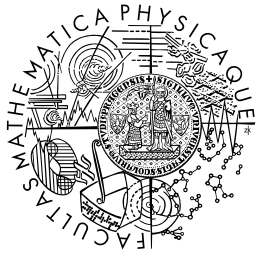


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Vojtěch Pröschl

Crowd simulation

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the bachelor thesis: prof. RNDr. Roman Barták, Ph.D.
Study programme: Computer Science

Prague 2025

I declare that I carried out this bachelor thesis on my own, and only with the cited sources, literature and other professional sources. I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

I would like to express my sincere gratitude to prof. RNDr. Roman Barták, Ph.D. for the opportunity to explore this topic and for his guidance throughout this work.

This work is especially dedicated to my parents, whose unwavering support, and belief in me have been the foundation of my academic journey. Thank you for always being there.

Title: Crowd simulation

Author: Vojtěch Pröschl

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: prof. RNDr. Roman Barták, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: Simulating crowds in complex terrain demands path planning that accounts for the terrain geometric properties. This thesis presents a mathematically rigorous traversal model grounded in differential geometry. The model evaluates candidate routes on a smooth height-map surface using generic cost functions derived from geometric properties of paths. The framework implicitly defines optimization problem that identifies realistic routes based on distance, surface normals and heading directions. The continuous formulation is then applied to crowd simulation by discretizing the model and employing path-finding strategies. Implementation samples the terrain into a weighted graph and uses search algorithm to compute cost-minimizing paths. This process yields realistic agent trajectories suitable for simulating crowds of agents in computer graphics.

Keywords: crowd, path-finding, environment

Název práce: Simulace davu

Autor: Vojtěch Pröschl

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: prof. RNDr. Roman Barták, Ph.D., Katedra teoretické informatiky a matematické logiky

Abstrakt: Simulace davů v členitém terénu vyžaduje plánování tras, které zohledňuje geometrické vlastnosti terénu. Tato práce představuje matematický model pohybu založený na diferenciální geometrii. Model vyhodnocuje možné trasy na hladké výškové mapě pomocí obecných cenových funkcí odvozených z geometrických vlastností cest. Model implicitně definuje optimalizační úlohu, která identifikuje realistické trasy na základě vzdálenosti, normál povrchu a směrů pohybu. Tato spojitá formulace je následně aplikována na simulaci davu pomocí diskrétního modelu a metod pro hledání cest. Implementace převádí terén na ohodnocený graf a využívá vyhledávací algoritmus k nalezení tras s minimální cenou. Tento proces vytváří realistické trajektorie agentů vhodné pro simulaci davů v počítačové grafice.

Klíčová slova: dav, hledání cest, prostředí

Contents

Introduction

Since the inception of computer graphics, researchers have aimed to create rendered motion pictures indistinguishable from real-life imagery. After decades of intensive research on rendering techniques, this goal has been achieved for a wide range of scenes. However, creating images indistinguishable from reality addresses only part of the challenge. In film production, directors often require visual effects departments to craft complex scenes involving many characters, such as the massive-scale battles depicted in movies such as *The Hobbit: The Battle of the Five Armies*, which feature thousands or tens of thousands of individual characters.

Animating even a few seconds of motion for a single character can require days or weeks of meticulous work by an experienced animator, rendering it practically impossible to animate scenes involving thousands of characters manually. Crowd simulation addresses precisely this challenge by automating the animation of large groups of virtual characters, thereby significantly reducing the extensive manual labor traditionally performed by skilled 3D artists.

Crowd simulations are used extensively across various media, including films, commercials, and video games, with each industry presenting distinct requirements. In film production, the primary goal of crowd simulations is to achieve the highest possible believability and visual fidelity. In such scenarios, longer computation times, ranging from hours to weeks, are acceptable to produce convincingly realistic results. In contrast, video game applications prioritize real-time responsiveness, enabling seamless player interaction within dynamic virtual environments. Although these objectives occasionally overlap, the differing priorities typically lead to the development of distinct algorithms optimized for visual fidelity or real-time performance.

This thesis focuses explicitly on offline crowd simulations, emphasizing the production of visually compelling and believable results suitable for cinematic applications.

Problem Formulation

The central problem addressed in this thesis is the design of a continuous terrain traversal model that can evaluate the quality of surface paths based on cost functions derived from the geometric properties of the surface paths. This model aims to be general enough to accommodate the terrain traversal preferences of various types of agents, such as human-like agents, robots, or even arbitrary artificial entities. A secondary objective is to demonstrate the feasibility of using this model to computationally generate crowd animations. Specifically, the animation will depict the motion of n agents in height-field terrain, where each agent is assigned a starting and final position.

Thesis Structure

This thesis is divided into two major parts, which are preceded by brief Chapter 1, which discusses related work, and Chapter 2, which discusses mathematical notation used throughout the thesis and provides the necessary working knowledge from differential geometry needed to understand this thesis that might not be covered in introductory calculus and real analysis courses.

Theoretical Model

The Chapter 3 operates at a theoretical level; terrain is described within the framework of differential geometry using smooth surfaces. We define paths on these surfaces connecting pairs of points and examine their geometric properties. Cost functions are introduced, with values dependent on the geometric characteristics of surface paths, particularly surface normals and tangent vectors. This approach provides a highly general and flexible model that can be tailored to specific scenarios and agent preferences.

Application to Crowd Simulation

In Chapter 4, we explain general strategies of how our model can be utilized during crowd simulation by employing appropriate discretization strategies, applying known path-finding algorithms, and interpolating discrete paths in order to generate smooth curves. We then analyze a proof of concept implementation based on the strategies discussed and perform an evaluation of the results in Chapter 5.

1 Related Work

Early crowd simulation research dates back to the late 1980s with Craig Reynolds' seminal Boids model [1]. Since then there has been significant amount of research conducted to simulate crowds. Rather than proposing a new taxonomy, we follow the structure of an existing, comprehensive survey by Yang et al. (2020) [2], which classifies crowd simulation strategies based on interaction scale. We also use representative examples of those classes, as mentioned in the survey.

Crowd Simulation Taxonomy

The taxonomy proposed by the authors consists of three different classes of crowd simulation strategies based on the scale of interaction they model. Those categories are

Microscopic Models

Microscopic models focus on low-level interactions by simulating individuals as discrete agents, each with their behaviors and responses. These models explicitly represent each entity's movement and decision-making, allowing for detailed simulations of crowd dynamics.

Microscopic models are typically further divided into several subcategories. One such subcategory consist of rule-based models, with the classic example being the Boids system [1]. The system uses a simple set of rules that guides alignment of agents, separation and cohesion. Despite their simplicity these models can produce emergent behaviors such as flocking or herding.

Another widely used subcategory are models based on social forces [3], where agents are influenced by attractive and repulsive forces representing personal goals, other agents, and environmental features. These models are popular for simulating dense crowds and pedestrian dynamics due to their continuous and physics-inspired formulation.

On the higher higher end of complexity for microscopic models it is possible to model agents by giving them virtual sensors and treating the simulation for a single agent as percept-action feedback loop. Such models are in theory extremely flexible, but defining agent function for each individual agent is very difficult process.

Macroscopic Models

Macroscopic models don't directly deal with individual agents but instead represent the entire crowd as a continuous entity and use techniques inspired, for instance, by fluid dynamics.

A noteworthy method from this category is "Continuum Crowds," [4] which is one of the first methods that demonstrated the use of macroscopic models to simulate large crowds. This method is even keepable of real-time performance.

Mesoscopic Models

Mesoscopic models are a class of models that represent a middle ground between the two extremes. They often split crowds into groups that they treat as individual entities. This allows for finer control than macroscopic models without explicitly dealing with individual agents.

Chapter Summary

This brief chapter covered only the bare essentials of crowd simulation, and to learn more, please refer to the existing literature, which does an excellent job of mapping the existing body of work such as the mentioned survey [2].

2 Mathematical Preliminaries

Before developing a mathematical model for terrain traversal and motion planning, we will introduce key concepts related to smooth functions and parameterized curves. These foundational ideas will help us to describe motion and terrain using continuous models that can be analyzed through differential geometry. We will also emphasize the relevance of these concepts to computer graphics, particularly procedural animation. While these tools are often applied intuitively in practice, we aim to define them precisely to avoid edge cases and ensure that they can be used rigorously in later chapters.

2.1 Notation

In this section, we provide a concise clarification of the notation used throughout the text to prevent potential misunderstandings. The topics addressed here are standard in undergraduate mathematics; hence, detailed explanations are omitted. We focus exclusively on the notation, not part of mathematical foundations. For instance, we do not explain the set theory notation, as it is typically widely recognized.

Definition 1 (Euclidean metric and norm [5])**.** *We define the Euclidean metric in $\mathbb{R}^n$ as the function $d : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ given by:*

$$d(p, q) := \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}.$$

Similarly, we define the Euclidean norm in $\mathbb{R}^n$ as the function $\| \cdot \| : \mathbb{R}^n \to \mathbb{R}$ given by:

$$\|p\| := \sqrt{\sum_{i=1}^{n} p_i^2}.$$

Definition 2 (Function image)**.** *Given a function $f : A \to B$ and a set $A' \subseteq A$, we use the following notation to denote the image of the set A':*

$$f[A'] := \{f(a) \mid a \in A'\}.$$

To denote the image of the function, we use the following notation:

$$\mathrm{Im}(f) := f[A].$$

Definition 3 (Extended real numbers [6, p. 133])**.** *We define the extended real numbers $\overline{\mathbb{R}}$ as follows:*

$$\overline{\mathbb{R}} := \mathbb{R} \cup \{-\infty, +\infty\}.$$

Definition 4. *For $n \in \mathbb{N}$ we define $[n] := \{1, ..., n\}$.*

Remark (Natural numbers)*.* Throughout this text, we consider the set $\mathbb{N}$ to include the number 0.

2.2 Smoothness

Intuitively, smoothness describes how easy a function is to handle, especially when considering its derivatives. This concept is fundamental in differential geometry, where many results rely on functions demonstrating consistent and predictable behavior throughout their domain.

Definition 5 (Smoothness classes [7, p. 52])**.** *Given $k \in \mathbb{N}$ and a function $f : M \to \mathbb{R}^m$, where M is an open subset of $\mathbb{R}^n$, we say that f belongs to the smoothness class C^k if and only if all partial derivatives of f of order less than or equal to k exist and are continuous on M.*

Definition 6 (Smooth function [7, p. 52])**.** *If f belongs to the class C^k for every $k \in \mathbb{N}$, then f is called a smooth function. The class of all such functions is denoted by C^∞.*

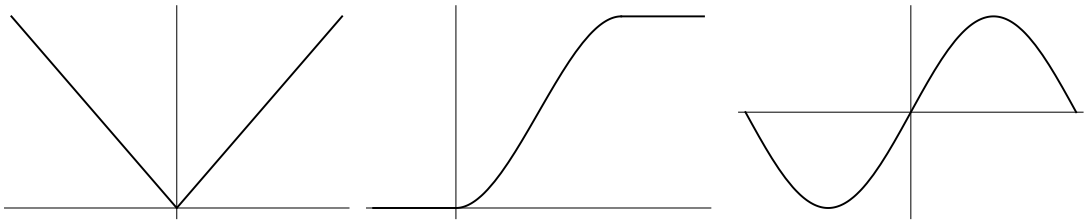
Example (Absolute value). The absolute value function $|x|$ is in the class C^0, but not in C^1. It belongs to C^0 because it is continuous. However, it is not in C^1 since the left and right derivatives at $x = 0$ differ.

Example (Smoothstep). Smoothstep is a function commonly used in computer graphics in many scenarios, for example, to interpolate between two values. Its most basic variant is defined as follows:

$$\text{smoothstep}(x) := \begin{cases} 0 & x \leq 0 \\ -2x^3 + 3x^2 & 0 < x < 1 \\ 1 & x \geq 1. \end{cases}$$

Definition of this function is based on [8, p. 27]. Smoothstep has continuous first-order derivatives everywhere. However, it is not twice differentiable at $x = 0$ and $x = 1$ due to a mismatch in the one-sided second-order derivatives. Therefore, it belongs to class C^1 but not C^2.

Example (Sine). A classical example of a function from C^∞ is the function sin. Inductively, it can be shown that all its derivatives exist and are continuous for all $x \in \mathbb{R}$.



(a) C^0—Absolute value **(b)** C^1—Smoothstep **(c)** C^∞—Sine

Figure 2.1 Functions of various smoothness levels

Remark (Smoothness in differential geometry). In differential geometry, functions are often assumed to be smooth, which means that they belong to the class C^∞. For example, Pressley [9] discusses curves and surfaces under this assumption, which simplifies the analysis by avoiding issues that can arise from functions with limited differentiability. Although this approach is common, more advanced texts consider different levels of smoothness.

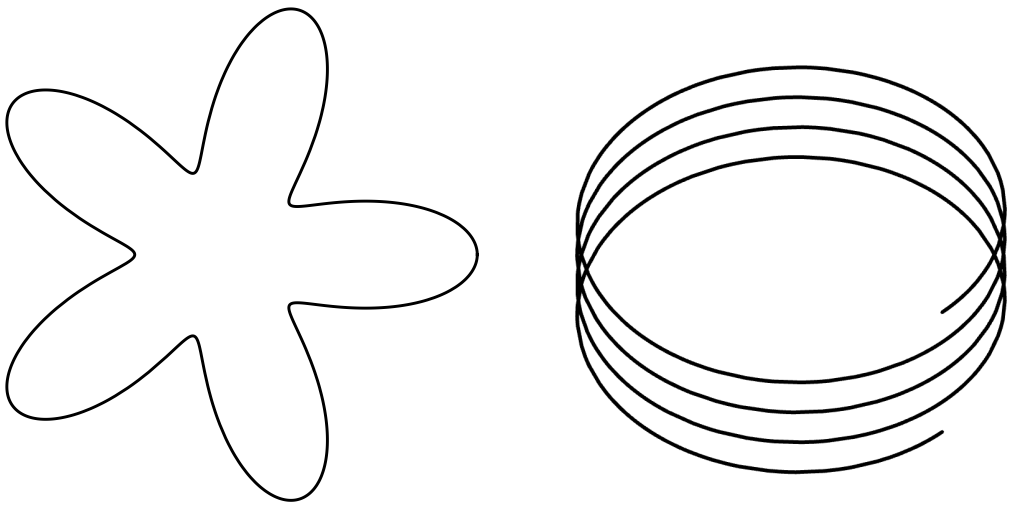
Remark (Smoothness in computer graphics). In computer graphics, functions often exhibit finite smoothness, such as C^0, C^1, or C^2. This characteristic complicates the direct application of the results of classical differential geometry. Although many theoretical results could be extended to functions with limited differentiability, this thesis adopts C^∞ smoothness as a standard assumption to maintain consistency with the established literature.

Remark (Justifying the smoothness assumption). The assumption of smoothness may seem disconnected from real-world applications at first glance. However, many practical functions can be approximated by smooth functions with arbitrary precision. Although the precise definitions of "reasonable" functions, methods for measuring approximation error, and techniques for constructing such approximations are beyond the scope of this thesis, they are well-established in mathematics.

For those interested, signal processing offers foundational techniques for smoothing specific function classes, with many computer graphics textbooks, such as [10], covering practical implementations. For a more abstract perspective, differential topology provides a rigorous framework for smooth approximations. Hirsch's textbook on differential geometry [11] is an excellent resource for these theoretical foundations.

2.3 Fundamentals of Curves

Curves are mathematical objects that are particularly well suited to encode motion. They naturally represent position as a function of time and derived quantities such as velocity, direction, and more. Studying curves provides a natural language for describing movement in both theoretical and applied contexts.



(a) Flower **(b)** Helix

Figure 2.2 Examples of curves

Definition 7 (Parametrized curve)**.** *Given an open interval $(\alpha, \beta) \subseteq \mathbb{R}$ with $\alpha, \beta \in \overline{\mathbb{R}}$ and $\alpha < \beta$, a parametrized curve is a smooth function $\varphi : (\alpha, \beta) \to \mathbb{R}^n$. This definition follows the presentation in [9, Chapter 1].*

Remark (Smoothness of curves). Technically, one could define a parametrized curve as merely a continuous function and subsequently distinguish between continuous and smooth curves. However, this would introduce unnecessary complexity in the notation and definitions used throughout this thesis. We therefore assume all curves to be smooth unless stated otherwise.

Example (Flower). We define a planar flower curve $\varphi_F : (0, 2\pi) \to \mathbb{R}^2$ as:k

$$\varphi_F(t) = \left(\frac{\cos(t)(\cos(5t) + 2)}{2}, \frac{\sin(t)(\cos(5t) + 2)}{2} \right).$$

In Figure 2.2a, we can see an illustration of the curve, representing its image $\mathrm{Im}(\varphi_F)$.

Example (Helix). A helix is a type of spatial curve that can be defined, for example, by the function $\varphi_H : (0, 2\pi) \to \mathbb{R}^3$ as:

$$\varphi_H(t) = (\cos(4t), \sin(4t), t/8)\,.$$

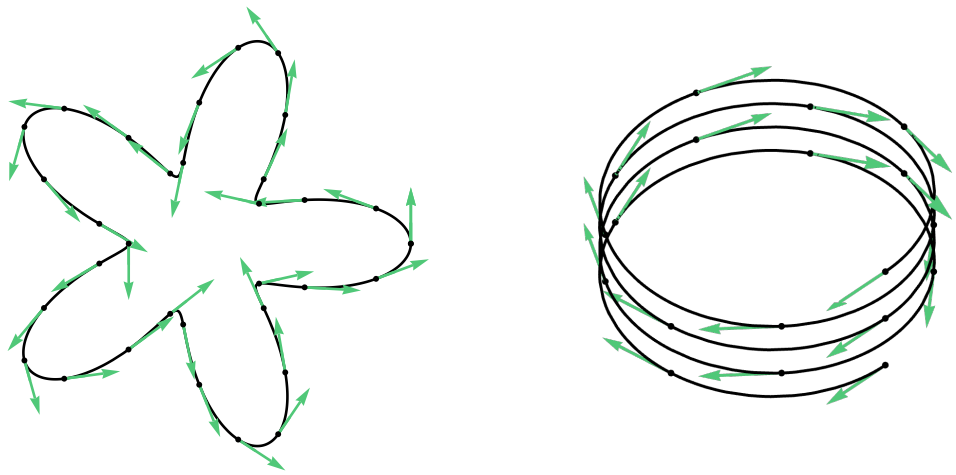
Similarly, the image of the helix, $\mathrm{Im}(\varphi_H)$, is illustrated in Figure 2.2b.

2.4 Properties of Curves

Curves are incredibly useful because they naturally encode meaningful geometric and physical properties. In this section, we introduce several core properties that will be indispensable in later chapters. For a more comprehensive treatment, we recommend [9].

Tangent Vector

Definition 8 (Tangent vector)**.** *Given a parametrized curve $\varphi : I \to \mathbb{R}^n$, we call its first derivative $\varphi'(t)$ the tangent vector of φ at the point $\varphi(t)$. See [9, Chapter 1].*



(a) Tangent vectors of the flower **(b)** Tangent vectors of the helix

Figure 2.3 Examples of tangent vectors (tangents are scaled)

Remark (The meaning of the tangent vector). The tangent of a curve encodes the direction in which the curve is heading at a specific point in time. The norm of this vector encodes the speed at a given point.

Example (Tangent vectors of the flower). The tangent vectors of φ_F are given by:

$$\varphi'_F(t) = \left(-\sin(t) - \sin(4t) - \frac{3}{2}\sin(6t), \cos(t) - \cos(4t) + \frac{3}{2}\cos(6t)\right)$$

Few sample vectors of the flower curve are illustrated by Figure 2.3a. To derive this formula, we compute the derivative of each output coordinate with respect to t. The computation itself is not particularly interesting. More notable is that we obtain two elementary functions, one for each coordinate. Each of these functions describes one coordinate of the tangent vector.

Example (Tangent vectors of the helix). The tangent vectors of φ_H are defined as follows:

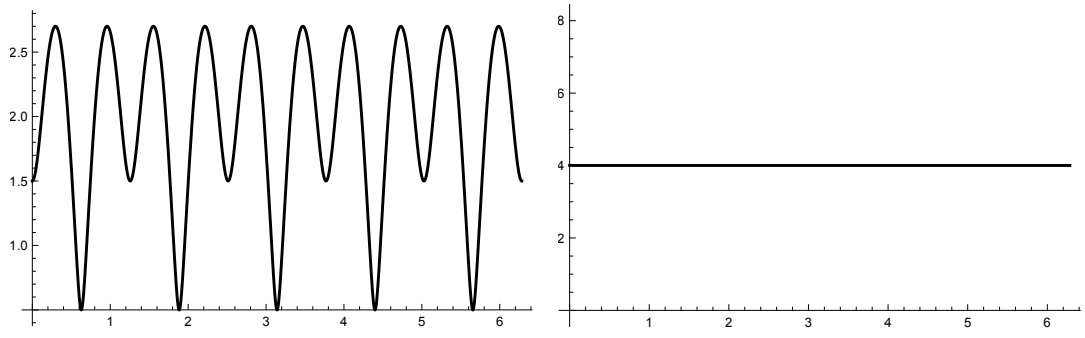
$$\varphi'_H(t) = \left(-4\sin(4t), 4\cos(4t), \frac{1}{8}\right).$$

The only difference here is that we compute three derivatives instead of two. As with the flower curve Figure 2.3b shows multiple instances of tangent vectors of the helix.

Speed

Definition 9 (Speed of a curve)**.** *Assume that $\varphi : I \to \mathbb{R}^n$ is a parametrized curve. For $t \in I$, we define its speed at t as $||\varphi'(t)||$. See [9, Chapter 1]*

Remark (Why speed matters). Speed is a crucial parameter of any motion. For example, when generating animation based on a curve, if the speed at a certain point is 5 kilometers per hour, it would be appropriate to generate a walking animation. In contrast, at a speed of 15 kilometers per hour, it would be better to generate running animation.



(a) Flower **(b)** Helix

Figure 2.4 Speed of curves over time

Example (Speed of a flower). The speed of φ_F at t is given by:

$$||\varphi'_F(t)|| = \sqrt{\cos(5t) - 3\cos(10t) + \frac{17}{4}}.$$

Example (Speed of a helix). The speed of φ_H at t is given by:

$$||\varphi'_H(t)|| = \frac{5\sqrt{41}}{8}.$$

It is intriguing to note that the helix maintains a constant speed, whereas the flower exhibits variable speed over time. Characterizing curves in terms of speed is highly beneficial in both theoretical and practical contexts. Consequently, we introduce the notions of unit speed and curve regularity, both of which are grounded in speed.

Curve Properties Based on Speed

Definition 10 (Unit speed curve)**.** *A parametrized curve $\varphi : I \to \mathbb{R}^n$ is considered unit speed if $\forall t \in I : ||\varphi'(t)|| = 1$. See [9, Chapter 1].*

Remark (Why are unit speed curves useful). Unit speed curves can be modified effortlessly to alter their speed. For example, if we have a unit speed curve φ and wish to double the speed, we can define a new curve ψ by setting $\psi(t) := \varphi(2t)$. Speed adjustments are not limited to straightforward scalar multiplication; more intricate functions can be composed with unit speed curves to achieve diverse behaviors.

Definition 11 (Regular curve)**.** *A parametrized curve $\varphi : I \to \mathbb{R}^n$ is considered regular if $\forall t \in I : ||\varphi'(t)|| \neq 0$. See [9, Chapter 1].*

Remark (Why are regular curves useful). This concept refers to curves that maintain a speed greater than zero. Although this idea is abstract, one might find a curve that decelerates to zero in practical scenarios. Nevertheless, as we will explore in the upcoming section on reparametrization, regular curves are useful, as they allow us to obtain a unit speed curve that traces the same path as a given regular curve.

Arc Length

Last property we will cover in this section is arc length. Intuitively it describes the distance we would travel while moving along the curve.

