corridor map method: a general framework for real-time high-quality path planning. *Computer Animation and Virtual Worlds*, 18(2):107–119, 2007.

[13] Arne Olav Hallingstad. Vault, slide, mantle: Building brink's smart system. Presented at the Game Developers Conference 2012, 2012. Accessed: 10-07-2012.

[14] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968.

[15] Marcelo Kallmann. Path planning in triangulations. In *Proceedings of the IJCAI workshop on reasoning, representation, and learning in computer games*, pages 49–54, 2005.

[16] Mubbasir Kapadia, Alejandro Beacco, Francisco Garcia, Vivek Reddy, Nuria Pelechano, and Norman I Badler. Multi-domain real-time planning in dynamic environments. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 115–124. ACM, 2013.

[17] Lydia E Kavraki, Petr Švestka, Jean-Claude Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566–580, 1996.

[18] Manfred Lau and James J Kuffner. Precomputed search trees: planning for interactive goal-driven animation. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 299–308. Eurographics Association, 2006.

[19] Steven M LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[20] Sergey Levine, Yongjoon Lee, Vladlen Koltun, and Zoran Popović. Space-time planning with parameterized locomotion controllers. *ACM Transactions on Graphics (TOG)*, 30(3):23, 2011.

[21] Thomas Lopez, Fabrice Lamarche, and Tsai-Yen Li. Space-time planning in changing environments: using dynamic objects for accessibility. *Computer Animation and Virtual Worlds*, 23(2):87–99, 2012.

[22] Mikko Mononen and contributors. Recast navigation open source project. `https://github.com/memononen/recastnavigation`. Accessed: 10-07-2015.

[23] John H Reif. Complexity of the movers problem and generalizations extended abstract. In *Proceedings of the 20th Annual IEEE Conference on Foundations of Computer Science*, pages 421–427, 1979.

[24] John H Reif. *Complexity of the Generalized Mover's Problem*. Center for Research in Computing Techn., Aiken Computation Laboratory, Univ., 1985.

[25] W.G. van Toll, R. Triesscheijn, M. Kallmann, R. Oliva, N. Pelechano, J. Pettré, and R. Geraerts. A comparative study of navigation meshes. In *Proceedings of the 9th International ACM SIGGRAPH Conference on Motion in Games*, pages 91–100, 2016.

[26] Unity. Unity documentation on navmesh baking, from the navigation overview section. `https://docs.unity3d.com/353/Documentation/Manual/Navmeshbaking.html`. Accessed: 09-07-2015.

[27] Unity. Unity documentation on off-mesh links, from the navigation overview section. `http://docs.unity3d.com/Manual/class-OffMeshLink.html`. Accessed: 08-07-2015.

[28] Ron Wein, Jur P Van Den Berg, and Dan Halperin. The visibility–voronoi complex and its applications. In *Proceedings of the Twenty-First Annual Symposium on Computational Geometry*, pages 63–72. ACM, 2005.