Definition 12 (Arc length)**.** *Assume that $\varphi : (\alpha, \beta) \to \mathbb{R}^n$ is a parametrized curve. Then, we define its arc length as:*

$$L(\varphi) = \int_\alpha^\beta ||\varphi'(t)||\, dt.$$

For a deeper analysis, see [9, Chapter 1]

Remark (About the definition). From geometric perspective we are integrating over the curve speed, which is informally speaking the reason why it represents its length. Observe that our parameterized curves are smooth, and thus the norm is always well defined. Given that the norm consistently yields a nonnegative result, the integral necessarily must be nonnegative too. It is implicitly understood that this integral can equate to $+\infty$; this scenario arises, for example, in the case of unit speed curves defined over $\mathbb{R}$. Nevertheless, within this thesis, we predominantly focus on curves possessing a finite arc length.

Example (Flower arc length).

$$L(\varphi_F) = \int_0^{2\pi} \sqrt{\cos(5t) - 3\cos(10t) + \frac{17}{4}}\, dt \approx 12.329.$$

Computing this integral symbolically is a non-trivial task, but numerical evaluation is straightforward. This is often the case with parametric curves; although exact expressions can be complex or even unattainable, numerical integration provides an efficient and reliable alternative when a formula is known.

Example (Helix arc length).

$$L(\varphi_H) = \int_0^{2\pi} \frac{5\sqrt{41}}{8}\, dt = \frac{5\sqrt{41}\pi}{4} \approx 25.145.$$

2.5 Describing Planar Motion With Curves

As we have mentioned previously, curves serve as a superb method to represent motion. In this section, we will explore how parametrized curves represent motion in $\mathbb{R}^2$, and in the following chapter, we will expand this to surfaces embedded in $\mathbb{R}^3$. Representing the position of an object with a curve is straightforward, since the curve directly provides the position at a specific parameter value. However, to fully depict motion in $\mathbb{R}^2$, it is necessary to also describe orientation. Orientation can be addressed in several ways; for instance, one might set a shape's facing direction and express the rotation angle required to reach it. However, a more practical method often involves constructing an orthonormal basis, which is precisely what we will undertake.

Motion Basis

Definition 13 (Planar curve motion basis)**.** *Let $\varphi : I \to \mathbb{R}^2$ be a regular parametrized curve. We establish a local orthonormal coordinate basis at any point $t \in I$ by proceeding as follows. The initial basis vector is described by the function $e_X^{\varphi}(t)$, which indicates the normalized tangent vector at the point t. Meanwhile, the second basis vector function, denoted by $e_Y^{\varphi}(t)$, is constructed by rotating $e_X^{\varphi}(t)$ counterclockwise by an angle of $\pi/2$.*

Example (Flower motion basis). Given the above definitions and the flower curve φ_F, we can derive its basis as follows:

$$e_X^{\varphi_F}(t) = \left(-\frac{2\sin(t) + 2\sin(4t) + 3\sin(6t)}{\sqrt{4\cos(5t) - 12\cos(10t) + 17}}, \frac{2\cos(t) - 2\cos(4t) + 3\cos(6t)}{\sqrt{4\cos(5t) - 12\cos(10t) + 17}} \right),$$

$$e_Y^{\varphi_F}(t) = \left(\frac{-2\cos(t) + 2\cos(4t) - 3\cos(6t)}{\sqrt{4\cos(5t) - 12\cos(10t) + 17}}, -\frac{2\sin(t) + 2\sin(4t) + 3\sin(6t)}{\sqrt{4\cos(5t) - 12\cos(10t) + 17}} \right).$$

We have employed Wolfram Mathematica [12] to attain these formulas. Deriving explicit basis formulas for various curves relevant to applications is significant. This enables the encoding of motion using those formulas, which will be further discussed.

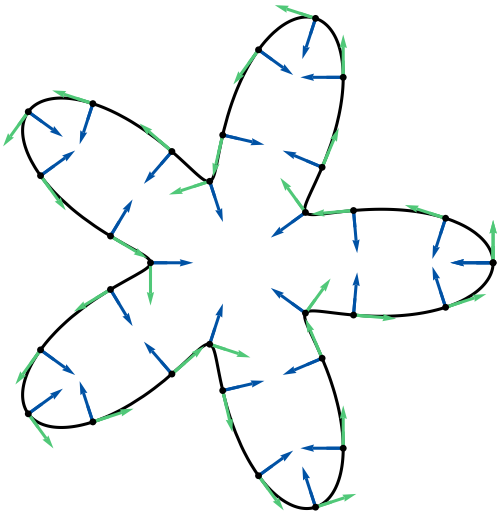


Figure 2.5 Flower curve motion basis (basis vectors were scaled)

Animating Shapes

Example (Square). Let us define a square shape that we will use to demonstrate how exactly we can generate animations given the definitions above. We define a square as:

$$S_Q := [-1/8, 1/8] \times [-1/8, 1/8].$$

Definition 14 (Curve based animation)**.** *Given a set $S \subseteq \mathbb{R}^2$ representing a certain shape and a regular curve φ, we can animate the motion of this shape along the curve as follows:*

$$S^{\varphi}(t) := \{p_1 e_X^{\varphi}(t) + p_2 e_Y^{\varphi}(t) + \varphi(t) | p \in S\}.$$

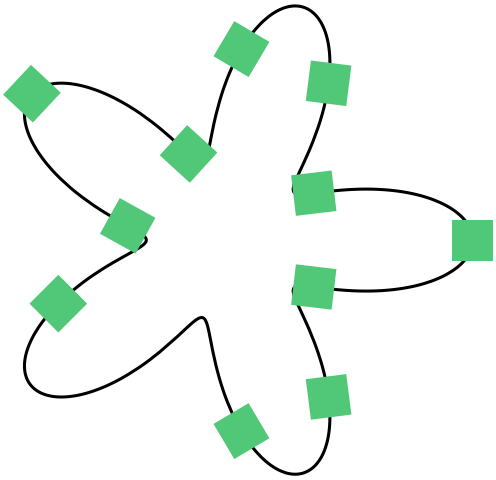


Figure 2.6 Flower-based animation

Remark (How does curve-based animation work). The core idea is to rotate the shape to align with the local coordinate basis of the curve at time t, then translate it to the corresponding point on the curve. This mimics the effect of the shape moving along the path and changing its orientation to always face forward.

Example (Flower-based animation). Using the square S_Q and the flower curve φ_F with different speed, we can generate a shape for every parameter value within the domain of φ_F using $S_Q^{\varphi_F}$. Visualizing animations with pictures isn't optimal, therefore Figure 2.6 shows positions of the rectangle at various values of input parameter.

2.6 Manipulating Curves

In the previous section, we observed that regular curves naturally represent animation. As noted in Remark 2.4, we discussed the ability to alter the speed of a curve. This raises the question: how can we modify curves to represent various animations? Changing speed is a form of reparameterization, which will be the primary topic of this chapter. Nevertheless, we will also touch upon some advanced techniques that are useful for procedural animation.

Reparametrization

Definition 15 (Reparametrization)**.** *Given parametrized curves $\varphi : I \to \mathbb{R}^n$ and $\psi : J \to \mathbb{R}^n$, we say that ψ is a reparametrization of φ if there exists a smooth, bijective map $\phi : I \to J$ such that the inverse map $\phi^{-1} : J \to I$ is also smooth and*

$$\forall t \in I : \varphi(t) = (\psi \circ \phi)(t).$$

For a more thorough treatment, see [9].

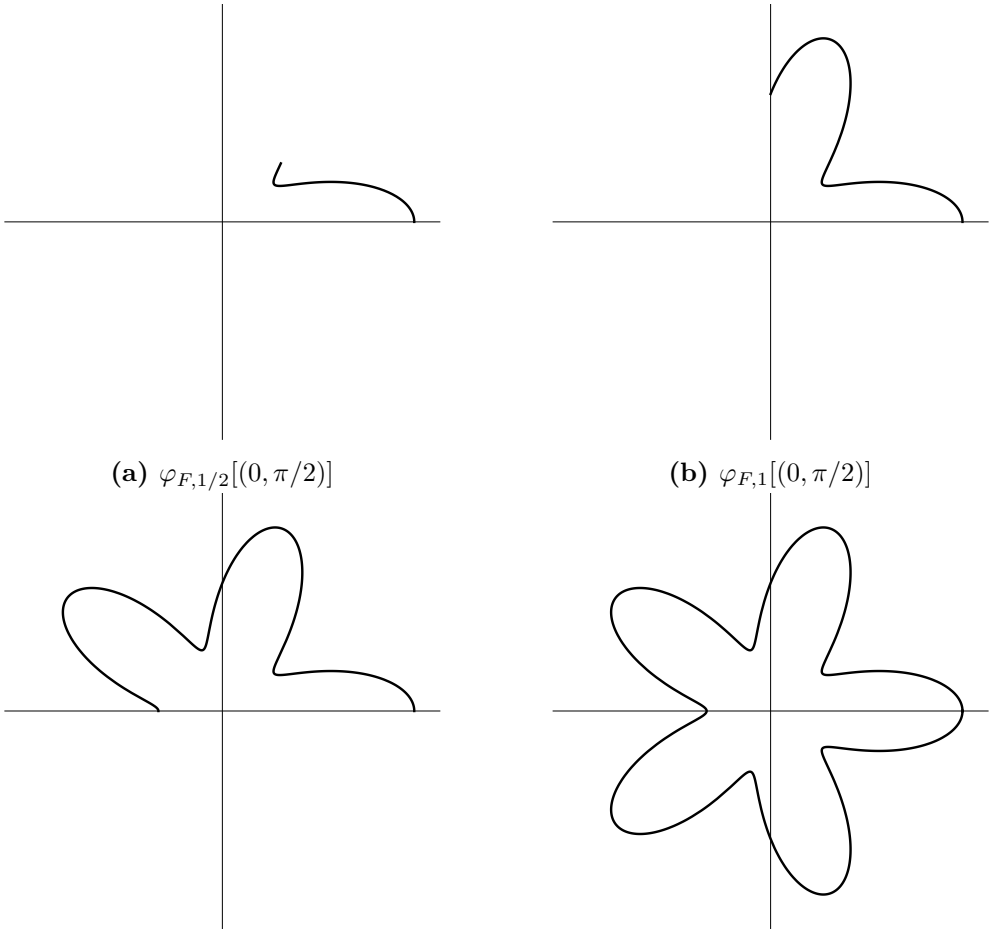
Example (Flower Reparametrization). We can create an entire class of curves that are reparametrizations of the curve φ_F by taking $\alpha \in \mathbb{R}^+$ and defining a curve $\varphi_{F,\alpha} : (0, \frac{2\pi}{\alpha}) \to \mathbb{R}^2$ by:

$$\varphi_{F,\alpha}(t) = \varphi_F(\alpha t).$$

Figure 2.7 demonstrates how altering the parametrizations influences the behavior of the curves by considering the image of a subset of their domains. More interesting results can be achieved by doing nonlinear changes to the input parameter, one such case is unit reparameterization, which is a nonlinear change for curves whose speed is not constant. We will explore this through Theorem 1.

Unit Reparametrization

Theorem 1 (Reparametrization theorem)**.** *A parametrized curve φ has a unit-speed reparametrization if and only if it is regular. We will not detail the proof here as it can be found in [9], however, we will go through unit reparametrization example of our helix curve, which demonstrates the main idea behind the proof, and practical application of this theorem.*



(a) $\varphi_{F,1/2}[(0,\pi/2)]$ **(b)** $\varphi_{F,1}[(0,\pi/2)]$

(c) $\varphi_{F,2}[(0,\pi/2)]$ **(d)** $\varphi_{F,4}[(0,\pi/2)]$

Figure 2.7 Reparametrizations of the flower curve

Example (Helix unit reparametrization). The most interesting part of the proof is the way in which the unit-speed parameterization is constructed. Given our helix function $\varphi_H : (0, 2\pi) \to \mathbb{R}^3$, we construct a function $s : (0, 2\pi) \to \mathbb{R}^+$ as follows:

$$s(t) = \int_0^t ||\varphi_H(x)||\, dx = \frac{5\sqrt{41}t}{8}.$$

The function s remaps points from the domain $(0, 2\pi)$ of φ_H to the domain $\left(0, \frac{5\pi\sqrt{41}}{4}\right)$ of the unit-speed parametrized curve. Notice that the upper bound of the interval is equal to the helix's arc length. Let us call the reparametrized curve ψ_H. Its values are defined as follows:

$$\psi_H(t) = \varphi_H(s^{-1}(t)) = \varphi_H\left(\frac{8t}{5\sqrt{41}}\right) = \left(\sin\left(\frac{5\sqrt{41}t}{2}\right), \cos\left(\frac{5\sqrt{41}t}{2}\right), \frac{5\sqrt{41}t}{64}\right).$$

If we compute the speed of this new curve, we get:

$$||\psi'_H(t)|| = \left|\left|\left(\frac{32\cos\left(\frac{32t}{5\sqrt{41}}\right)}{5\sqrt{41}}, -\frac{32\sin\left(\frac{32t}{5\sqrt{41}}\right)}{5\sqrt{41}}, \frac{1}{5\sqrt{41}}\right)\right|\right|_2 = 1.$$

Thus, we can see that this curve is, in fact, a unit speed.

Advanced manipulations

Advanced curve manipulations frequently involve creating new curves derived from preexisting ones. We will illustrate this type of manipulation by using our flower curve as a foundation and adjusting its position with a different curve.

Example (Oscillating flower curve). We will construct the oscillating flower curve $\varphi_F^O : (0, 2\pi) \to \mathbb{R}^2$ as follows:

$$\varphi_F^O(t) := \varphi_F(t) + \frac{1}{10}(\cos(50t), \sin(50t)).$$

The concept behind the construction involves using our flower curve as the primary determinant of position, with a slight modification made using a circular curve defined by the formula $\frac{1}{10}(\cos(50t), \sin(50t))$. The outcome of this adjustment is illustrated in Figure 2.8.

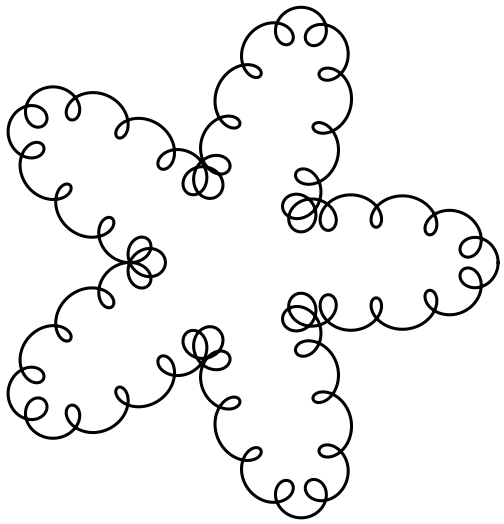


Figure 2.8 Oscillating flower curve

Remark (Extending this further). To expand on this concept, it is not necessary to use the parameter t directly to "control" the second curve. Instead, attributes of the first curve can be utilized, including its speed, tangent direction, or curvature, which essentially indicates the degree of turning of the curve. These attributes are detailed in the differential geometry textbook by Pressley [9]. We encourage the reader to explore how these intrinsic properties can be harnessed creatively to generate new curve constructions, as this often leads to elegant and surprising results.

Chapter Summary

In summary, this chapter has covered the mathematical instruments essential for future chapters. These range from smoothness and curve fundamentals to foundational principles for crafting animation with curves. These ideas are key for describing animation curves in our simulations and are foundational in assessing

how effectively an agent can navigate a given path, which will be examined in the following chapter. In subsequent chapters, we will use these to develop a technique for generating animation paths for both groups and individual agents.

3 Theoretical Terrain Traversal Model

The preceding chapter presented a framework for describing planar motion via parameterized curves. This chapter broadens the discussion to encompass motion on $\mathbb{R}^2$ surfaces embedded in $\mathbb{R}^3$. These surfaces are modeled using smooth height maps, which allow us to apply differential geometry for precise analysis. Such representations effectively depict various real-world terrains, including hills, valleys, and mountainous regions. The curves on these surfaces represent the paths of agents traversing the terrain. To assess the feasibility of these paths, we introduce traversal cost functions that quantify the effort needed to navigate particular routes. These functions consider factors such as slope steepness and heading direction, thus providing a systematic approach to optimize movement over intricate surfaces.

3.1 Terrain Geometry

This section presents a smooth, height map-based terrain representation. We define the surface in such a way that it is suitable for analysis using differential geometry.

Terrain

Definition 16 (Terrain)**.** *A terrain is a smooth function T of the form $T : D \to \mathbb{R}$, where $D \subseteq \mathbb{R}^2$, satisfying the following:*

1. *D is an open set,*
2. *T is a smooth map,*
3. *for every $a, b \in D$, there exists a continuous map $\varphi : [0, 1] \to D$ such that φ is smooth on $(0, 1)$ and $\varphi(0) = a$, $\varphi(1) = b$.*

Remark (Why these conditions)*.* The first condition requires the domain to be open, a technical assumption necessary to define derivatives everywhere in D. The second condition guarantees smoothness, allowing us to apply fundamental tools from differential geometry without handling edge cases. The third condition intuitively ensures that the domain is path-connected by smooth curves in the interior of their domains.

Remark (Strength of conditions)*.* It may be possible to remove the first condition since the smoothness requirement implicitly assumes the domain is open. Additionally, the third condition could potentially be weakened to remove the requirement of smoothness in the interior. One possible way to justify this could be to use the Whitney embedding theorem [11]. However, exploring this further lies well beyond the scope of this thesis.

Example (Hill). Consider the function $T_H : \mathbb{R}^2 \to \mathbb{R}$ defined by:

$$T_H(x, y) = \frac{\exp(-10y^2)}{x^2 + 1}.$$

This function satisfies our terrain definition: $\mathbb{R}^2$ is open, T_H is infinitely differentiable (hence smooth), and any two points in the domain can be connected by a straight line, the interior of which is smooth. Beyond satisfying formal conditions, this function is also meaningful from a didactic perspective: a human-like agent would find it very difficult to traverse the shortest surface path φ between $(0, 2, 0)$ and $(0, 0, 1)$. Traversing path ψ would be much easier since its slope is gentler, even though the path is much longer.

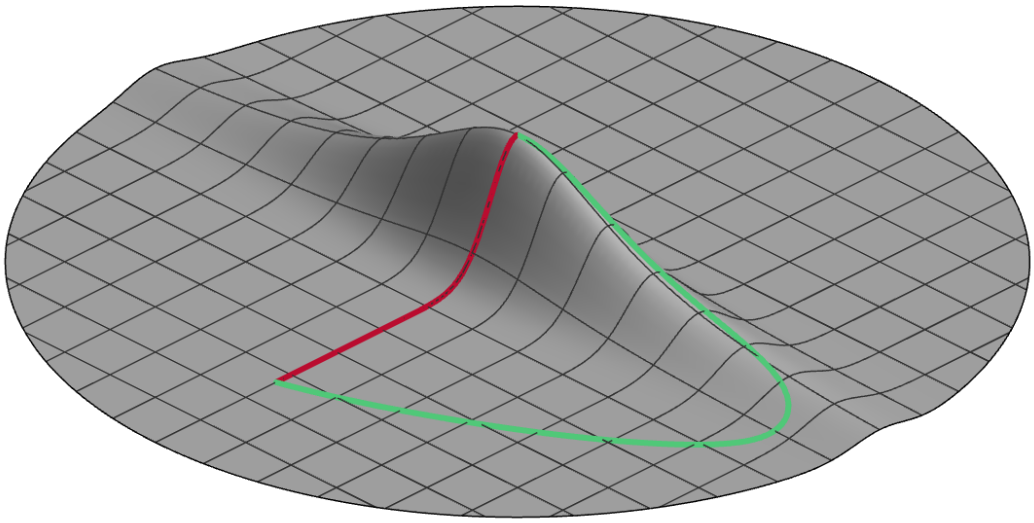


Figure 3.1 Hill Illustration

Terrain Surface

To investigate the geometric properties of our terrain, it is essential to first define it as a two-dimensional surface embedded in a three-dimensional space. This is accomplished through the following definition.

Definition 17 (Terrain surface)**.** *Let $T : D \to \mathbb{R}$ be a terrain. Then, we define the surface associated with the terrain as the set $\mathcal{S}(T)$ given by:*

$$\mathcal{S}(T) = \{(x, y, T(x, y)) \mid (x, y) \in D\}.$$

Surface Normals

Before we define the curves on surfaces and their associated cost functions, we introduce a key geometric concept that allows us to measure the steepness of the terrain: the surface normal. Intuitively, the normal surface at a point is a vector perpendicular to the surface at that point. Apart from steepness normal direction will also enable us to derive motion basis for agents moving on this surface.

Definition 18 (Terrain surface normals)**.** *Given a terrain map $T : D \to \mathbb{R}$ we define surface at point $(a, b) \in D$ as N_T. For clarity purposes, we will use auxiliary notation:*

$$v_x(a,b) = \left(1, 0, \frac{\partial T}{\partial x}(a,b)\right),$$

$$v_y(a,b) = \left(0, 1, \frac{\partial T}{\partial y}(a,b)\right),$$

$$N_T(a,b) = \frac{v_x(a,b) \times v_y(a,b)}{||v_x(a,b) \times v_y(a,b)||}.$$

This definition is based on the standard construction of surface normals in differential geometry [9, p. 89]. However, the notation and naming conventions presented here are adapted to fit the height map-based terrain representation introduced in this thesis.

Remark (How does the normal definition work). The vector $v_x(a, b)$ represents the tangent to the surface in the x-direction, and $v_y(a, b)$ represents the tangent in the y-direction. Their cross product yields a vector orthogonal to both - that is, a vector orthogonal to the surface. We normalize this vector to ensure it has unit length.

Example (Hill normals). Given the definition above, normals of T_H are given by:

$$N_{T_H}(x,y) = \frac{\left(\frac{2xe^{-10y^2}}{(x^2+1)^2}, \frac{20e^{-10y^2}y}{x^2+1}, 1\right)}{\left|\left|\left(\frac{2xe^{-10y^2}}{(x^2+1)^2}, \frac{20e^{-10y^2}y}{x^2+1}, 1\right)\right|\right|}.$$

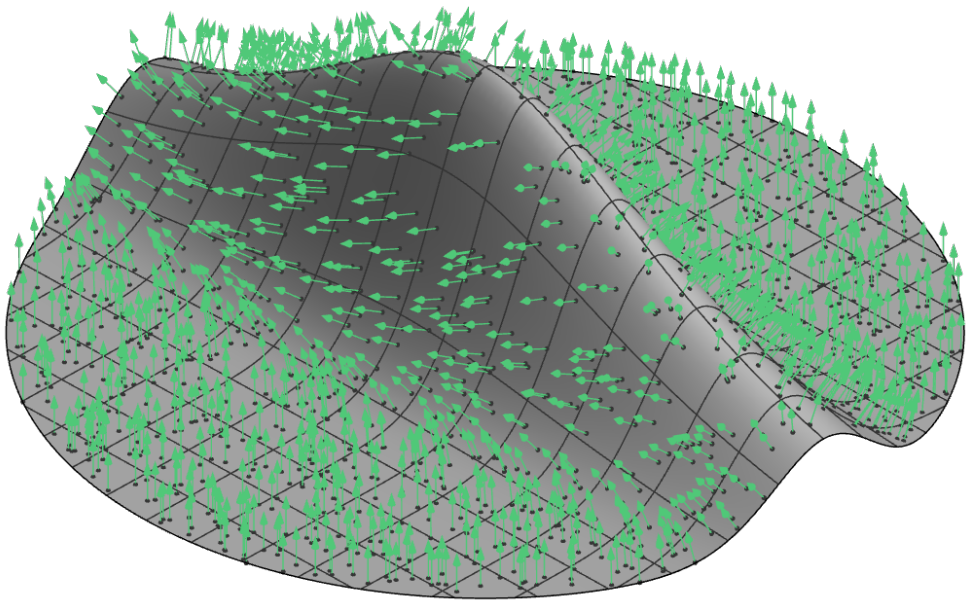


Figure 3.2 Hill normals (normals were scaled)

3.2 Smooth Path Existence

Before we start working with curves on terrain surfaces, there's a crucial technical detail we must consider. Does our definition ensure that smooth paths can be found between points on a surface? We need such paths to describe motion, and while it might seem self-evident that they exist, improper handling can result in the absence of such paths. To rigorously establish their presence, we depend on two standard results from mathematical analysis.

Auxiliary Theorems

Theorem 2 (Composition preserves continuity)**.** *Let $f : X \to Y$ and $g : Y \to Z$, where $X \subseteq \mathbb{R}^n$, $Y \subseteq \mathbb{R}^m$, and $Z \subseteq \mathbb{R}^k$. If f is continuous at $x \in X$ and g is continuous at $f(x)$, then the composition $g \circ f : X \to Z$ is continuous at x. For more details, see [5, p. 29].*

Remark (On generality). Tao states this theorem in the general context of arbitrary metric spaces. However, we only require it in the Euclidean setting, so the above formulation is a special case of Tao's formulation.

Corollary. Given the notation of Theorem 2, if the functions f and g are continuous (i.e., continuous at every point of their respective domains), then their composition $g \circ f$ is also continuous. This follows immediately by applying Theorem 2 pointwise.

Theorem 3 (Composition of C^k functions)**.** *Let $A \subseteq \mathbb{R}^m$ and $B \subseteq \mathbb{R}^n$ be open sets. Let $f : A \to \mathbb{R}^n$ and $g : B \to \mathbb{R}^p$, where $f[A] \subset B$. If f and g are of class C^k, then the composition $g \circ f \in C^k$. For more details, see [13, p. 58].*

Smooth Path Existence

Theorem 4 (Smooth terrain surface path existence)**.** *Assume $T : D \to \mathbb{R}$ is a terrain. Then for any $a, b \in \mathcal{S}(T)$, there exists a continuous map $\varphi : [0, 1] \to \mathcal{S}(T)$ such that $\varphi(0) = a$, $\varphi(1) = b$, and the restriction of φ to $(0, 1)$ is smooth.*

Remark. Before proving this theorem, let us first clarify its meaning. This theorem guarantees that for any two points on our terrain, we can find a continuous path connecting them that is also smooth on its interior. The existence of such a path is central to the motion planning problem: continuity ensures the points are always connected, while smoothness guarantees that we can apply the differential geometry toolkit developed in the previous chapter. If we didn't know that such a path existed, motion planning on our surface would be ill-defined.

Proof. Let $a, b \in \mathcal{S}(T)$. By definition of the terrain surface, we have:

- $a = (a_1, a_2, T(a_1, a_2))$,
- $b = (b_1, b_2, T(b_1, b_2))$.

Since T is a terrain, there exists a continuous map $\psi : [0, 1] \to D$ such that:

- $\psi(0) = (a_1, a_2)$,

- $\psi(1) = (b_1, b_2)$,
- ψ is smooth on $(0, 1)$.

We define the surface projection map $P_T : D \to \mathcal{S}(T)$ by:

$$P_T(x, y) = (x, y, T(x, y)).$$

Since T is smooth, P_T is smooth and therefore continuous. Now we define $\varphi = P_T \circ \psi$, then:

- φ is continuous by Corollary 3.2,
- φ is smooth on $(0, 1)$ by Theorem 3,
- $\varphi(0) = P_T(\psi(0)) = a$,
- $\varphi(1) = P_T(\psi(1)) = b$.

Hence, the desired path exists. □

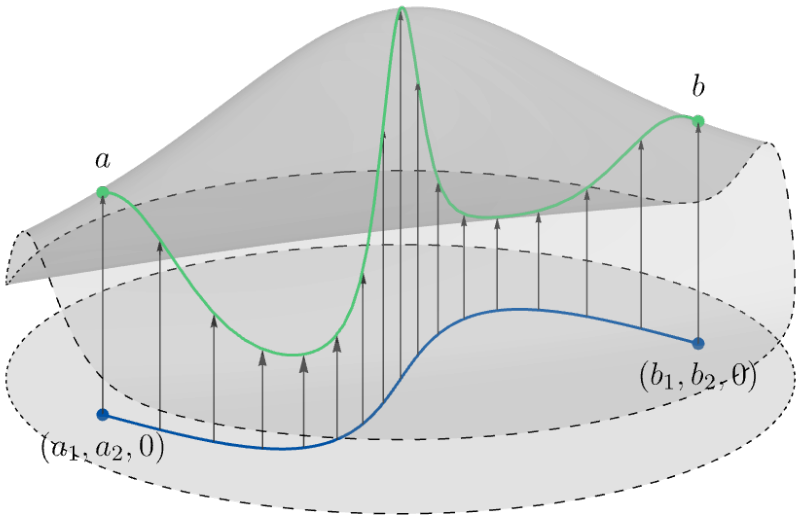


Figure 3.3 Illustration of the core idea behind the proof of Theorem 4.

Practical Remarks

Remark. This proof also provides a procedure to construct surface paths from domain paths, which is often very useful. We will utilize this procedure on multiple occasions.

Remark (Path Regularity). In our definition, we have omitted one potential edge case. This construction could yield a path that is not regular. Such paths would not be very useful, since we are interested in constructing tangents. We could explicitly address this issue by extending the definition. Alternatively, it may be possible to prove it from our current definition. However, to keep the scope of this thesis manageable, we omit this detail.

3.3 Describing Surface Motion With Curves

Because of the path existence theorem, we can find a well-behaved path between any two points on the terrain surface. We can now consider a class of all such paths and study their properties. We will be interested almost exclusively in paths, that are regular as well.

Remark (What is different from planar motion). In $\mathbb{R}^2$, a curve's normalized tangent defines two possible local bases by picking one of two perpendicular vectors. However, in $\mathbb{R}^3$, a single tangent vector is insufficient to define a full orthonormal basis unambiguously. There are infinitely many ways to choose the remaining two vectors, most of which do not produce visually meaningful animations.

Central example

Example (Deformed semi-circle). To explore the characteristics of terrain paths and effectively present forthcoming concepts, we propose a straightforward yet versatile class of parametrized curves. These curves, which we call deformed semi-circles, serve as illustrative examples for the examination of surface navigation, cost calculation, and motion visualization. The images of these curves are contained within the domain of our hill function T_H. Given $\alpha, \beta \in \mathbb{R}^+$, we define the curve $\varphi_{\alpha,\beta} : [0, \pi] \to \mathbb{R}^2$ as follows:

$$\varphi_{\alpha,\beta}(t) = (\alpha \cos(t), \beta \sin(t)).$$

Throughout the remainder of this chapter, we will employ six representative instances of these deformed semi-circles to demonstrate key ideas and visualize path traversal under varying terrain conditions. Specifically, we select the following parameterizations:

$$\varphi_{2,2}, \quad \varphi_{2,3/2}, \quad \varphi_{2,1}, \quad \varphi_{2,2/3}, \quad \varphi_{2,1/3}, \quad \varphi_{2,1/10}.$$

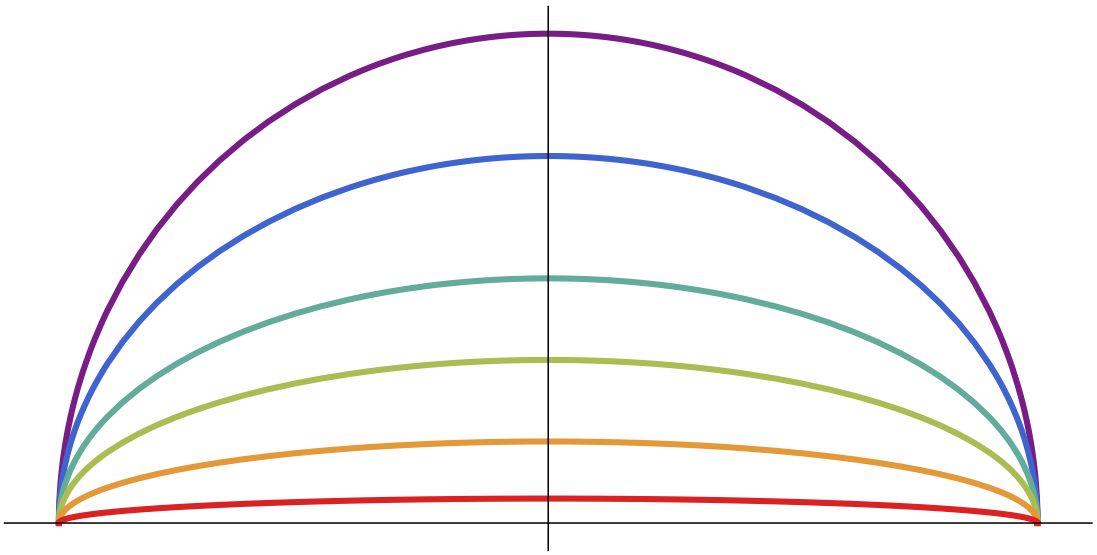
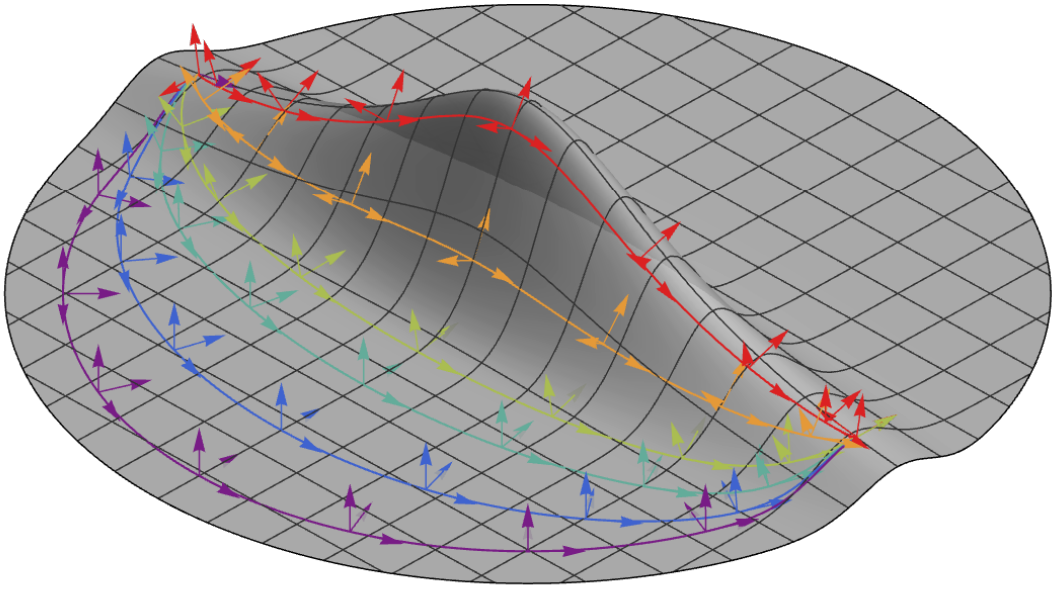


Figure 3.4 Deformed semi-circles

Motion Basis

Remark (Frenet-Serret basis). One classical solution is to use the Frenet-Serret equations [9, p. 51]. If the curve meets certain criteria, it can define a well-behaved orthonormal basis. However, such construction might work sub-optimally on our curves, which are embedded on a surface.

Remark (Using surface normal). Instead of using the Frenet-Serret equations, we will utilize the surface normal to obtain the second basis vector. This approach still leaves us with two possible choices for the third vector to complete the orthonormal basis.



(a) $\psi_{2,2}$ $\psi_{2,3/2}$ $\psi_{2,1}$ $\psi_{2,2/3}$ $\psi_{2,1/3}$ $\psi_{2,1/10}$

Figure 3.5 Deformed semi-circles and their surface motion bases

Definition 19 (Surface motion basis)**.** *Given a terrain map $T : D \to \mathbb{R}$, a terrain surface $\mathcal{S}(T)$, and a smooth, regular curve $\varphi : I \to \mathcal{S}(T)$, we define its basis as follows:*

$$e_X^\varphi(t) = \frac{\varphi'(t)}{||\varphi'(t)||_2},$$

$$e_Y^\varphi(t) = e_X^\varphi(t) \times e_Z^\varphi(t),$$

$$e_Z^\varphi(t) = N_T(\varphi(t)_1, \varphi(t)_2).$$

Example (Deformed semi-circle hill surface basis). To embed our deformed semi-circles onto the hill terrain surface, we apply the same construction used in the proof of Theorem 4. Given a curve $\varphi_{\alpha,\beta} : [0, \pi] \to \mathbb{R}^2$, we define its surface projection $\psi_{\alpha,\beta} : [0, \pi] \to \mathcal{S}(T_H)$ as:

$$\psi_{\alpha,\beta}(t) = \left(\alpha \cos(t), \beta \sin(t), \frac{\exp(-10\beta^2 \sin^2(t))}{\alpha^2 \cos^2(t) + 1}\right).$$

Using the curve $\psi_{\alpha,\beta}$, we can compute the three basis vectors as defined earlier. The resulting expressions are quite complex, so we omit them here; they can be easily computed with symbolic computation software such as Mathematica [12]. Instead, we provide visualizations of the bases for various curves in this family in Figure 3.5.

$\psi_{2,2}$ $\psi_{2,3/2}$ $\psi_{2,1}$ $\psi_{2,2/3}$ $\psi_{2,1/3}$ $\psi_{2,1/10}$

Remark (Animating motion). To define curve-based animation, we can use a method similar to the one introduced in Definition 14. The only difference is the addition of one extra dimension, which is handled by introducing an additional basis vector.

Example (Shortest path). Given our previous example, we naturally arrive at the question: which of these paths has the shortest arc length? The paths we're working with have the following arc lengths:

- $L(\psi_{2,2}) \approx 6.28$,
- $L(\psi_{2,3/2}) \approx 5.52$,
- $L(\psi_{2,1}) \approx 4.84$,
- $L(\psi_{2,2/3}) \approx 4.46$,
- $L(\psi_{2,1/3}) \approx 4.24$,
- $L(\psi_{2,1/10}) \approx 4.5$.

Observe that the path $\psi_{2,1/3}$ is the shortest. Nevertheless, it crosses a steep slope, making it more difficult for an agent to travel on than $\psi_{2,2/3}$, despite the latter being longer as we can see in Figure 3.5. Using solely arc length as a cost metric might suggest that $\psi_{2,1/10}$ is less costly than $\psi_{2,1}$, even though the agent encounters steep slopes.

This discrepancy underscores the need for more nuanced cost functions. Initially, we will formulate a heading cost function that penalizes travel uphill and another function that imposes a penalty for moving along steep contours. However, before developing these functions, we need to examine geometric properties of our surface paths by defining the heading and slope angles, which will serve as inputs for our cost functions.

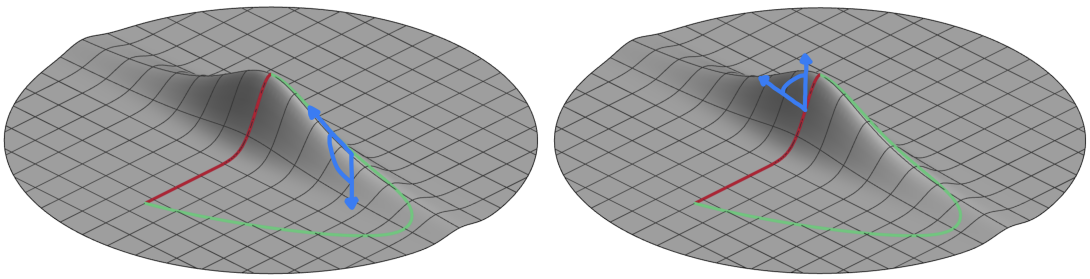
3.4 Geometric Properties of Surface Paths

This section delves into two essential geometric characteristics of our surface paths: the heading angle and the slope angle. The heading angle is visually represented in Figure 3.6a, while the slope angle is illustrated in Figure 3.6b.

Heading Angle

Definition 20 (Heading angle)**.** *Assume that $T : D \to \mathbb{R}$ is a terrain. Let $\varphi : I \to \mathcal{S}(T)$ be a parameterized curve that is regular. We then define its heading angle function $\varphi_H : I \to (0, \pi)$ as follows:*

$$\varphi_H(t) = \arccos\left(\frac{(0,0,-1)\cdot\varphi'(t)}{||\varphi'(t)||_2}\right).$$

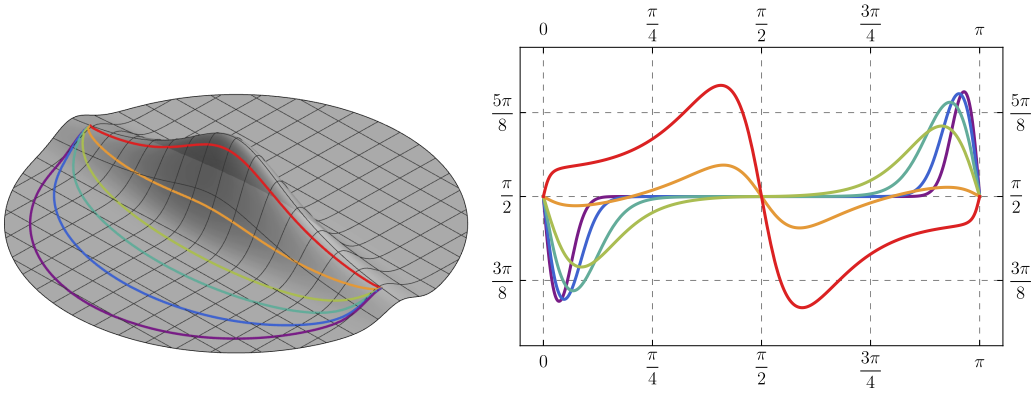


(a) Heading angle **(b)** Slope angle

Figure 3.6 Heading and slope angles

Remark (How does the definition work). The dot product in the numerator computes the cosine of the angle between the vectors $(0, 0, -1)$ and $\varphi'(t)$, multiplied by their magnitudes. Using this basic geometric relationship, we extract the angle with the arccos function. The vector $(0, 0, -1)$ represents the direction of gravity in our context.

Example (Deformed semi-circles heading angles). To understand this definition, let us apply it to our deformed semi-circles example. We can observe that when the agent moves in a direction aligned with the XY-plane, the angle is equal to $\pi/2$, which matches our intuition. When the agent travels uphill, the angle increases, and when it moves downhill, the angle decreases.



(a) $\psi_{2,2}$ $\psi_{2,3/2}$ $\psi_{2,1}$ $\psi_{2,2/3}$ $\psi_{2,1/3}$ $\psi_{2,1/10}$ **(b)** $\psi_{2,2,H}$ $\psi_{2,3/2,H}$ $\psi_{2,1,H}$ $\psi_{2,2/3,H}$ $\psi_{2,1/3,H}$ $\psi_{2,1/10,H}$

Figure 3.7 Heading angles of deformed semi-circles

Slope Angle

Definition 21 (Slope angle)**.** *Assume that $T : D \to \mathbb{R}$ is a terrain. Let $\varphi : I \to \mathcal{S}(T)$ be a parameterized curve. We define its slope angle function as $\varphi_S : I \to (0, \pi/2)$ given by:*

$$\varphi_S(t) := \arccos\left((0, 0, 1) \cdot N_T(\varphi(t)_1, \varphi(t)_2)\right).$$

Remark (Geometric interpretation). The vector $(0, 0, 1)$ represents the upward direction, while the normal vector N_T at a point on the surface indicates the local

orientation of the surface. Taking the dot product and computing the arccos gives us the slope angle: how steeply the surface is inclined relative to the vertical axis.

Additional Properties

As we mentioned earlier, curves possess more intriguing characteristics, such as curvature and, in a three-dimensional space, torsion, which simply means measuring how much the curve twists. Examining these properties more closely and potentially developing new cost functions based on them is a fascinating extension of our research. However, we will not explore this extension at this time.

3.5 Cost Functions

In Example 3.3, we examined the limitations of using arc length as a cost function, and in the previous chapter, we introduced the concepts of heading and slope angles for surface paths. In this section, we apply these definitions to construct generic cost functions designed to flexibly represent different agent behaviors. While the model is not strictly derived from physical principles, it is possible to incorporate physically based cost functions into this framework.

Heading Cost Function

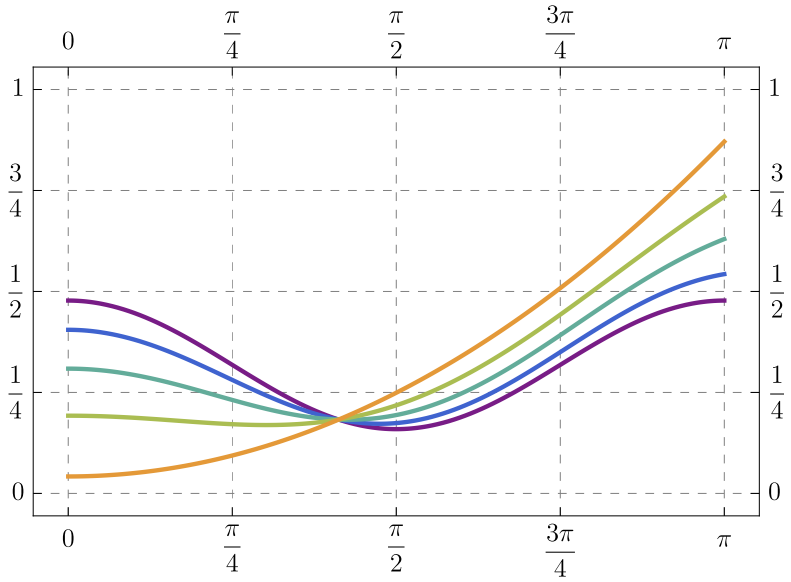
Definition 22 (Heading cost function)**.** *A heading cost is a function of the form* $C_H : (0, \pi) \to \mathbb{R}^+$*, which is bounded and smooth.*

Remark. We require the function to output positive values to avoid degenerate costs. The boundedness and smoothness conditions guarantee well-defined integrability and stable behavior.

Example (Simple heading cost)*.* One simple example of a heading cost function is the following parameterized function. Assume that $b \in [0, 1]$, then we define $C_{H,b} : [0, \pi] \to \mathbb{R}^+$ as follows:

$$C_{H,b}(x) := \frac{bx^2 + (1-b)\cos^2(x) + \frac{1}{2}}{\int_0^\pi (bx^2 + (1-b)\cos^2(x) + 1/2)\, dt} = \frac{bx^2 + (1-b)\cos^2(x) + \frac{1}{2}}{\frac{1}{6}\pi\,(2\pi^2 - 3)\,b + \pi}.$$

Notice that we have normalized the function so that its integral from 0 to π equals 1. This step is not strictly necessary, but it simplifies visualization and analysis.



(a) $C_{H,0}$ $C_{H,1/20}$ $C_{H,1/7}$ $C_{H,1/3}$ $C_{H,1}$

Figure 3.8 Various simple heading cost functions

Definition 23 (Heading cost of a path)**.** *Assume that $T : D \to \mathbb{R}$ is a terrain. Let $\varphi : (\alpha, \beta) \to \mathcal{S}(T)$ be a regular parameterized curve and let C_H be a heading cost function. We then define the heading cost of the path as:*

$$C_H(\varphi) = \int_\alpha^\beta C_H(\varphi_H(t)) \cdot ||\varphi'(t)||\, dt.$$

Example (Deformed semi-circles heading cost). For this example, sub-figures in the Figure 3.9 represents a specific path as defined in Example 3.3 along with a specific cost function as defined in Example 3.5. Each plot contains three distinct curves: blue represents the heading angle at each parameter value, red represents the cost function applied to the angle, and green represents the cost function weighted by the path's speed. Visually, the integral is represented by the area under the green curve, intuitively signifying the accumulated cost over time. The evaluated integrals for the given examples are as follows:

- $C_{H,0}(\psi_{2,1}) \approx 0.81$,
- $C_{H,1/3}(\psi_{2,1}) \approx 1.09$,
- $C_{H,0}(\psi_{2,1/10}) \approx 0.85$,
- $C_{H,1/3}(\psi_{2,1/10}) \approx 1.02$.

Interestingly, with the cost function $C_{H,0}$, the path preferred by a human-like agent is indeed cheaper, whereas for $C_{H,1/3}$, the steeper path is favored. This is a direct consequence of the cost function's design. One could, for example, make the penalization grow exponentially with increasing angles to force agents to avoid moving against gravity more strongly. We will explore other heading cost functions in the evaluation section, but for now we will move to the slope cost function.

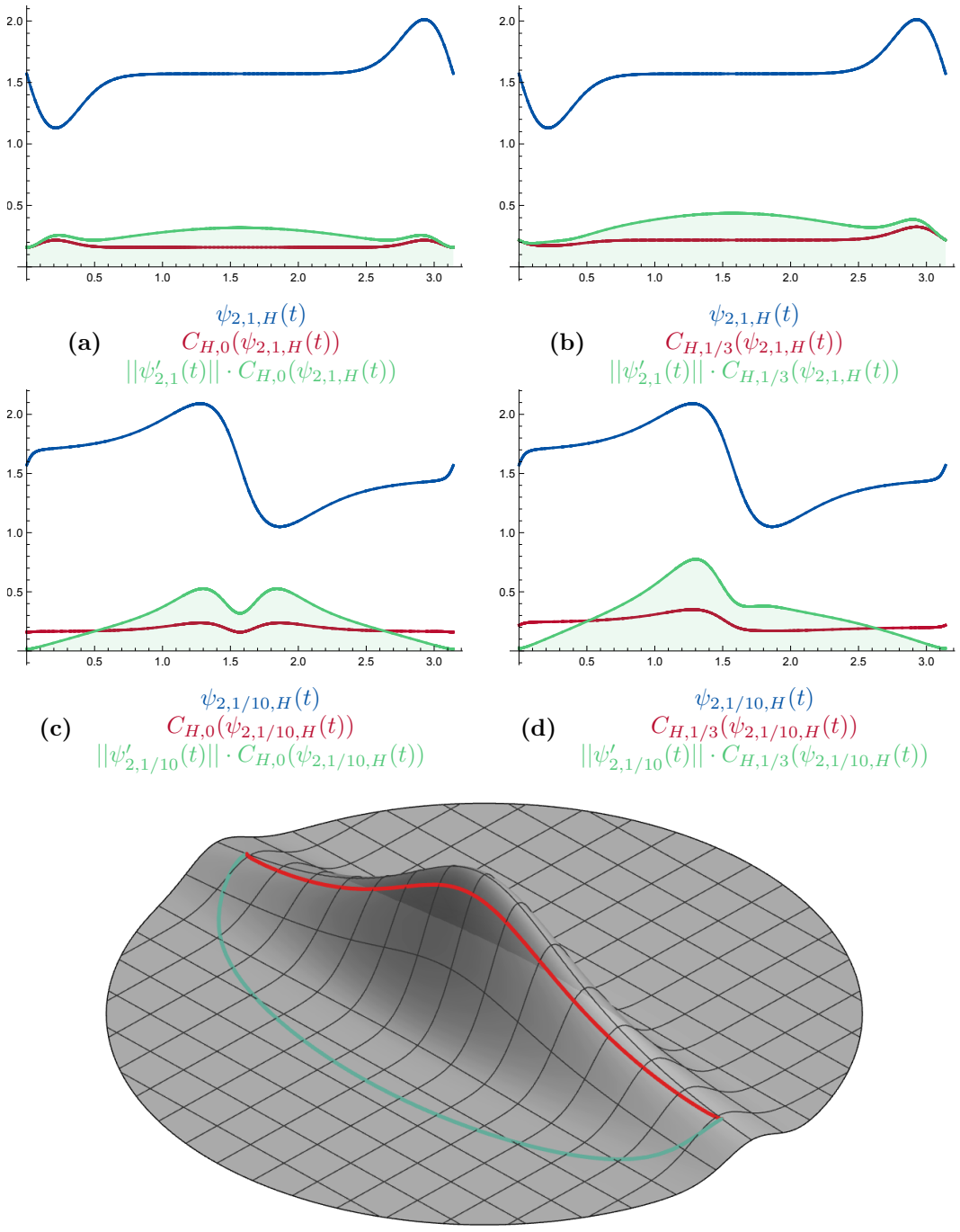


Figure 3.9 Heading costs of the paths

Slope Cost Function

One might argue that the heading cost function alone should be sufficient for evaluating the path. In the early design stages of this system, we shared the same perspective. However, specific scenarios reveal significant limitations. One particularly challenging case is described in the following example.

Example (Limitation of heading cost). Regarding the path $\psi_{2,1/3}$ depicted in Figure 3.10, from the point of view of the heading cost function, the agent travels

nearly parallel to the XY plane, resulting in minimal cost, given that we penalize agents who move uphill. However, if we examine the surface slope that the agent navigates, it is evident that this path should certainly not be inexpensive. Consider the extreme analogy of a rock climber moving sideways across a cliff face: though the heading direction may appear trivial, the slope they contend with is immense. Such discrepancies would not be captured without a cost that penalizes movement along steep slopes, highlighting the importance of incorporating a slope cost.

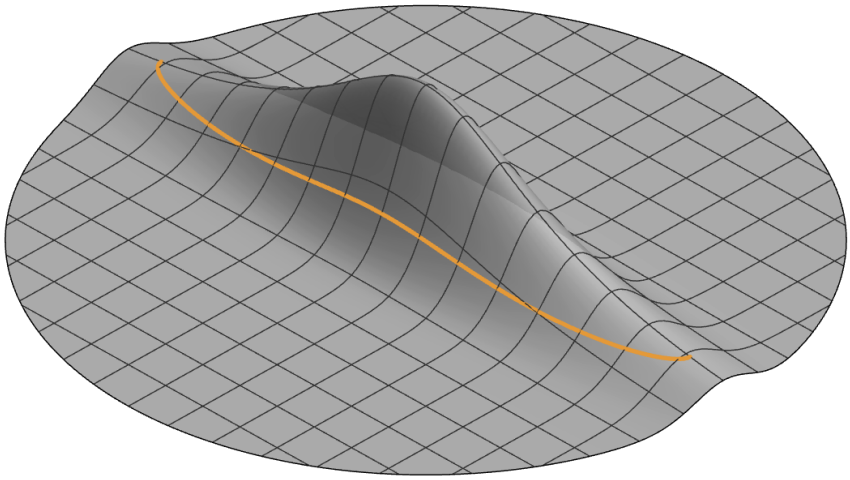


Figure 3.10 Example of a path for which heading cost does not suffice

Remark (Comparison with heading cost). The slope cost function operates under principles similar to those of the heading cost: we define a scalar cost that penalizes traversal based on the inclination of the surface along the path. Given that the geometric interpretation is largely analogous, we will not dissect it with the same granularity as we did for the heading cost. Instead, we will present the definitions and analyze their behavior in the evaluation chapter.

Definition 24 (Slope cost function)**.** *A slope cost function is a smooth and bounded map of the form* $C_S : (0, \pi/2) \to \mathbb{R}^+$.

Definition 25 (Slope cost of a path)**.** *Assume that* $T : D \to \mathbb{R}$ *is a terrain. Let* $\varphi : (\alpha, \beta) \to \mathcal{S}(T)$ *be a regular parameterized curve and let* C_S *be a slope cost function. The slope cost of the path is then defined as:*

$$C_S(\varphi) = \int_\alpha^\beta C_S(\varphi_S(t)) \cdot ||\varphi'(t)|| \, dt.$$

Joint Cost Function

In the previous subsections, we introduced two cost functions based on different geometric properties of terrain traversal: the heading cost and the slope cost. Using only one of these functions may not provide enough flexibility to capture realistic movement preferences. Optimal results can often be achieved when these two cost functions are combined. This motivates the definition of a joint cost function.

Definition 26 (Joint cost function)**.** *Assume that $T : D \to \mathbb{R}$ is a terrain. Let $\varphi : (\alpha, \beta) \to \mathcal{S}(T)$ be a regular parameterized curve, let C_S be a slope cost function, and let C_H be a heading cost function. We then define the joint cost of the path φ as follows:*

$$C(\varphi) := C_S(\varphi) + C_H(\varphi).$$

Remark (Practical extensions)*.* In practice, it may be helpful to extend this definition by introducing linear scaling with positive nonzero scalars α and β, resulting in the expression:

$$C(\varphi) = \alpha C_S(\varphi) + \beta C_H(\varphi).$$

This allows flexible weighting of the contributions of slope and heading to the overall cost. However, from a theoretical standpoint, this scaling is not strictly necessary, as the cost functions can be designed to encode this relationship directly.

Remark (Optimization perspective)*.* Given the heading and slope cost functions and two points on a terrain surface, we could formulate an optimization problem to find a smooth path on the surface that minimizes the joint cost function. This perspective is compelling for future research, especially in continuous optimization on smooth manifolds.

Remark (Scope and practical considerations)*.* Although the continuous optimization problem is mathematically elegant, solving it rigorously is beyond the scope of this thesis. Instead, we will address it discretely, using a heuristic-based optimization strategy. This simplified approach is sufficient for computer graphics applications. However, studying the optimization problem more rigorously—perhaps through the lens of discrete differential geometry could be a promising direction for future work.

Remark (Extensibility)*.* The cost function framework offers various possibilities for expansion. Using principles from differential geometry, costs stemming from torsion and curvature can be included to capture the effort needed to change direction, a key factor in applications like robotics. In addition, adding cost functions based on the position of the surface can, for example, help penalize terrains with deep snow or reduce costs associated with paved roads. Additionally, we can extend this framework beyond the confines of height maps, allowing it to be applied to more general manifolds.

3.6 Cost Based Reparametrization

To close this chapter, we will illustrate the usage of our joint cost function for reparametrizing curves so they mirror more realistic movement patterns. This forms the basis for our shift to the discrete model in the following chapter. We aim to reparametrize curves such that their speed corresponds to the terrain difficulty they traverse.

Definition 27 (Cost based reparametrization)**.** *Assume that $T : D \to \mathbb{R}$ is a terrain. Let $\varphi : (\alpha, \beta) \to \mathcal{S}(T)$ be a regular parameterized curve, let C_S be a*

slope cost function, and let C_H be a heading cost function. We then define map $s : (\alpha, \beta) \to \mathbb{R}^+$ as follows:

$$s(t) := \int_\alpha^t (C_S(\varphi_S(x)) + C_H(\varphi_H(x))) \cdot ||\varphi'(x)|| \, dx.$$

Due to the smoothness of our functions and the positivity of the cost functions s is a strictly growing function, therefore it has an inverse. Using the inverse, we define the curve ψ as:

$$\psi(t) := \varphi(s^{-1}(t)).$$

Function ψ is cost-based reparametrization.

Remark (Interpretation). The curve ψ acts as a cost-based reparameterization of φ, where the agent spends more time traversing high-cost regions and less time in low-cost regions. This results in a smooth, visually realistic motion that aligns with traversal difficulty.

Chapter Summary

With our continuous model now thoroughly developed, we are set to advance to the next stage, focusing on illustrating a specific application. This phase will involve using the model to simulate movement of crowds of agents in a manner that is suitable for computation.

4 Application to Crowd Simulation

This chapter will conceptually demonstrate how our terrain traversal model, developed in Chapter 3, can be used to simulate the movement of individual agents and their crowds. The solution is divided into three main stages.

1. **Discretizing the problem:** we transform the continuous traversal model into a discrete representation that is computationally traceable.
2. **Finding paths:** pathfinding algorithms are applied to this discrete model to discover viable traversal routes, utilizing the cost functions defined in the previous chapter.
3. **Generating animation curves:** the discrete paths are then refined into curves that realistically represent the movement of the agent on the surface.

At each stage, we will explore multiple algorithmic options, discussing their respective advantages and disadvantages. The aim is to provide flexibility in implementation while grounding the solution in the continuous cost-based traversal model introduced earlier.

It is important to emphasize that this method does not aim to solve the continuous optimization problem with precision guarantees. Instead, it provides a practical adaptation of the continuous model, showcasing its applicability in computationally feasible scenarios. In the following chapter, we will demonstrate a specific implementation based on the methods described here.

4.1 Problem Formulation

To simulate terrain traversal computationally, we will leverage portions of the model developed in the previous chapter to describe agent behavior and terrain properties. We will also introduce additional constraints that make the problem more suitable for computational methods. Furthermore, we will specify the agents' starting and target positions and clearly describe the expected output of the simulation.

Problem Specification

Definition 28 (Problem specification)**.** *The following components define the problem:*

- *A terrain map $T : D \to \mathbb{R}$, where:*
  - *D is a convex set,*
  - *D is a bounded set,*
  - *D is parametrized by $\rho : I_x \times I_y \to D$.*
- *A function $A_S : [n] \to \mathcal{S}(T)$ that specifies the starting positions of the agents.*

- *A function $A_T : [n] \to \mathcal{S}(T)$ that specifies the target positions of the agents.*

Remark (Stronger conditions). In crowd simulation scenarios, it is practical to define a bounding volume for each agent to prevent collisions. Although we do not explicitly incorporate this in the problem definition, we will mention which methods can address collision avoidance. Additionally, it may be beneficial to enforce pairwise distinctness and a minimum separation distance for both starting and target positions. This makes it easier to avoid initial collisions and ensures non-degenerate agent placement. These enhancements are straightforward to implement, and we will indicate where specific algorithms can handle these constraints.

Solution Requirements

Definition 29 (Solution Requirements)**.** *Given the problem specification, our goal is to find continuous maps $\varphi_1 : [\alpha_1, \beta_1] \to \mathcal{S}(T), ..., \varphi_n : [\alpha_n, \beta_n] \to \mathcal{S}(T)$ such that for every $i \in [n]$:*

- $\varphi_i(\alpha_i) = A_S(i)$,
- $\varphi_i(\beta_i) = A_T(i)$.

Remark (Smoothness of the solution). Ideally, we would like the continuous maps to be smooth in the interior of their domains. However, achieving full smoothness (C^∞) is often impractical in discrete simulation settings. Instead, we will generate paths that are at least C^1 (continuously differentiable) and preferably smoother where feasible. The focus will be on achieving practical smoothness for animation purposes, rather than strict mathematical guarantees.

4.2 Discrete Terrain Representation

To simulate terrain traversal computationally, we need a discrete representation of the surface that allows efficient computation and path finding. We achieve this by using a graph-based model, enriched with geometric properties captured through vertex and edge functions.

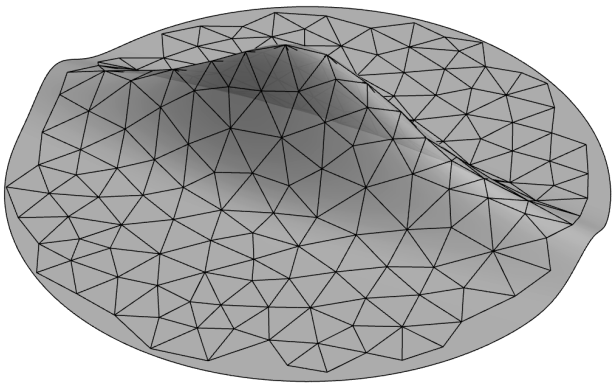


Figure 4.1 Illustration of terrain discretization

Definition 30 (Discrete Terrain)**.** *Assume that we are given a problem description, then we define a discrete terrain representation as the triplet* (G, P, C) *where:*

- $G = (V, E)$ *is a directed graph,*
- $P : V \to \mathcal{S}(T)$ *is a function that maps vertices to points on the surface,*
- $C : E \to \mathbb{R}^+$ *is an edge cost function.*

We additionally require that for every $i \in [n]$*, there exist vertices* $v, w \in V$ *such that* $A_S(i) = P(v)$ *and* $A_T(i) = P(w)$*. Here,* n *represents the number of agents as defined in the problem specification.*

Remark (About the Definition). This definition captures multiple important aspects of our problem. The function P effectively represents an embedding of the graph G on the surface of our terrain. Moreover, it guarantees that each starting and ending point of our agents is represented by at least one vertex. Optionally, we could enforce a minimum separation distance between these vertices to avoid overlaps. Finally, this representation provides everything we need to perform shortest path searches with custom edge costs, which are represented as non-negative real numbers.

The next step is to construct this discrete representation based on our problem specification. This process will be broken into three parts:

- **Section 4.3:** vertex generation and the construction of the surface mapping function P.
- **Section 4.4:** Edge creation strategies that are consistent with our terrain model.
- **Section 4.5:** Calculation of edge costs, leveraging the traversal cost model developed in Chapter 3.

After establishing these components, we will be ready to apply search algorithms to find optimal paths over the terrain in Section 4.6.

4.3 Vertex Generation

This section explores various methods for generating vertices on a terrain surface, ranging from simple domain sampling to more sophisticated spatially aware strategies. Our goal is not to prescribe a single method, but rather to present a spectrum of approaches, each with strengths and limitations.

Remark (Sampling and open domains). In Chapter 3, we rigorously discuss the importance of open domains for smoothness and well-defined derivatives. This chapter maintains that convention, even though practical implementations often require sampling near boundaries.

Sampling algorithms described in this section are defined for parametrizations with closed domains. To avoid issues near the boundaries, we introduce a small constant ε, representing a small positive real number. In practice, when we sample over an open interval (a, b), we sample instead over a slightly contracted interval $[a+\varepsilon, b-\varepsilon]$. To simplify notation, we omit the explicit mention of ε in algorithms, assuming the reader can select an appropriate value according to context.

Example (Open Disk Domain). This section will use the open-disk domain to demonstrate the upcoming methods. This choice is due to its simplicity in parameterization while simultaneously highlighting challenges in uniform sampling. We define parametrization as $\rho_C : [0, R) \times [0, 2\pi] \to \mathbb{R}^2$, where $R \in \mathbb{R}^+$, as follows:

$$\rho_C(r, \alpha) := r(\cos(\alpha), \sin(\alpha)).$$

This parametrization extends to the entire plane $\mathbb{R}^2$ if we set $R = +\infty$, effectively describing two-dimensional Euclidean space in polar coordinates.

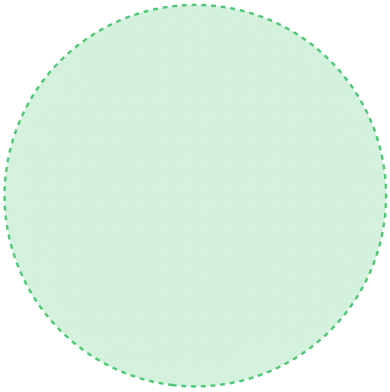


Figure 4.2 Open disk domain

Remark (Domains we are dealing with). Keep in mind that utilizing parametrization to generate vertices on a surface involves operating in two distinct domains. The first is the domain of the parametrization, which varies based on the specific parametrization used. The second is the domain of the terrain itself. When sampling points, we select them from the parametrization domain, and their images reside in the terrain domain, after which we project these points onto our surface. This is important, because uniform sampling within the parameter domain does not always lead to uniform sampling within the terrain domain, as we will demonstrate soon.

Uniform Parametrization Domain Sampling

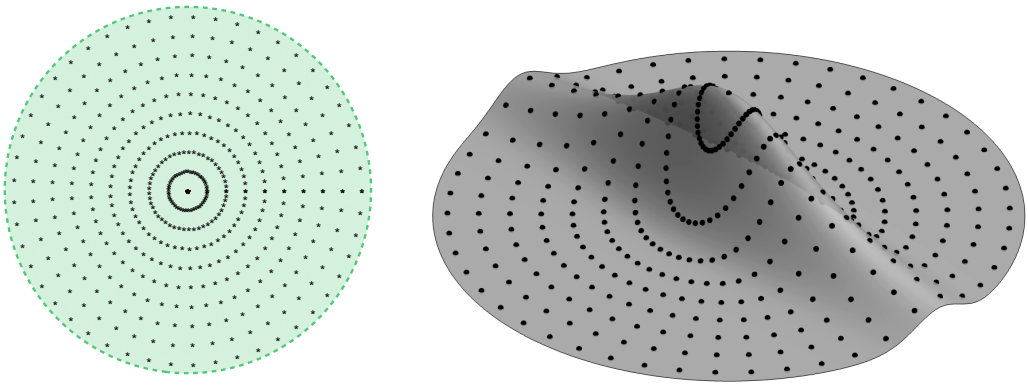
Algorithm 1 Uniform parametrization domain sampling

Require:

$n, m \in \mathbb{N} \land n, m > 1$ ▷ Numbers of samples

$\rho : [u_{\min}, u_{\max}] \times [v_{\min}, v_{\max}] \to D$ ▷ Domain parametrization

for $i = 0$ to $n - 1$ **do**
  for $j = 0$ to $m - 1$ **do**
    $u \leftarrow u_{\min} + (u_{\max} - u_{\min})\frac{i}{n-1}$
    $v \leftarrow v_{\min} + (v_{\max} - v_{\min})\frac{j}{m-1}$
    $s \leftarrow \rho(u, v)$
    $S \leftarrow S \cup \{(s_1, s_2, T(s_1, s_2))\}$ ▷ Project onto terrain surface.
  end for
end for
return S



(a) Terrain Domain **(b)** Terrain Surface

Figure 4.3 Uniform sampling illustration

Example (Uniform parametrization domain sampling of a disk). Using this algorithm with our hill terrain from the previous chapter, we generate a set of sample points that are uniformly spaced in the parametrization domain. However, their images in the terrain domain are not sampled uniformly. This is due to the nature of the parametrization, where equal steps in the parameter space correspond to shrinking distances near the center. Moreover, when we project those points onto our terrain surface, the problem is amplified.

This issue can be partially mitigated through reparametrization, but that is not always feasible. More critically, the regularity of the grid introduces visible directional artifacts that are problematic for terrain traversal and agent simulation. The following algorithm addresses this by introducing randomness into the sampling process to break up the artificial structure.

Randomized Domain Sampling

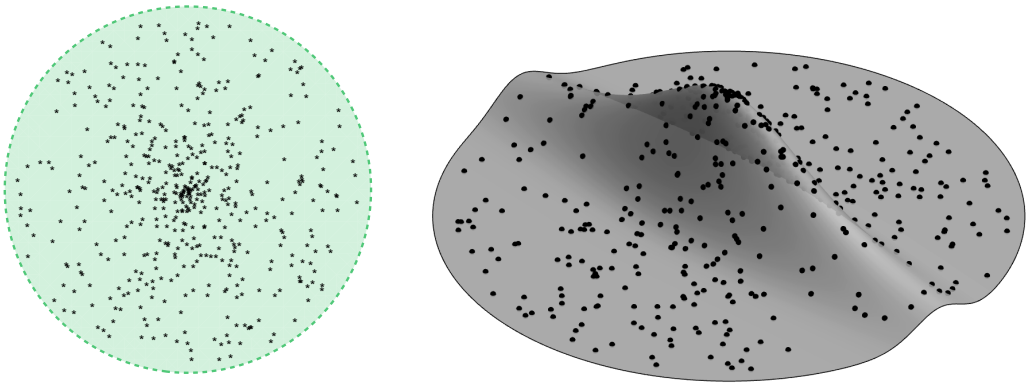
Definition 31 (Random Number Generator)**.** *For the purposes of randomized algorithms, we define a function* $\text{UniformSample} : \mathbb{R} \times \mathbb{R} \times \mathbb{N} \to \mathbb{R}$ *that, for input parameters* $\text{UniformSample}(\min, \max, \text{seed})$*, generates a uniformly distributed random value within the interval* $[\min, \max]$*. The parameter* seed *represents the index of the random trial, allowing a deterministic generation of values if the same seed is reused.*

We do not delve into the details of probability theory, as they are beyond the scope of this thesis. In practical applications, this function would be implemented using pseudo-random number generators.

Algorithm 2 Randomized uniform domain sampling

Require: $n \in \mathbb{N}, \rho : [u_{\min}, u_{\max}] \times [v_{\min}, v_{\max}] \to D$

$S \leftarrow \text{Im}(A_S) \cup \text{Im}(A_T)$ ▷ Include all start and target points.
for $i = 0$ to $n - 1$ **do**
    $u \leftarrow \text{UniformSample}(u_{\min}, u_{\max}, 2i)$
    $v \leftarrow \text{UniformSample}(v_{\min}, v_{\max}, 2i + 1)$
    $s \leftarrow \rho(u, v)$
    $S \leftarrow S \cup \{(s_1, s_2, T(s_1, s_2))\}$ ▷ Project onto terrain surface.
end for
return S



(a) Terrain Domain **(b)** Terrain Surface

Figure 4.4 Random sampling illustration

Example (Randomized uniform domain sampling of a disk)*.* Unlike the previous method, this sampling strategy eliminates directional artifacts by randomizing parameter selection. However, the issue of denser sampling near the disk center persists. Furthermore, random sampling tends to produce clusters of points, which can result in regions of high local density. Although this improves grid artifacts, it still falls short of ideal vertex distribution. The following algorithm addresses these limitations by enforcing a minimum separation distance between samples.

Poisson Disk Sampling

Algorithm 3 Poisson Sphere Sampling

Require:

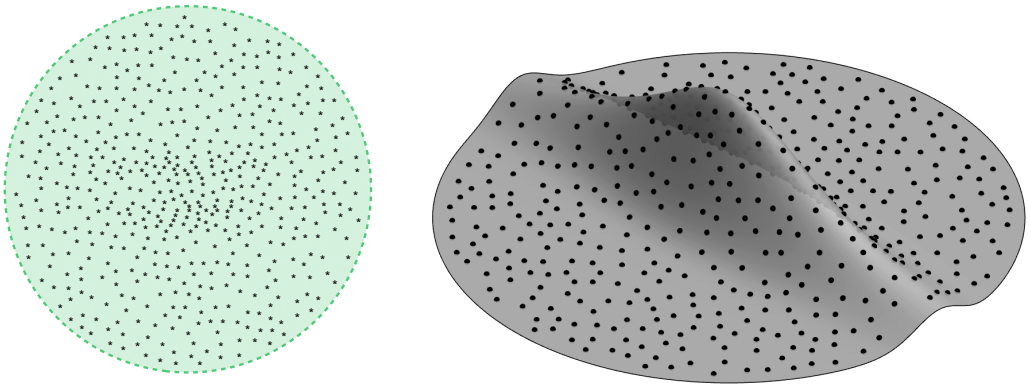
$n \in \mathbb{N}$ ▷ Number of trials

$m \in \mathbb{N}, m \geq |\operatorname{Im}(A_S) \cup \operatorname{Im}(A_T)|$ ▷ Desired number of samples

$r \in \mathbb{R}^+$ ▷ Separation radius

$\rho : [u_{\min}, u_{\max}] \times [v_{\min}, v_{\max}] \to D$ ▷ Domain parametrization

$S \leftarrow \operatorname{Im}(A_S) \cup \operatorname{Im}(A_T)$ ▷ Include all start and target points.
for $i = 0$ to $n - 1$ **do**
  if $|S| \geq m$ **then**
    break
  end if
  $u \leftarrow \text{UniformSample}(u_{\min}, u_{\max}, 2i)$
  $v \leftarrow \text{UniformSample}(v_{\min}, v_{\max}, 2i + 1)$
  $s \leftarrow \rho(u, v)$
  $s' \leftarrow (s_1, s_2, T(s_1, s_2))$ ▷ Project onto terrain surface.
  if $\min_{p \in S} d(p, s') > r \vee S = \emptyset$ **then** ▷ Discard points.
    $S \leftarrow S \cup \{s'\}$
  end if
end for
return S



(a) Terrain Domain **(b)** Terrain Surface

Figure 4.5 Poisson sampling illustration

Example (Poisson Sphere Sampling of Terrain with Disk Domain). Suppose that we apply this algorithm to our hill terrain and the disk domain. In that case, we obtain a well-distributed set of samples that are guaranteed to be separated by at least a distance r in a three-dimensional space. Unlike the previous methods, this sampling approach completely avoids clustering.

Remark (Implementation Note). The algorithm presented here adheres to the core principles of Poisson disk which were popularized by Cook [14]. Specifically, ensuring that at least a minimum distance of r separates any two samples. We

call it Poisson sphere sampling because the sampling and distance checks occur directly in three-dimensional space, accounting for the height-map elevation. The presented algorithm is naive, as it samples candidate points uniformly in the parameter domain and then filters them based on the separation constraint. Also note that the minimum distance query can slow down this algorithm significantly, naive implementation would perform this operation in linear time complexity, whereas with space partitioning data structures, we get much closer to logarithmic time complexity for this query. More sophisticated methods, such as the Bridson algorithm [15], optimize this process for far greater performance than the naive method.

Alternative Methods

Remark (Alternative Methods). There are numerous alternative methods for generating vertex samples, each with its own trade-offs:

- **Relaxation methods:** these techniques begin with an arbitrary set of samples, and their positions are iteratively adjusted to minimize local clustering and maximize even spacing. Popular algorithms like Lloyd's algorithm [16] optimize the distribution by relocating points to the centroids of their Voronoi cells. Method that is particularly suitable for our problem is the capacity-constrained variant of Lloyd's method [17]. These methods produce high-quality distributions, although naive implementations might introduce directional artifacts.
- **Remeshing algorithms:** although remeshing algorithms are not specifically developed to address our problem, they can be used for this purpose. Numerous remeshing algorithms exist and [18] offers an excellent literature review to begin exploring them. The main concept involves starting with an initially uniformly sampled mesh and employing remeshing techniques to achieve a mesh optimized for uniform edge lengths.

In practice, many 3D graphics tools offer efficient implementations of these techniques. For our implementation, we will utilize a built-in operator in SideFX Houdini, a widely used software for procedural modeling and simulation. Houdini's relaxation-based point sampling is highly optimized, providing distribution quality comparable to Poisson sphere sampling. This allows us to focus on the simulation logic rather than low-level point sampling, without sacrificing fidelity in vertex distribution.

With vertex generation completed, we are now ready to move on to the next section, where we explore edge construction and how these vertices are connected to form traversable paths on the terrain surface.

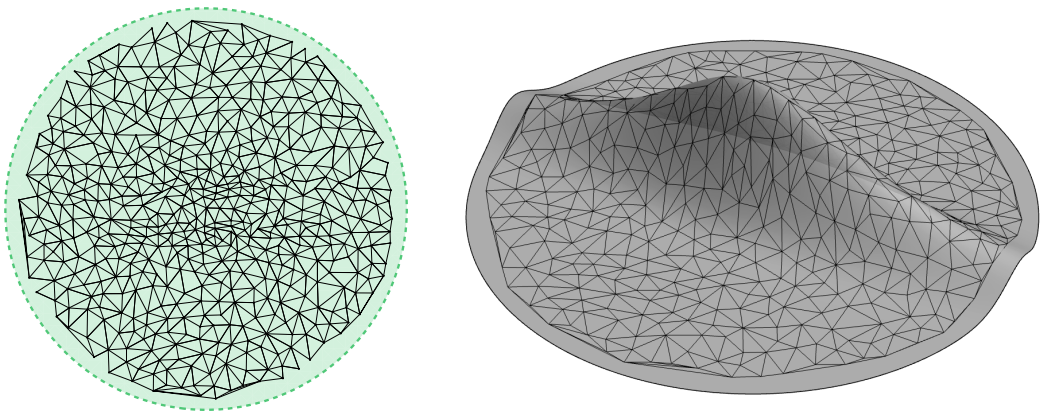
4.4 Edge Generation

Edge generation from vertices is a crucial step in constructing the graph representation of the terrain. Just as with vertex generation, there are multiple approaches to this. However, Delaunay triangulation is the most commonly used method for creating edges or meshes from sets of points. This section explains its suitability for our discrete terrain representation.

Delunay Triangulation

Remark (What is Delaunay triangulation). Delaunay triangulation is a method of connecting a set of points on a plane into triangles that do not overlap, ensuring that the circumcircle of any triangle in the network is empty of other points. This characteristic optimizes the minimum angle of the triangles, preventing the formation of excessively narrow triangles and leading to a more uniform mesh. A formal definition or algorithmic description is not provided here as it falls beyond the scope of this thesis. For comprehensive coverage, standard texts in computational geometry can be consulted [19].

Example (Delaunay triangulation of a hill). In this example, we employ the Poisson Sphere detailed in the previous section, and the sampling method employed generates the initial set of vertices. We then apply Delaunay triangulation to produce the edges. It is important to note that Delaunay triangulation is inherently undirected; however, for our graph representation, we require directed edges. The solution is straightforward: we create two directed edges for each undirected edge in the triangulation, one for each direction.



(a) Terrain Domain **(b)** Terrain Surface

Figure 4.6 Deaunay triangulation

Remark (Computational efficiency). One of the main reasons for the popularity of Delaunay triangulation is its computational efficiency. In two dimensions, Delaunay triangulation can be computed in $O(n \log n)$ time complexity, where n is the number of vertices. This makes it efficient enough for practical applications in crowd simulation.

4.5 Cost Computation

The final stage of our discretization process is to compute the costs associated with our edges. Thanks to our earlier efforts, this task is relatively simple; it simply requires us to specify the curve delineated by a particular edge and then assess the cost function.

Edge Interpretation

Definition 32 (Edge curves)**.** *Given a set of vertices V, and a set of edges E generated using the methods mentioned in the previous sections, we define the edge curve for every edge $(v, w) \in E$, we define the curve $\varphi_{(v,w)} : [0, 1] \to \mathcal{S}(T)$. For simplicity, we introduce the auxiliary notation:*

$$\psi_{(v,w)}(t) := (1 - t) \cdot P(v) + t \cdot P(w),$$

and define the edge curve as:

$$\varphi_{(v,w)}(t) := \left(\psi_{(v,w)}(t)_1, \psi_{(v,w)}(t)_2, T(\psi_{(v,w)}(t))\right).$$

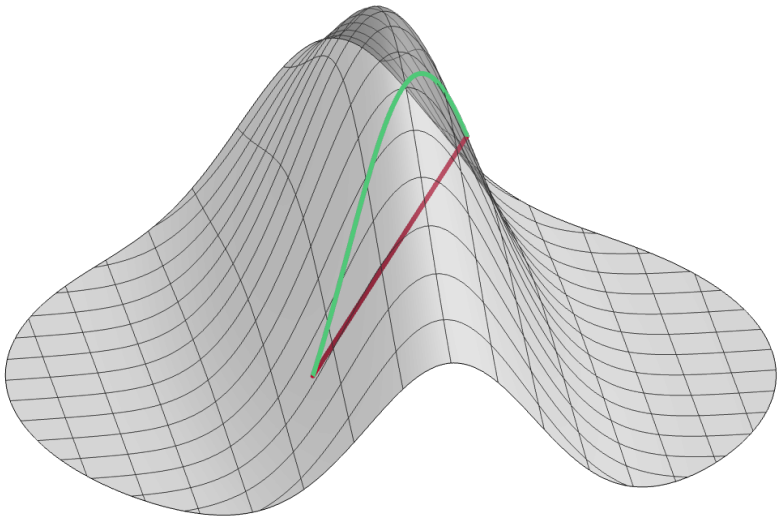


Figure 4.7 Edge curve

Remark (Geometric interpretation). From a geometric point of view, this is equivalent to projecting the line segment between $P(v)$ and $P(w)$ onto the surface of the terrain. This is analogous to the path projection executed in the proof of Theorem 4, although adapted for line segments. The derived path may not be the shortest or most cost-effective, yet it remains smooth and well defined. The reason why it is well defined is that we have explicitly requested that the domain be a convex set.

Cost Computation

Definition 33 (Edge Cost)**.** *Assume that we have edges E, slope cost function C_S and heading cost function C_H. Let C be their joint cost function. For each edge $(v, w) \in E$, we define the cost as:*

$$C(v, w) := C(\varphi_{(v,w)}).$$

Remark (Interpretation). The cost of an edge is equivalent to the traversal cost of its corresponding edge curve. Since edge curves are smooth and parameterized by $t \in [0, 1]$, the cost is computed as the integral of the joint cost function along this curve. This follows directly from the cost definitions established in Chapter 3.

Remark (Practical computation). Now that we have established methods for generating vertices, constructing edges, and assigning costs to those edges, we can finally combine all these components. The following section will explore how to find paths on our discrete terrain.

4.6 Path Finding

By modeling the terrain as a directed graph with a positive cost function, we are ready to utilize the pathfinding algorithms. This section offers a concise overview of common pathfinding methods that fit our model, emphasizing their advantages and drawbacks. For clarity, the discussion will distinguish between single-agent and multi-agent pathfinding.

Definition 34 (Agent paths)**.** *Given the problem specification, where we have function specifying starting points $A_S : [n] \to \mathcal{S}(T)$ and a function $A_T : [n] \to \mathcal{S}(T)$ specifying target points, for every agent $i \in [n]$ we want to find a sequence of vertices $(v^i_j)_{i=0}^{n_i}$ such that they form a path within the graph and $P(v^i_0) = A_S(i)$ and $P(v^i_{n_i}) = A_T(i)$. Fortunately, given the work done in the previous sections, we can directly employ a known pathfinding algorithm to solve the problem.*

Single Agent Path Finding

Single-agent pathfinding is the simplest case where we only consider one agent at a time. Such algorithms are extensively studied in algorithm textbooks like [20] and even artificial intelligence textbooks like [21].

Example (Dijkstra's algorithm). One of the most fundamental pathfinding algorithms is Dijkstra's Algorithm, which finds the shortest path from a starting vertex to every other vertex in a weighted graph. It operates in $\mathcal{O}(|E| + |V| \log |V|)$ time complexity using a priority queue. While robust and optimal for graphs with non-negative edge costs, it does not leverage spatial information, leaving room for optimization in structured domains like ours.

Example (A* search). A* extends Dijkstra's algorithm by using a heuristic to prioritize nodes that are likely closer to the goal. Despite sharing the same worst-case time complexity as Dijkstra's, A* typically examines fewer nodes. The core of this approach is the heuristic function $h(v)$, which predicts the cost of achieving the goal from node v. To ensure optimality with A*, the heuristic must satisfy specific conditions, as discussed in textbook [21]. Developing such a heuristic for our problem could be challenging.

Remark (Limitations of single-agent pathfinding). Using single-agent pathfinding for each agent independently works well if the number of agents is small relative to the terrain size, as collisions are unlikely. However, it cannot guarantee collision-free paths in dense scenarios. Simple collision avoidance strategies or local path adjustments can mitigate this issue. However, more complex scenarios call for multi-agent pathfinding algorithms.

Multi Agent Path Finding

Multi-agent pathfinding (MAPF) aims to find non-colliding paths for multiple agents moving simultaneously on the graph. In its simplest form, it is defined on graphs with unit edge costs and discrete time steps, ensuring that no two agents occupy the same vertex simultaneously. There are various extensions, for instance, for cost-based graphs, agents that might occupy multiple vertices simultaneously, or extensions that deal with edge-based collisions. Standard terminology for MAPF, and various extensions of the problem are detailed in the following publication [22].

Remark (Application to Terrain Traversal). In real-world situations, agents rarely adhere to meticulously synchronized, pre-determined paths. Excessive coordination can lead to unrealistic behavior in visual simulations. Consequently, even though MAPF ensures paths with no collisions, it might lead to movement patterns that appear too artificial. This becomes particularly challenging for crowd simulations in computer graphics, where maintaining a natural flow is more crucial than ensuring absolute collision avoidance.

Remark (Implementation Decision). To achieve natural movement and maintain computational efficiency in our implementation, we will employ single-agent pathfinding strategies. When collisions occur, they can be locally managed by adjusting the timing or spatial positioning of the intersecting paths, though a detailed explanation will not be provided here. Expanding this approach to encompass the complete MAPF presents an intriguing avenue for future research; however, it falls beyond the scope of this thesis.

4.7 Path Smoothing

In earlier sections, we outlined a method for calculating discrete paths across our graph-based model of the terrain. Constructing the surface path as a projection of lines connecting consecutive vertices onto the surface through the z-axis, the path would result in a continuous path; however, it wouldn't maintain continuous differentiability at the vertices. To address this, we apply a straightforward interpolation method that ensures C^1 smoothness. This section will emphasize a practical, high-level approach instead of delving into detailed derivations.

Catmull-Rom Spline

To achieve smooth interpolation, we utilize the Catmull-Rom spline. The complete mathematical description can be found in [23], but we will provide a high-level overview. Catmull-Rom spline is constructed from a sequence of vertices and it is a curve that passes through all of them in the specified order. The interpolating curve can be generalized to arbitrary number of dimensions and it provides C^1 smoothness.

To represent both spatial and cost information simultaneously, we extend the dimensionality from $\mathbb{R}^3$ to $\mathbb{R}^4$, where the fourth dimension tracks the *cumulative cost* of the path. This allows us to interpolate both the spatial path and its associated traversal cost.

Constructing Interpolating Path

Definition 35 (Cumulative path cost)**.** *Given a path in our discrete terrain* $P = (v_i)_{i=0}^n$*, we define the cumulative cost* $\mathcal{C}$ *as follows:*

$$\mathcal{C}(v_0) := 0, \quad \mathcal{C}(v_i) := \mathcal{C}(v_{i-1}) + C(v_{i-1}, v_i) \quad \textit{for } i \in [n].$$

Definition 36 (Interpolating path)**.** *Given the path* $P = (v_i)_{i=0}^n$*, we construct a sequence of points in* $\mathbb{R}^4$*:*

$$P^* = (P(v_i)_1, P(v_i)_2, P(v_i)_3, \mathcal{C}(v_i))_{i=0}^n \,.$$

This seqeunce of points represents input we will use to create Catmull-Rom interpolating spline. This will give us C^1 *curve* $\varphi : [0, \alpha] \to \mathbb{R}^4$*. Here, the fourth coordinate represents the accumulated cost.*

Cost-Based Reparametrization

Although the interpolating path φ is C^1, the parameterization may not be optimal, as the cumulative cost could rise nonlinearly. This issue can be easily resolved by employing the fourth component of our function curve φ and using its inverse to formulate the curve $\psi(t) := \varphi(\varphi_4^{-1}(t))$. The fourth component is intentionally strictly increasing, allowing for the inverse to exist. This approach will be utilized in the practical implementation, and it will ensure section, ensuring that the agent's speed at any point closely aligns with the challenge of navigating that terrain.

Chapter Summary

This chapter outlined a structured approach to simulate crowds on uneven terrain, which includes discretizing smooth paths and adapting them for agent movement. By creating vertices and edges and employing specific cost functions, we converted the theoretical framework into a format suitable for computational processing. In the following chapter, we will assess the outcomes possible with this system.

5 Evaluating Crowd Simulation

This chapter presents an experimental evaluation of our crowd simulation framework, focusing on key aspects that demonstrate its efficiency and scalability. The primary objective is to understand the performance characteristics and identify potential constraints under varying conditions.

Cost Function Comparison

The first part visually evaluates the impact of different cost functions on single-agent path traversal, including the joint cost function, slope cost, heading cost, and distance-based cost. This analysis highlights the influence of each cost strategy on path selection.

Agent Behavior Analysis

The second part examines agent behavior in relation to local traversal costs, speed adaptation, and the expected difficulty of traversal at a given point. We analyze the slopes agents traverse, their heading directions, and explore how well they avoid high-cost regions.

Collision Analysis

The final experiment analyzes path behavior in relation to collision events. This section explores how agent density leads to congestion and inefficiencies in pathfinding.

5.1 Implementation

SideFX Houdini

The implementation of our crowd simulation system, as outlined in the previous chapter, was carried out using SideFX Houdini[24], a leading software in procedural generation and simulation. Houdini's node-based architecture enables a fully procedural and nondestructive workflow, allowing each operation, whether geometric transformation, simulation, or shading, to be encapsulated as a node within a graph network (see Figure 5.1). This design facilitates efficient parameter adjustments, real-time visualization, and modular development, making it an ideal platform for implementing complex simulations. Furthermore, Houdini's proprietary scripting language, VEX, provides powerful programmatic access to its internal functionalities, enabling custom logic and advanced pathfinding behaviors directly within its node graph architecture. This integration of high-level node manipulation with low-level scripting capabilities significantly accelerated development and improved control over simulation parameters.

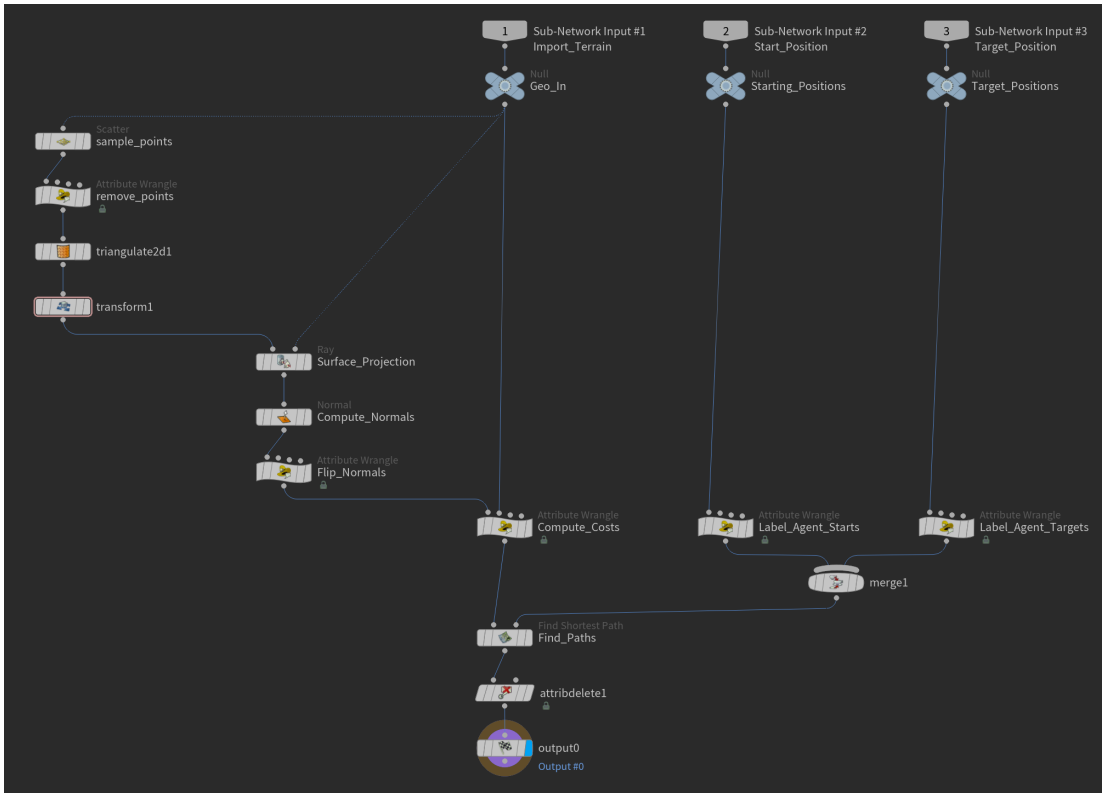


Figure 5.1 Houdini network graph

Justifying Choice

We chose Houdini over more traditional approaches such as C++ implementations with geometry libraries such as CGAL or game engines such as Unity for several reasons. Houdini's vast library of geometric algorithms and node-based design proved to be optimal for rapid prototyping and real-time feedback during simulation adjustments. Its modular nature allowed us to seamlessly integrate procedural terrain generation, agent navigation, and collision detection within a unified environment, avoiding the need for extensive low-level geometry processing. Additionally, Houdini's visual programming model supports high-resolution animation rendering, making it well suited for generating visualizations of crowd movement animations presented in this work.

Reproducing Results

To allow full reproducibility and exploration, the complete Houdini simulation file is included as an attachment to this thesis. It contains all the procedural setups, the pathfinding setups, and the animation setup used throughout the evaluation. Assuming working knowledge of the software the file is fully self-documented; however, due to the professional-grade nature of Houdini's interface, some navigation may require familiarity with node-based workflows. We encourage readers who wish to explore the implementation to use Houdini Apprentice, a free non-commercial version of the software that fully supports our project setup. Installation details and software documentation are available at [24].

5.2 Experiment Design

This section outlines the design and configuration of our experimental setup. For all experiments, data collection, terrain generation, and animation rendering were conducted using SideFX Houdini [24], while evaluation and visualization were performed using Wolfram Mathematica [12]. This combination is particularly powerful: Houdini is optimized for efficient geometric operations and large-scale procedural generation, while Mathematica excels at data analysis and seamless visualization.

Terrain

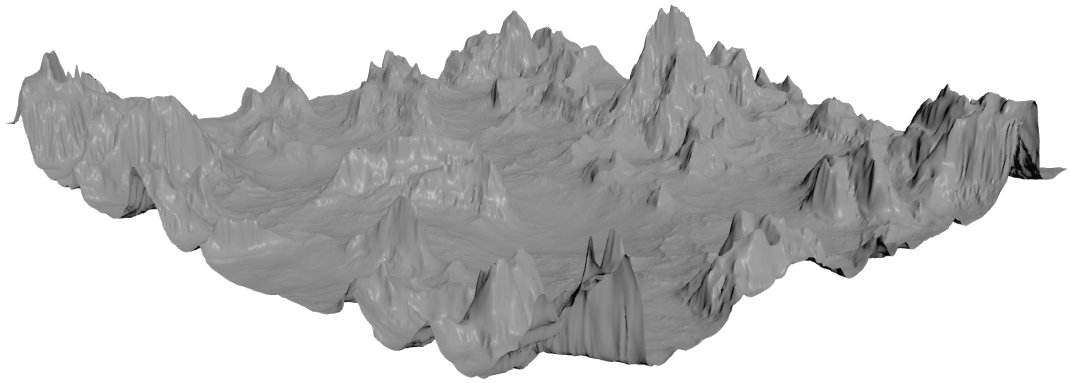


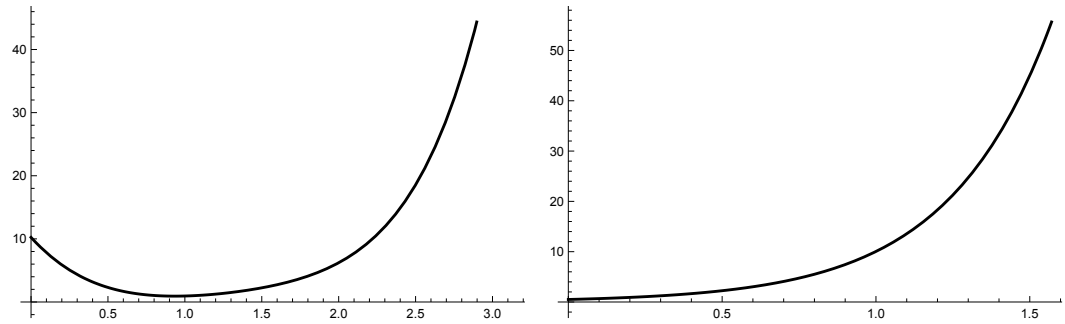
Figure 5.2 Procedural terrain

The experiments were carried out on procedurally generated height-field terrain designed to emulate realistic mountainous landscapes. We opted for procedural generation instead of real-world height maps for two main reasons.

- Arbitrary levels of detail: procedurally generated terrain allows for fine-grained geometric control and highly accurate surface normal calculations, which are often limited in real-world measured data.
- Consistency with intended application: the system is intended for applications in computer graphics and simulation, where procedurally generated environments are standard practice.

Cost Functions

Two experimental cost functions were designed to capture pathfinding behavior that mimics realistic human traversal patterns.



(a) Heading cost function **(b)** Slope cost function

Figure 5.3 Experimental cost functions

Definition 37 (Experimental Heading Cost Function)**.**

$$C_H^E(x) := 2\left(x - \frac{3}{2}\right)^4 + \frac{1}{2}e^{2x-\frac{3}{2}}.$$

Definition 38 (Experimental Slope Cost Function)**.**

$$C_S^E(x) := \frac{e^{3x}}{2}.$$

Single-Agent Experiment

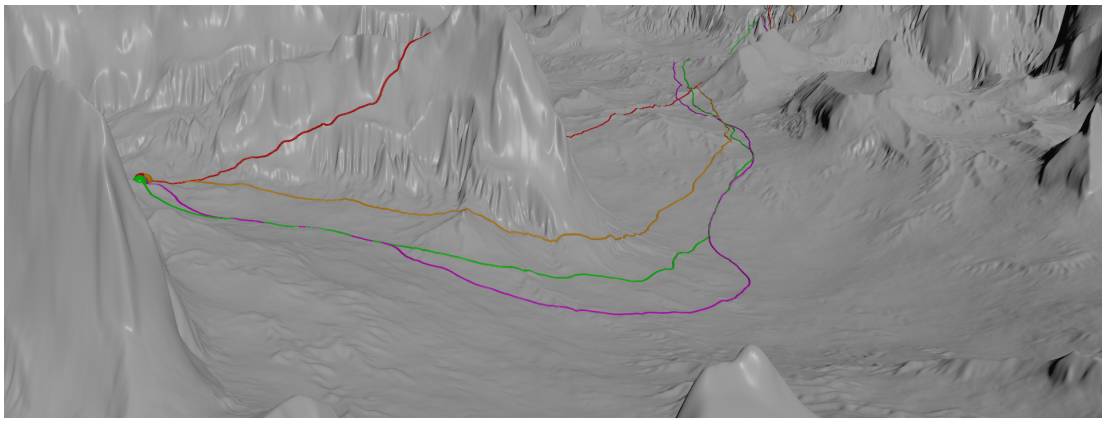


Figure 5.4 Single agent experiment

In the single-agent scenario, starting and target locations were manually selected to evaluate path construction across various cost functions. We examined four distinct variants.

- Joint cost function: demonstrates system behavior when both cost functions are simultaneously applied, promoting optimal pathfinding.
- Heading cost function: evaluates path behavior when penalization is applied solely based on heading alignment.
- Slope cost function: analyzes navigation when only slope-based cost calculations are considered.

- Distance-based cost function: serves as a baseline, illustrating the limitations of naive pathfinding strategies that prioritize only the shortest distance.

The joint cost function demonstrates how combining heading and slope considerations produces more realistic paths. In contrast, individual cost functions reveal that optimizing for only one aspect (heading or slope) is insufficient for optimal traversal. The purely distance-based approach, while direct, fails to account for terrain complexity, leading to suboptimal paths.

Remark (Animation). An animation illustrating this experiment is provided as an attachment. This visualization effectively highlights the differences in the pathfinding between the various cost strategies. We encourage you to watch it before proceeding with the subsequent sections, as it will be referenced during the evaluation. The animation is exactly 120 seconds long, rendered at 24 frames per second. Although the simulation time does not correspond to real-world traversal speeds, it was chosen to represent movement across the terrain on a scale suitable for visual interpretation.

Crowd Simulation Experiment

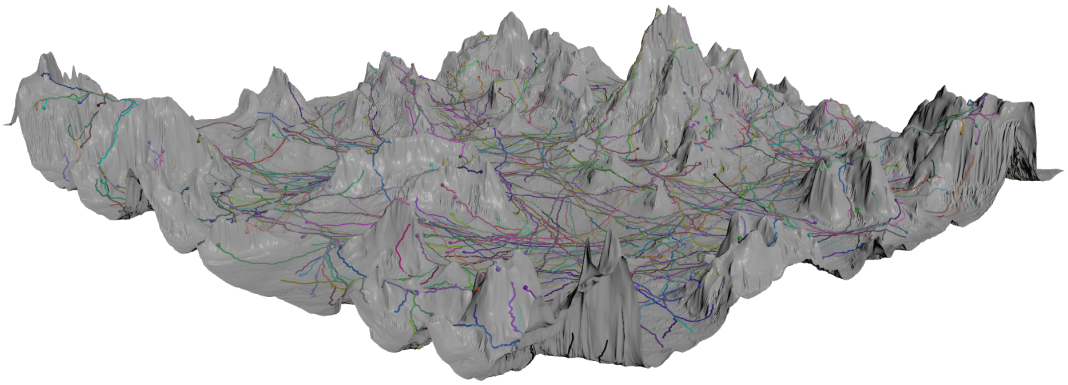


Figure 5.5 Multi agent experiment

The experiment on crowd simulation involved placing agents at random positions, uniformly spread across the terrain. A total of 500 paths were generated utilizing joint cost function based on experimental heading and slope cost functions detailed in this chapter. Animations were produced to depict the interactions among multiple agents and the dynamics of the crowd.

Remark (Animation). An animation illustrating this experiment is included as an attachment, showcasing behavior of the crowd simulation. This visualization effectively demonstrates congestion patterns and multi agent dynamics. As with the single-agent simulation, the duration is exactly 120 seconds at 24 frames per second.

5.3 Cost Function Comparison

This section presents an analysis of agent behavior under the four cost functions outlined in the experimental design. To visualize traversal, the terrain

surface is color-coded based on its slope: light gray indicates flat surfaces, while black represents steep inclines. All paths were reparametrized according to their respective costs, ensuring that each agent reaches the final destination precisely at $120\,[s]$. This normalization allows for consistent temporal comparison across all pathfinding strategies.

Visual Temporal Analysis

In this subsection, we take a concise look at agent behaviors at specific moments throughout the simulation, highlighting key decision-making junctures and path anomalies. This targeted analysis provides insight into the strengths and weaknesses of each cost function under varying terrain conditions.

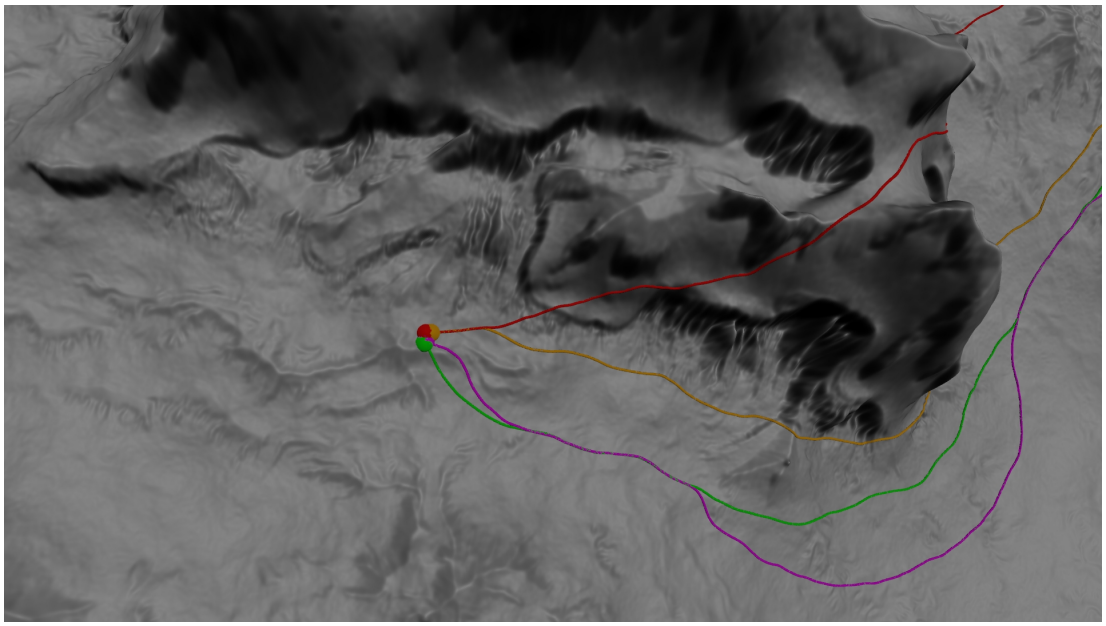


Figure 5.6 Temporal analysis at: $0\,[s]$

Remark (Time: $0\,[s]$). This is the initial starting point for all agents, positioned with equal opportunities to explore their respective pathfinding strategies.

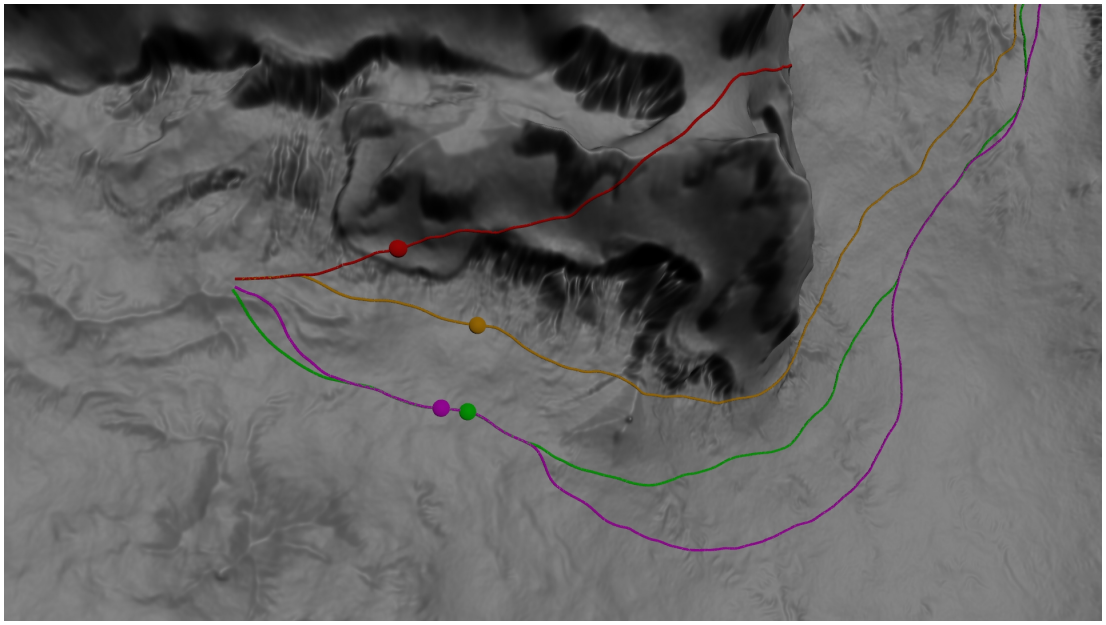


Figure 5.7 Temporal analysis at: 7 [s]

Remark (Time: 7 [s]). At this timestamp, the distance-based agent begins scaling a steep mountain that appears to be a shortcut in terms of raw distance. However, the steep incline imposes significant traversal costs, highlighting a core limitation of purely distance-optimized paths.

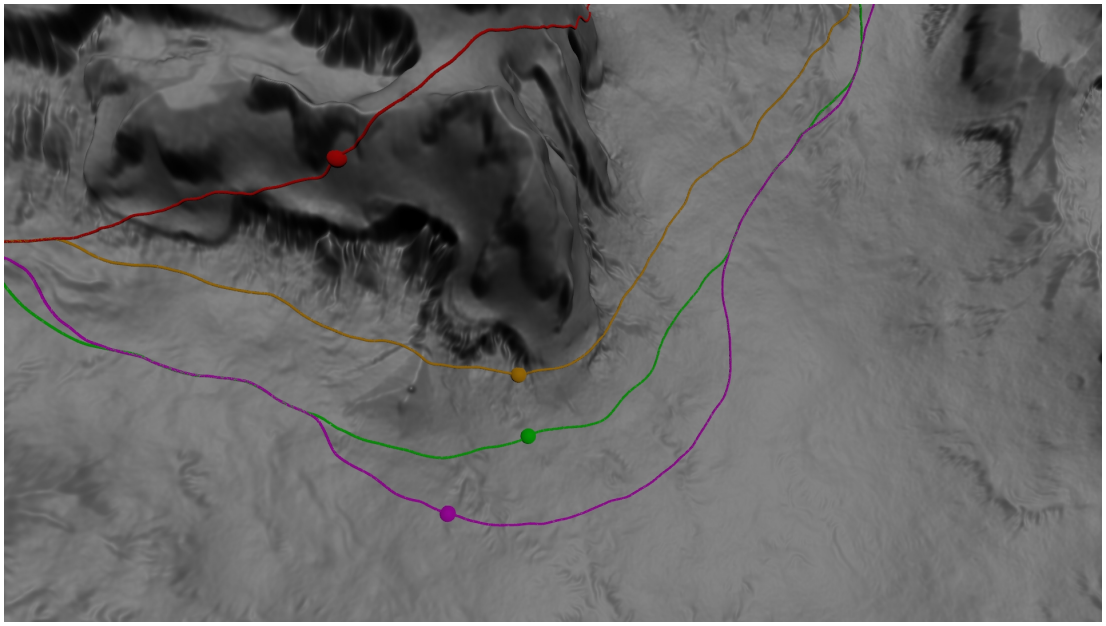


Figure 5.8 Temporal analysis at: 16 [s]

Remark (Time: 16 [s]). Here, all agents visibly diverge according to their cost priorities. The distance-based agent continues uphill, while the heading-based agent moves along steep contours. In particular, the slope-based agent takes a larger detour to avoid higher slope angle, whereas the joint cost agent maintains a balanced trajectory, reflecting a more human-like decision-making process.

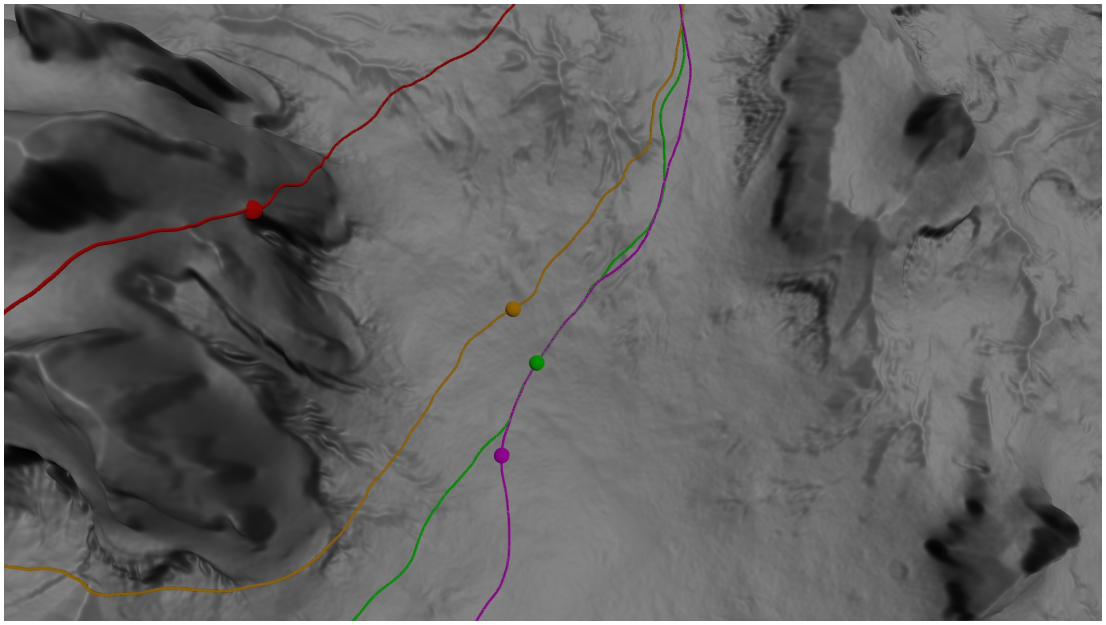


Figure 5.9 Temporal analysis at: 30 [s]

Remark (Time: 30 [s]). Paths of the joint cost agent and slope-based agent converge, though the slope-based agent arrives slightly later due to its extended detour. This demonstrates the effectiveness of joint optimization in reducing unnecessary traversal time.

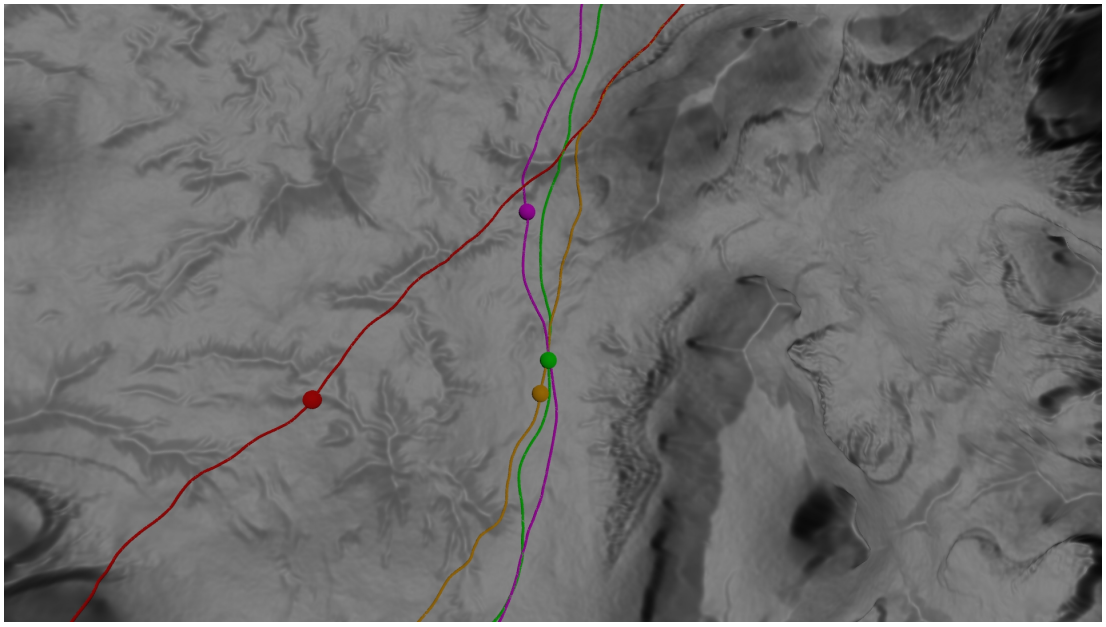


Figure 5.10 Temporal analysis at: 47 [s]

Remark (Time: 47 [s]). At this point, all agents except the distance-based agent share a convergence path. Interestingly, the slope-based agent is significantly ahead, taking advantage of a large low-slope region—an optimal decision given its cost function.

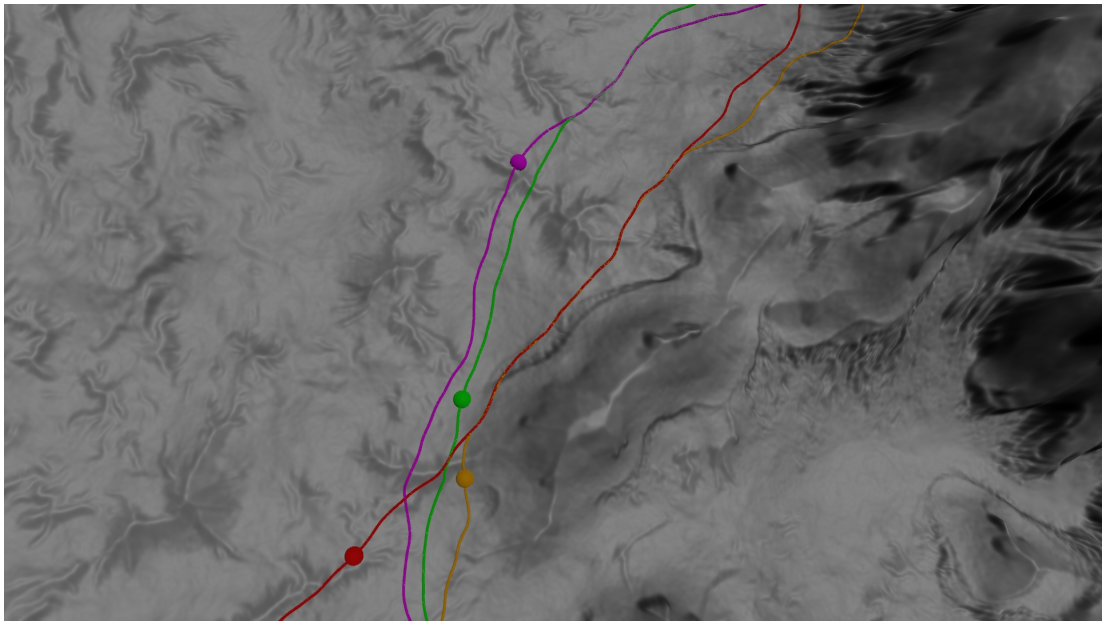


Figure 5.11 Temporal analysis at: 57 [s]

Remark (Time: 57 [s]). While all agent paths have now converged closely, they reached this position at slightly varying time stamps, with the slope-based agent still holding the lead.

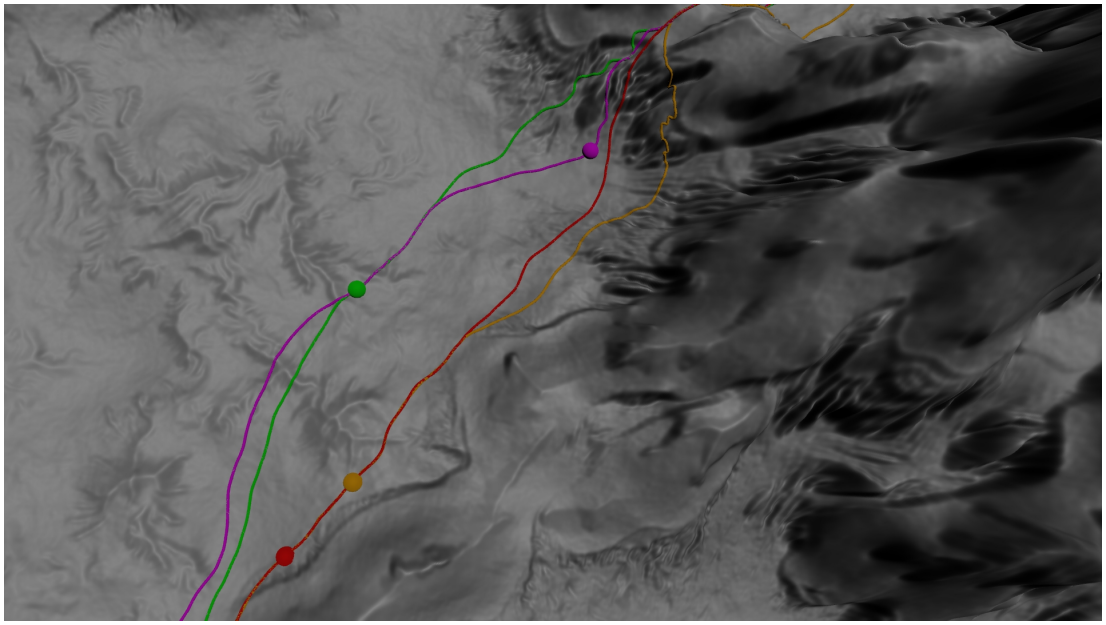


Figure 5.12 Temporal analysis at: 68 [s]

Remark (Time: 68 [s]). The slope-based agent encounters a hill and evaluates it as worth traversing rather than avoiding, other agents took the same decision, but did not arrive yet.

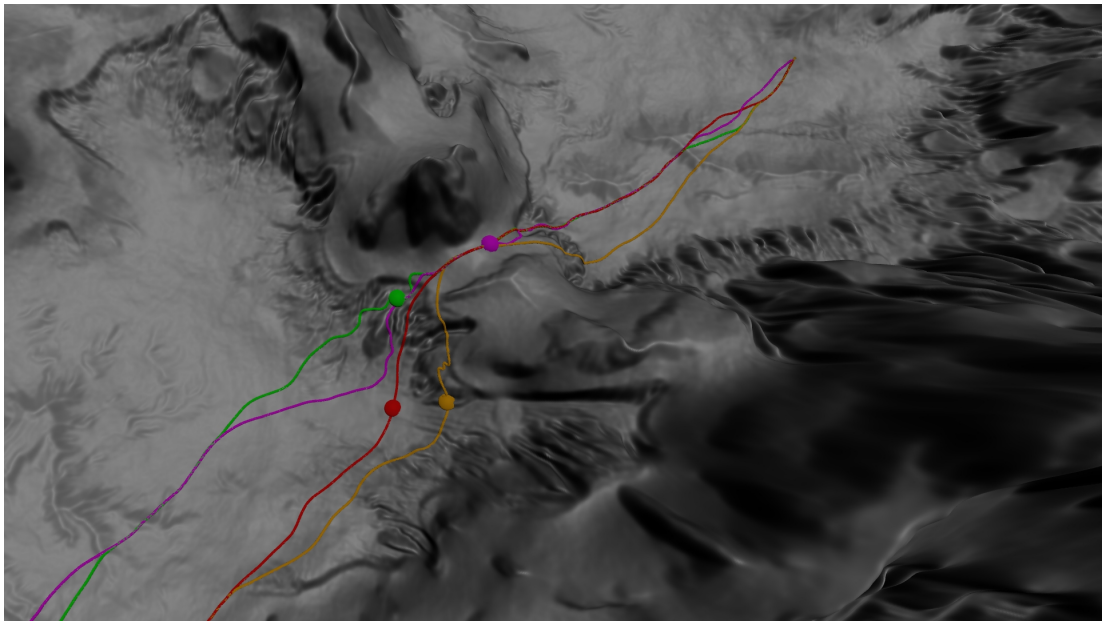


Figure 5.13 Temporal analysis at: 89 $[s]$

Remark (Time: 89 $[s]$). All agents have now arrived at the hill. At this stage, the distance-based agent continues at a steady speed despite the terrain, due to its absence of slope penalization. In contrast, the others experience a significant reduction in speed.

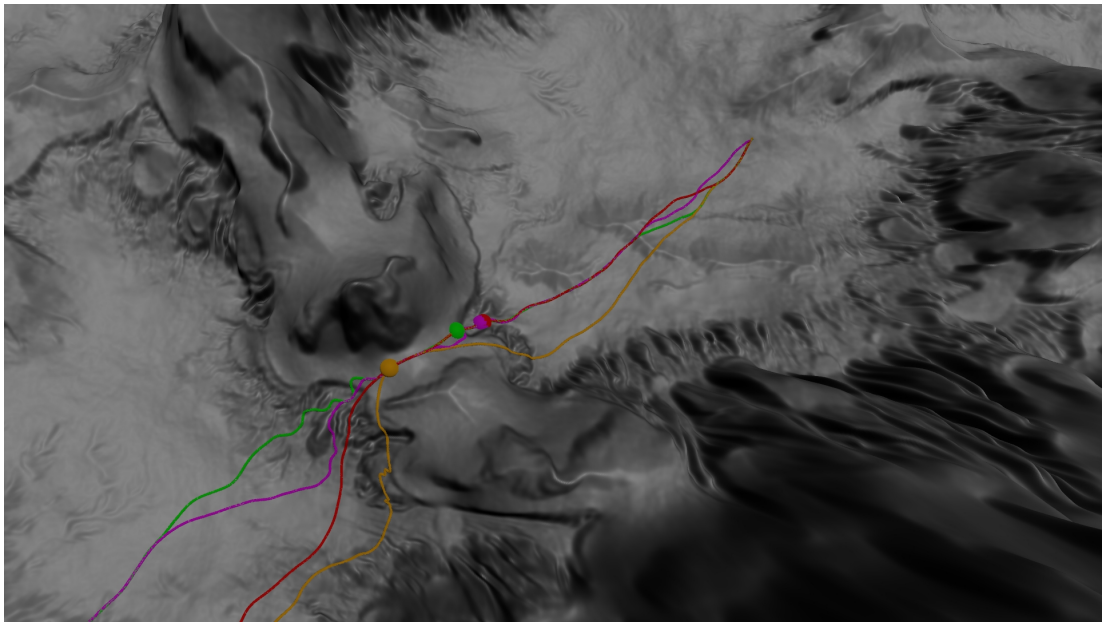


Figure 5.14 Temporal analysis at: 102 $[s]$

Remark (Time: 102 $[s]$). As the agents progress along the hill, both the heading-based and joint cost agents exhibit smoother movement. In contrast, the distance-based agent behaves sub-optimally by not modulating its speed while ascending or descending. Meanwhile, the slope-based agent does not exhibit a higher speed when moving downhill, compared to its uphill pace.

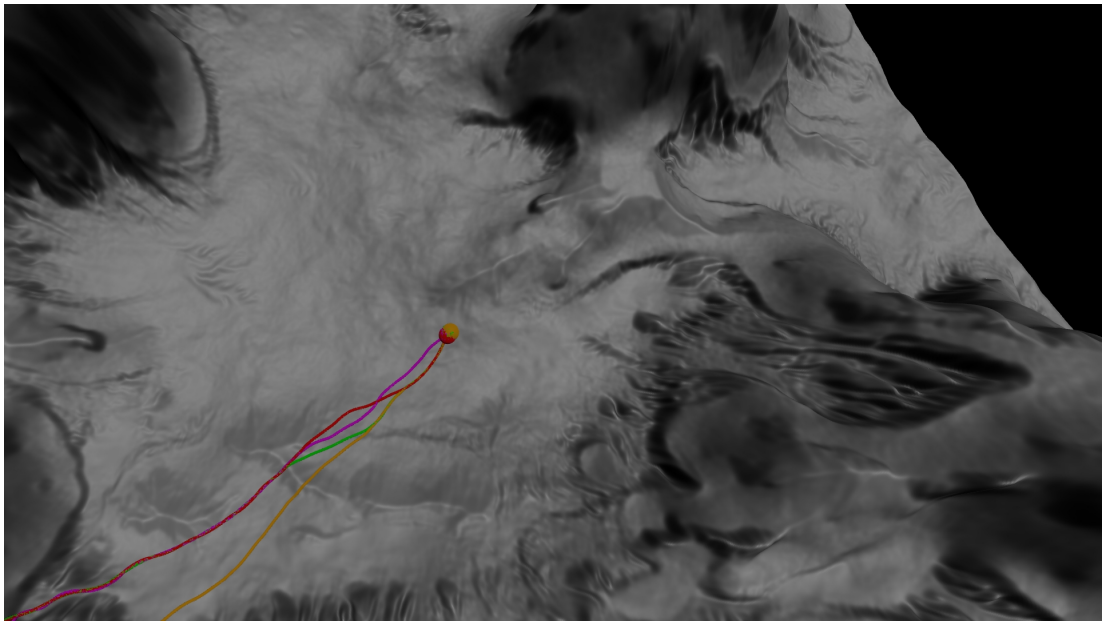


Figure 5.15 Temporal analysis at: 120 $[s]$

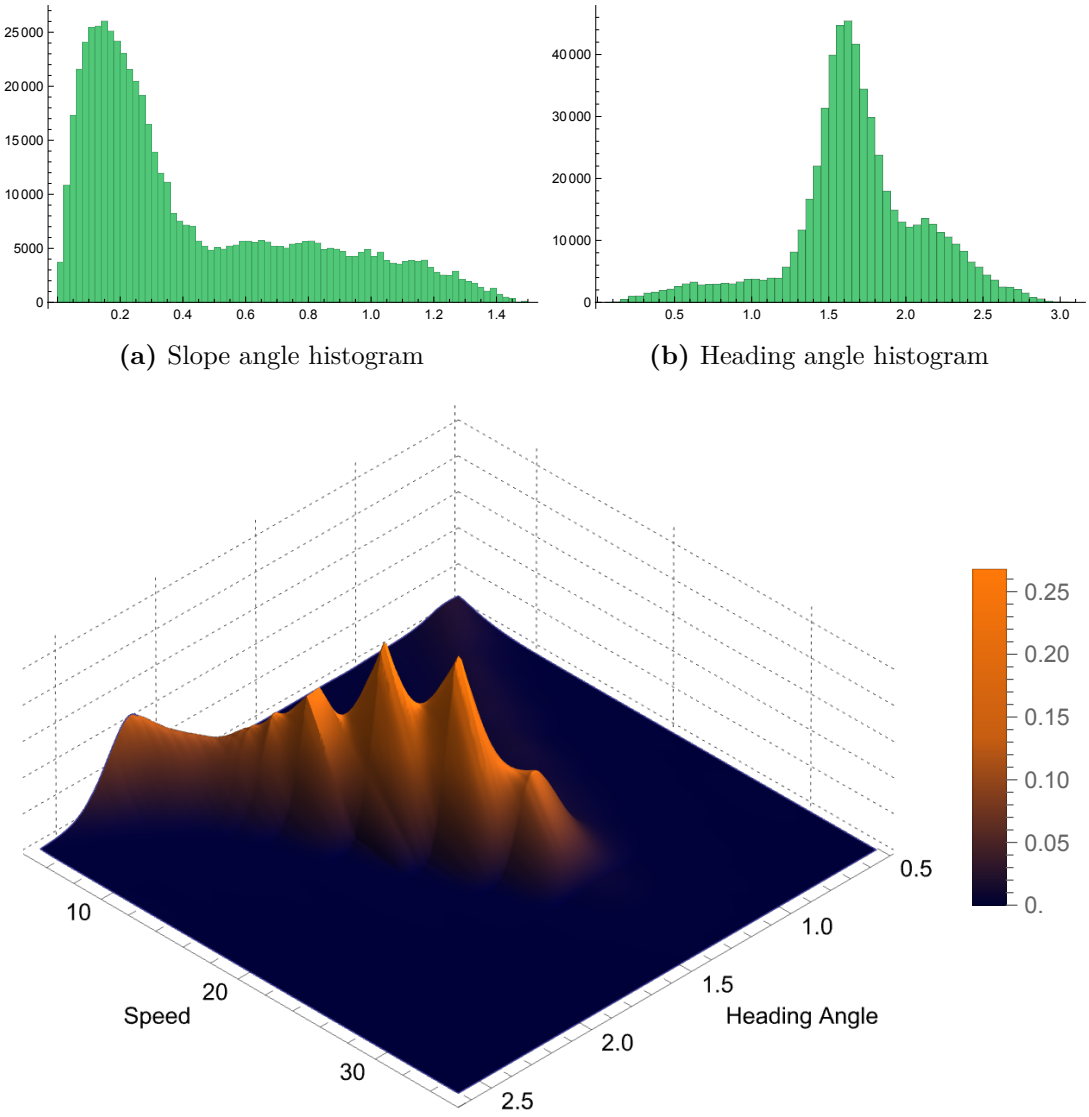
Remark (Time: 120 $[s]$). All agents successfully reach the target point. The joint cost agent demonstrates the most balanced approach, optimizing both slope and heading while avoiding costly detours.

Conclusions

The analysis reveals clear distinctions in agent behavior based on the chosen cost function. The distance-based agent, while efficient on flat terrain, exhibits highly inefficient traversal on steep inclines, sacrificing energy for minimal distance gains. The heading-based agent generally maintains good speed but is prone to traversing steep contours unnecessarily. The slope-based agent effectively avoids sharp elevation gains and steep slopes but does not accordingly modulate its speed when moving downhill, maintaining a consistent speed with uphill movement. Finally, the joint cost agent demonstrates the most balanced and efficient traversal, optimizing both heading and slope to navigate the terrain efficiently. This evidence strongly suggests that a combined cost function is superior for realistic and optimal pathfinding in uneven environments.

5.4 Agent Behavior Analysis

This section presents an analysis of agent behavior within our simulation. Throughout the simulation, for each frame, we recorded critical agent data, including its position, heading direction, speed, and proximity to other agents. The proximity data is reserved for analysis in the subsequent section, focusing on collision detection and interaction patterns.



(a) Slope angle histogram

(b) Heading angle histogram

(c) Speed and heading angle density histogram

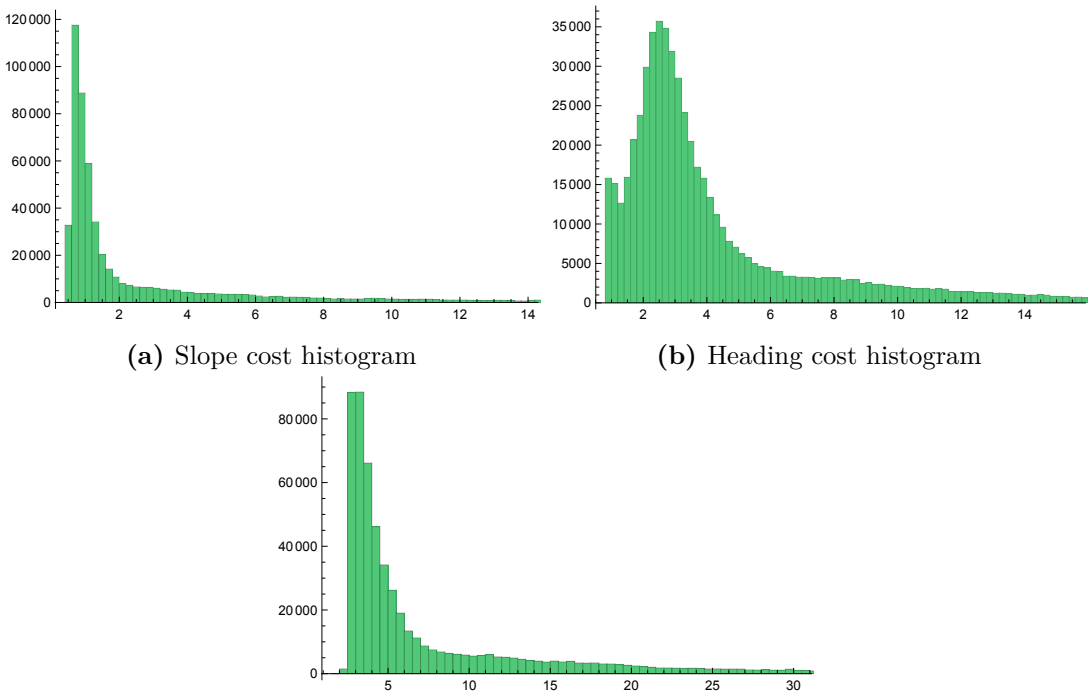
Figure 5.16 Heading, slope and speed histograms

The distribution of slope angles in Figure 5.16a is in agreement with the expected result: agents predominantly traverse regions of low slope, reflecting the influence of the slope cost function, which heavily penalizes steep inclines. This behavior optimally reduces the traversal costs and aligns with human-like path-finding strategies.

The heading angle distribution, shown in Figure 5.16b, initially appears unexpected, as it indicates a higher frequency of upward movement compared to downward. This contradicts the local minimum of the cost function at approximately 0.9, which theoretically encourages downhill movement. This discrepancy can be explained by the speed difference: agents move faster when heading downhill, spending less time on low-cost paths, resulting in fewer recorded samples. This is further corroborated by the density plot in Figure 5.16c, which confirms

that the agents accelerate downhill, supporting the observed sample distribution.

Aggregate cost properties



(a) Slope cost histogram

(b) Heading cost histogram

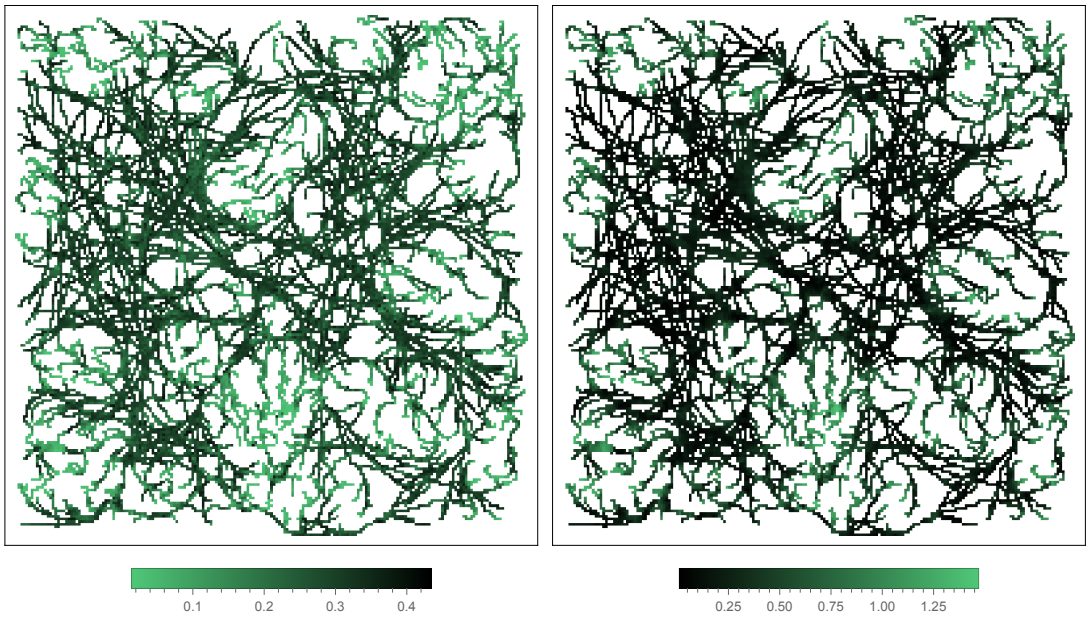
(c) Joint cost histogram

Figure 5.17 Heading, slope and speed histograms

The cost distributions shown in Figure 5.17 reflect the traversal preferences observed in the geometric analysis. The heading cost histogram (Figure 5.17b) displays a subtle valley near its lowest costs, consistent with the prior analysis of limited samples in low-heading regions. This result reinforces the observation that agents spend less time on downhill trajectories due to velocity-induced sampling bias.

Both slope and heading costs follow patterns aligned with their respective cost functions, validating the behavioral adherence observed earlier. Furthermore, the joint cost histogram (Figure 5.17c) reveals a distribution skewed towards lower costs, indicating that agents strongly prioritize less penalized paths when using the combined cost metric. This distribution suggests that the joint cost function effectively enforces optimization across both slope and heading simultaneously.

Spatial traversal speed and cost analysis



(a) Inverse joint cost **(b)** Slope angle

Figure 5.18 Spatial distribution of joint cost values and heading angles

For better visualization, the cost function in Figure 5.18a is presented as its inverse. Both plots depict the average measurements of the agents across the surface of the terrain. The visualization of the slope angle in Figure 5.18b confirms that agents predominantly explore regions of low slope, a behavior consistent with the earlier histogram analysis. Compared with the cost distribution in Figure 5.18a, we observe a strong correspondence between low average slope and reduced traversal cost, verifying the effectiveness of the cost function in penalizing steep inclines.

Speed and cost relationship

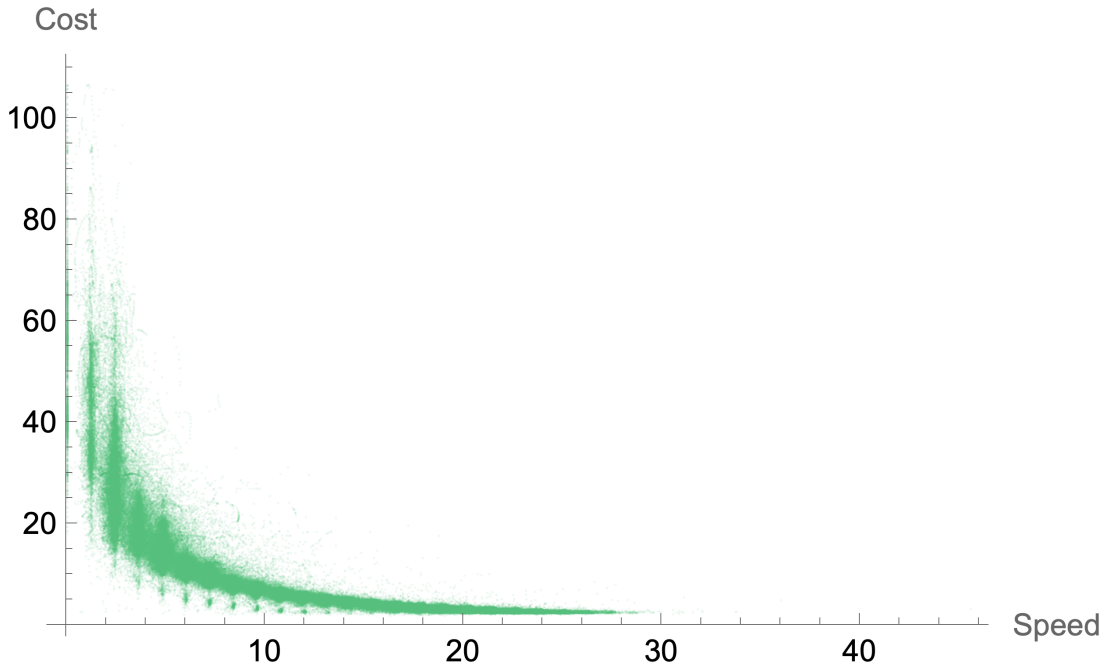


Figure 5.19 Relationship between speed and cost

Figure 5.19 illustrates the inverse relationship between agent speed and cost. As cost increases, agents decelerate, supporting the hypothesis that our reparametrization effectively modulates traversal speed based on terrain difficulty. This behavior confirms that agents encountering difficult terrain slow down, while those traversing flat surfaces accelerate. The noise observed in the plot is attributed to curve smoothing during path interpolation.

5.5 Collission Analysis

In this section, we briefly explore areas where our agents experience collisions and how often they collide. To define what we mean by collision, collision for us occurs whenever the distance of the agent to any other agent is less than the sum of their radii. Since we consider all agents to have radius 1, a collision occurs whenever the distance between two agents is less than 2. Whenever collision happens, we count it for each agent individually; therefore, if two agents collide, we would have two collision events. Also note that during this analysis, once the agent finishes its path, it is removed from the simulation, and it does not contribute to the number of collisions.

Collision Hot-spots

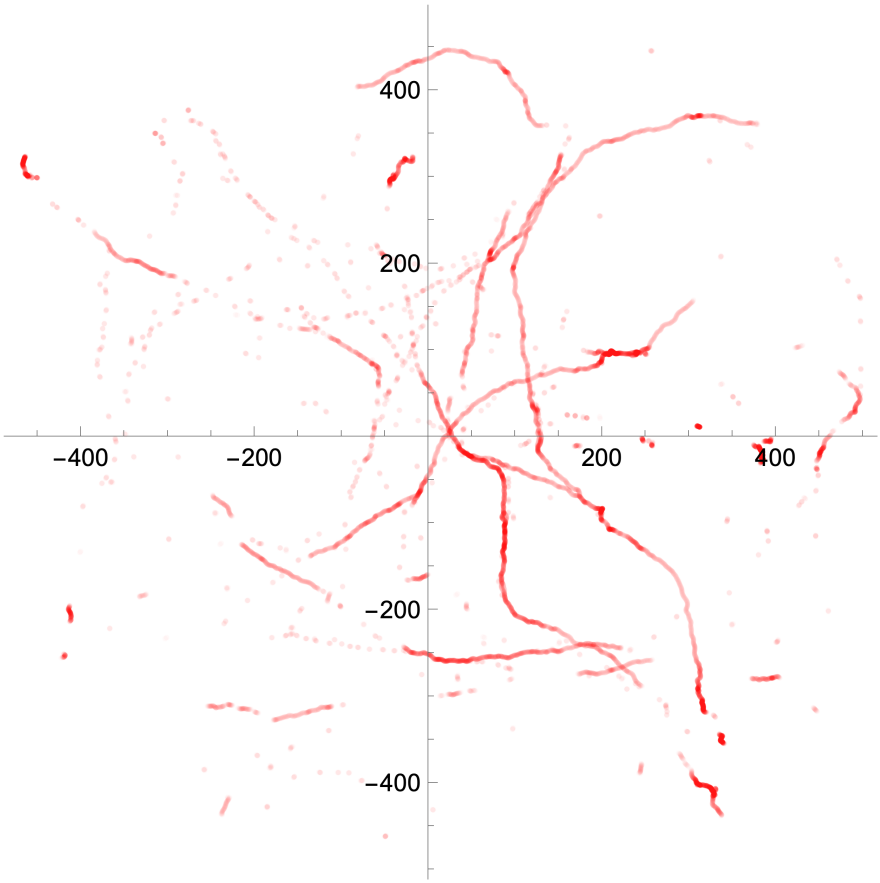


Figure 5.20 Collision locations

As we can see in Figure 5.20 collisions predominantly occur on relatively narrow paths, which we suspect are regionally optimal traversal paths, therefore most agents pick such routes. Interestingly, relative to surface area more collisions seem to happen closer to the center, which is likely caused by the strategy chosen for selecting agent starting and target locations.

Concurrent Collisions

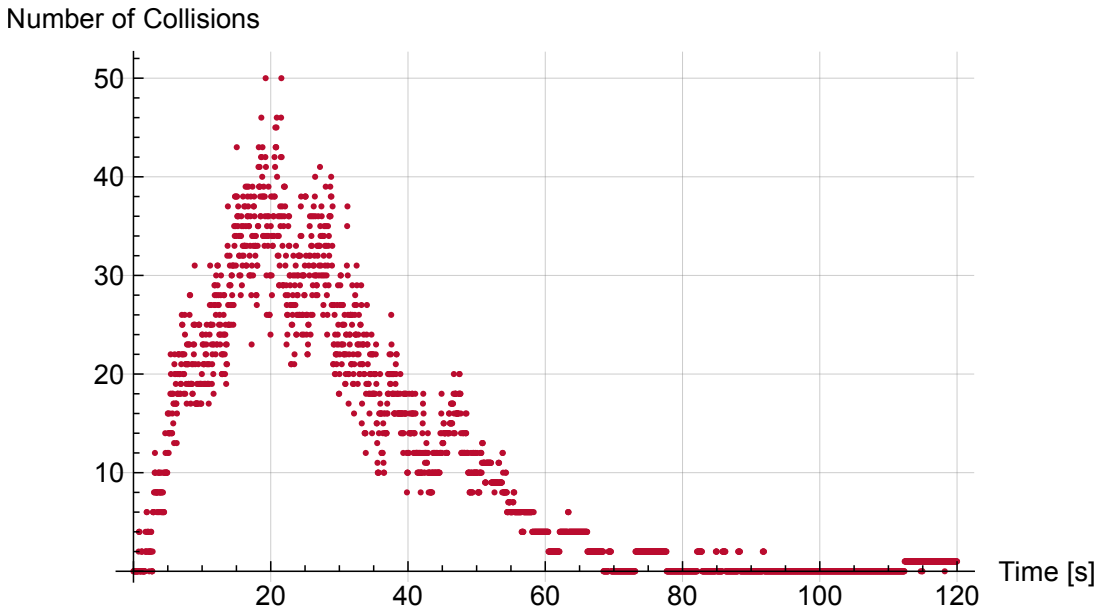


Figure 5.21 Number of concurrent collisions

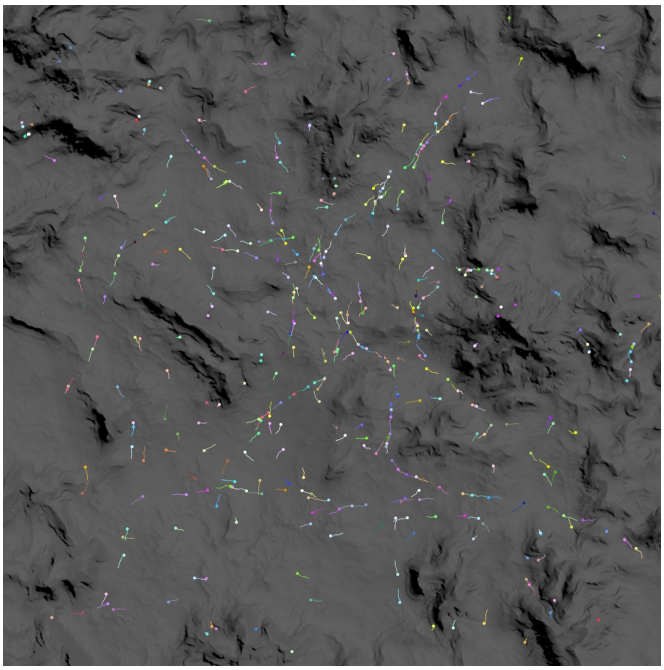


Figure 5.22 Crowd simulation at 20 $[s]$

Observe in Figure 5.21 that the number of collisions that occur initially increases until it reaches its peak at around the 20 $[s]$ time stamp. At this point, agents had traversed a nontrivial portion of their prescribed path and most did not finish yet; therefore, they were not removed from the simulation. When we visualize this timestamp as a keyframe of the attached animation, we can observe the congestion points.

Evaluation Summary

During the evaluation chapter, we explored the performance of our crowd simulation framework, focusing on various cost functions, agent behavior, and collision events. We summarize key findings as follows.

Single-Agent Experiment Analysis

- The joint cost function consistently produced the most human-like paths, balancing both spatial and speed-wise behavior
- Slope cost function effectively captured slope avoidance. However, it often took significant detours, and its uphill speed was the same as its downhill speed.
- The slope-based cost function captured the difference between uphill and downhill motion well, but the agent did not avoid steep contours of the terrain
- The distance-based function favored geometrically shortest paths, but it failed to capture the perceived difficulty of terrain traversal

Multi-Agent Behavior Analysis

The multi-agent experiment provided us with a large number of data points and helped us verify that agents effectively avoid both steep slopes and sharp elevation gains. We have also verified that speed corresponds to the perceived terrain traversal difficulty.

Multi-Agent Collision Analysis

Collisions mainly occurred in narrow paths that are likely locally optimal. The number of collisions for a given number of agents was relatively high, highlighting the need for a collision avoidance strategy.

Implications and final remarks

The experiments validate the joint function strategy as a strong option when planning motion. Future work could generalize cost functions even further to capture more detailed behavior.

Conclusion

We set out to develop a continuous terrain traversal model based on differential geometry that encodes diverse movement behaviors through smooth path-cost evaluation and integrates seamlessly into crowd simulations through terrain discretization, graph-based pathfinding, and curve interpolation. Although designed for crowd movement, this framework also serves as a rigorous mathematical backbone applicable to other domains beyond crowd simulation.

Conceptual Summary

Terrain as a Smooth Manifold

The terrain is modeled as a smooth two-dimensional manifold embedded in $\mathbb{R}^3$ via a height map. This guarantees the existence of smooth paths connecting any pair of surface points, providing a robust setting for analyzing classes of trajectories.

Flexible Cost Function Framework

We introduced two generic cost functions:

- Heading cost penalizes movement direction relative to gravity, capturing uphill/downhill difficulty.
- Slope cost penalizes local incline, capturing effort along steep contours.

By combining these, the joint cost can be tailored to model a wide variety of agent preferences (e.g., favoring gentle slopes, avoiding sharp ascents, or balancing distance and effort).

Cost Based Reparametrization

We showed how to reparametrize any trajectory so that an agent's instantaneous speed reflects local traversal difficulty. High-cost regions yield slower progress; low-cost regions allow faster motion. This links spatial cost directly to temporal behavior, producing smooth, realistic motion without ad-hoc rules.

Proof of Concept Implementation

While the central contribution is theoretical, we discretized the terrain into a graph, sampled vertices via parametrization, generated edges, evaluated edge costs, and applied standard pathfinding. Interpolating discrete paths back into smooth curves illustrated the model's expressivity: agents guided by our joint cost framework navigate more balanced, realistic routes than those optimizing only distance or a single cost.

Reflection

Studying terrain traversal in a continuous setting offers richer geometric insight than starting from discrete representations. The smooth-manifold perspective motivated much of this work and furnished an extensible framework. In many cases, analytic integration yields exact cost values; where it does not, numerical integration provides arbitrarily precise estimates. This dual capability—analytic when possible, numerical when necessary—underscores the strength of a continuous approach.

Future Work

There are several promising directions to extend and deepen this framework:

Generalized Surface Models

Height maps suffice for many scenarios but fall short in environments like caves or overhangs. Adopting more general manifold representations would broaden applicability.

Additional Cost Functions

Beyond heading and slope, one could introduce curvature-based costs (to penalize sharp turns) or torsion-based costs (to penalize twisting motions). A spatial cost field could explicitly discourage traversal of hazardous or restricted areas.

Rigorous Discretization Guarantees

Developing discretization strategies with theoretical guarantees on path-cost optimality—perhaps via discrete differential geometry—would strengthen the connection between the continuous model and its computational approximation.

Multi-Agent Collision Avoidance

Integrating explicit multi-agent pathfinding algorithms or local collision-resolution techniques into the continuous framework would enable realistic group dynamics and safety constraints.

Conclusion

In short, there are many avenues to improve and apply this system further, and the ideas presented here offer valuable insights. We hope these foundations will serve as a versatile platform for future research.

Disclaimer: This text was written using OpenAI, Writeful, and Grammarly AI models. The models were used to check grammar, provide stylistic suggestions, and occasionally paraphrase existing text.

Bibliography

1. Reynolds, Craig W. Flocks, herds and schools: A distributed behavioral model. *ACM SIGGRAPH Computer Graphics* [online]. 1987, vol. 21, no. 4, pp. 25–34 [visited on 2025-06-25]. ISSN 0097-8930. Available from DOI: 10.1145/37402.37406.
2. Yang, Shanwen; Li, Tianrui; Gong, Xun; Peng, Bo; Hu, Jie. A review on crowd simulation and modeling. *Graphical Models* [online]. 2020, vol. 111, p. 101081 [visited on 2025-04-29]. ISSN 15240703. Available from DOI: 10.1016/j.gmod.2020.101081.
3. Helbing, Dirk; Molnár, Péter. Social force model for pedestrian dynamics. *Phys. Rev. E.* 1995, vol. 51, pp. 4282–4286. Available from DOI: 10.1103/PhysRevE.51.4282.
4. Treuille, Adrien; Cooper, Seth; Popović, Zoran. Continuum crowds. *ACM Trans. Graph.* 2006, vol. 25, no. 3, pp. 1160–1168. ISSN 0730-0301. Available from DOI: 10.1145/1141911.1142008.
5. Tao, Terence. *Analysis II*. Vol. 38 [online]. Singapore: Springer Nature Singapore, 2022 [visited on 2025-04-19]. Texts and Readings in Mathematics. ISBN 978-981-19-7284-3. Available from DOI: 10.1007/978-981-19-7284-3.
6. Tao, Terence. *Analysis I*. Vol. 37 [online]. Singapore: Springer Nature Singapore, 2022 [visited on 2025-04-19]. Texts and Readings in Mathematics. ISBN 978-981-19-7261-4. Available from DOI: 10.1007/978-981-19-7261-4.
7. Munkres, James R. *Analysis on manifolds*. Redwood City, Calif: Addison-Wesley Pub. Co., Advanced Book Program, 1991. ISBN 978-0-201-51035-5.
8. Ebert, David S. (ed.). *Texturing and modeling*. 2. ed. San Diego, Calif.: AP Professional, 1998. ISBN 978-0-12-228730-5.
9. Pressley, Andrew. *Elementary Differential Geometry* [online]. London: Springer London, 2010 [visited on 2025-04-29]. Springer Undergraduate Mathematics Series. ISBN 978-1-84882-890-2 978-1-84882-891-9. Available from DOI: 10.1007/978-1-84882-891-9.
10. Foley, James D. (ed.). *Computer graphics: principles and practice*. 2. ed. in C, reprinted with corr., 25. print. Boston: Addison-Wesley, 2010. The systems programming series. ISBN 978-0-201-84840-3.
11. Hirsch, Morris W. *Differential topology*. New York Heidelberg Berlin: Springer-Verlag, 1976. Graduate texts in mathematics, no. 33. ISBN 978-1-4684-9451-8 978-1-4684-9449-5. Available from DOI: 10.1007/978-1-4684-9449-5.
12. Inc., Wolfram Research. *Mathematica, Version 14.2*. [N.d.]. Available also from: https://www.wolfram.com/mathematica. Champaign, IL, 2024.
13. Carmo, Manfredo Perdigão do. *Differential geometry of curves and surfaces*. Englewood Cliffs: Prentice Hall, 1976. ISBN 978-0-13-212589-5.
14. Cook, Robert L. Stochastic sampling in computer graphics. *ACM Trans. Graph.* 1986, vol. 5, no. 1, pp. 51–72. ISSN 0730-0301. Available from DOI: 10.1145/7529.8927.

15. Bridson, Robert. Fast Poisson disk sampling in arbitrary dimensions. In: *ACM SIGGRAPH 2007 Sketches*. San Diego, California: Association for Computing Machinery, 2007, 22–es. SIGGRAPH '07. ISBN 9781450347266. Available from DOI: 10.1145/1278780.1278807.

16. Lloyd, S. Least squares quantization in PCM. *IEEE Transactions on Information Theory*. 1982, vol. 28, no. 2, pp. 129–137. Available from DOI: 10.1109/TIT.1982.1056489.

17. Balzer, Michael; Schlömer, Thomas; Deussen, Oliver. Capacity-constrained point distributions: a variant of Lloyd's method. *ACM Transactions on Graphics* [online]. 2009, vol. 28, no. 3, pp. 1–8 [visited on 2025-03-30]. ISSN 0730-0301, ISSN 1557-7368. Available from DOI: 10.1145/1531326.1531392.

18. Khan, Dawar; Plopski, Alexander; Fujimoto, Yuichiro; Kanbara, Masayuki; Jabeen, Gul; Zhang, Yongjie Jessica; Zhang, Xiaopeng; Kato, Hirokazu. Surface Remeshing: A Systematic Literature Review of Methods and Research Directions. *IEEE Transactions on Visualization and Computer Graphics*. 2022, vol. 28, no. 3, pp. 1680–1713. Available from DOI: 10.1109/TVCG.2020.3016645.

19. *Computational geometry: algorithms and applications*. 3rd ed. Berlin: Springer, 2008. ISBN 978-3-540-77973-5.

20. Cormen, Thomas H.; Leiserson, Charles Eric; Rivest, Ronald Linn; Stein, Clifford. *Introduction to algorithms*. Fourth edition. Cambridge, Massachusetts London: The MIT Press, 2022. ISBN 978-0-262-04630-5 978-0-262-36750-9.

21. Russell, Stuart; Norvig, Peter. *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson, 2020. ISBN 9780134610993. Available also from: http://aima.cs.berkeley.edu/.

22. Stern, Roni; Sturtevant, Nathan; Felner, Ariel; Koenig, Sven; Ma, Hang; Walker, Thayne; Li, Jiaoyang; Atzmon, Dor; Cohen, Liron; Kumar, T. K. Satish; Boyarski, Eli; Bartak, Roman. *Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks*. 2019. Available also from: https://arxiv.org/abs/1906.08291. _eprint: 1906.08291.

23. Catmull, Edwin; Rom, Raphael. A CLASS OF LOCAL INTERPOLATING SPLINES. In: Barnhill, ROBERT E.; Riesenfeld, RICHARD F. (eds.). *Computer Aided Geometric Design* [online]. Academic Press, 1974, pp. 317–326 [visited on 2025-05-14]. ISBN 978-0-12-079050-0. Available from DOI: 10.1016/B978-0-12-079050-0.50020-5.

24. SideFX. *Houdini Documentation*. 2025. Available also from: https://www.sidefx.com/docs/houdini/. Accessed: 2025-05-15.

List of Figures

A Attachments

All attachments are stored inside the *Attachments.zip* archive. The structure of the archive is as follows:

- Houdini Project - contains the implementation of the experiments,
- Experiment Results - contains results of the experiments,
- Animations - contains rendered animations.

A.1 Houdini Project

Houdini Project contains *main.hiplc* file, which is proprietary format used to store Houdini projects. In order to open it you will need to download *SideFX Houdini*. *SideFX* provides free version of *Houdini* called *Houdini Apprentice*, which is able to open this file. We have used *Houdini* 20.5.613 on *MacOS* operating system, however the file should work fine on *Houdini* 20.5.X, where $X > 613$.

The subdirectory *geo* contains cached data from simulation and terrain generation. The project is configured so that if you open the *main.hiplc* file from the project directory, the cached files should load, as they use relative addressing. If cache fails to load properly, you might have to relink the cached data to their respective file cache nodes.

Once you open the file, you should see the project as displayed in the figure below. However, note that Houdini by default uses the Y axis as the upward direction, therefore, you might need to change the setting as described in the notes within the project.

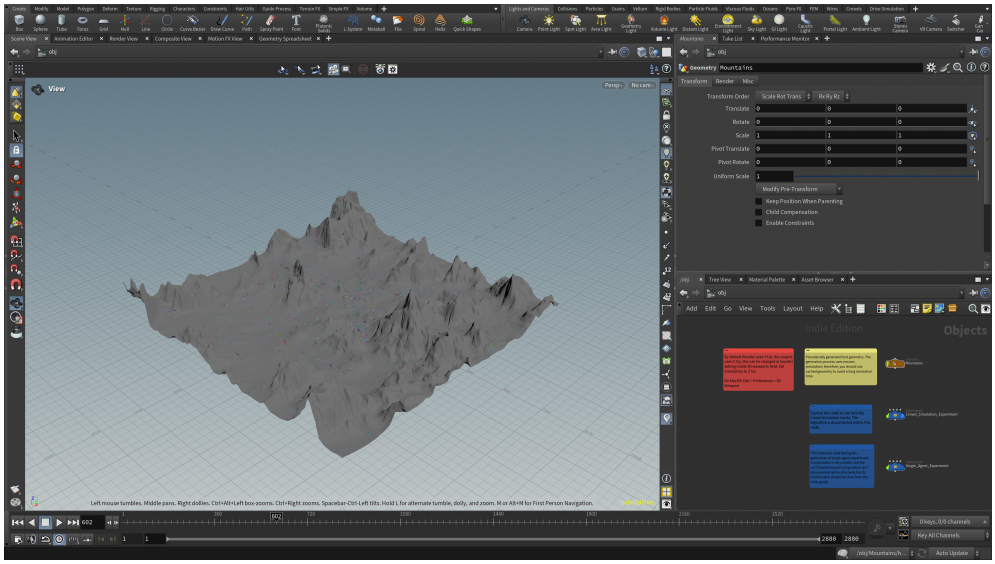


Figure A.1 Houdini Project

It might take a few seconds for the scene to load due to certain computations needing to be executed prior, once that is done, you should be able to scroll

through the animation using the keyframes slider on the bottom of the user interface.

When you click on certain nodes, its properties are displayed in the Properties panel. Double clicking on a node takes you to the subnetwork that defines the given node. For example, to inspect how the simulation works, double-click on the *Crowd_Simulation_Experiment* node.

If you have any trouble navigating Houdini, please refer to documentation. *Houdini* is a relatively complex application that is used mainly in professional computer graphics applications, and describing how to use the system is beyond the scope of this thesis. Therefore, we assume certain working knowledge of the system in order to fully grasp how the simulation works.

A.2 Experiment Results

The results of the experiment are stored in *csv* format, where each file represents a distinct frame of the simulation. The experiment results stored in the directory are possibly slightly different from the results presented in the evaluation chapter due to accidental seed change. This affects starting and target positions of agents; however, the underlying position generating distribution is equivalent, therefore statistically measured properties should be nearly equivalent.

The properties in the *csv* files have the following interpretations.

- *point_number* represents the id of the point representing the agent, this value is not persistent throughout the frames, and the agent represented by point X in frame Y might be different from the agent represented by point X in frame Z.
- (x, y, z) represent spatial coordinates
- (tx, ty, tz) represents tangent direction
- (nx, ny, nz) represents surface normal direction
- *speed* represents speed of the agent
- *nearest* represents distance to the closest agent

A.3 Animations

This directory stores animations that were mentioned in the evaluation section. For single-agent simulation, there is one animation with the camera following the agent. For a crowd simulation scenario, there are two views of the same animation, one is a top-view and the other is a perspective projection.