UNIVERSITY OF CALIFORNIA SAN DIEGO

Efficient Grid-Based Algorithms for Visibility Problem in Level Set Framework

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy

in

Mathematics

by

Chuqing Shi

Committee in charge:

Professor Li-Tien Cheng, Chair
Professor Melvin Leok
Professor Bo Li
Professor Xiaochuan Tian
Professor Chia-Ming Uang

2024

The Dissertation of Chuqing Shi is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2024

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

My deepest thanks go to my advisor and committee chair, Professor Li-Tien Cheng, for his invaluable guidance, insight, and unwavering support throughout my research. His diligent and meticulous working style, as well as his positive attitude, has inspired me to approach my work with focus and dedication. I am especially thankful for his emotional support, patience, and generosity in accepting my flaws and mistakes during these years. His mentorship has been a pivotal factor in the completion of this dissertation, and I truly appreciate all the encouragement and constructive feedback he has provided. A thousand thanks!

I would also like to express my heartfelt gratitude to my family and friends for support and encouragement. Lastly, thanks to my cat, Motto, for her constant companionship throughout my graduate studies.

VITA

2017	B.S. in Computational Mathematics, Nankai University
2017–2018	Ph.D. Student and Teaching Assistant, Department of Mathematics University of California San Diego
2018–2019	Approved Leave of Absence (one-year withdrawal from the Ph.D. program)
2019–2022	Resumption of Ph.D. Program, Teaching Assistant, Department of Mathematics University of California San Diego
2022-2023	Ph.D. Candidate and Teaching Assistant, Department of Mathematics University of California San Diego
2023 Fall	Associate Instructor, Department of Mathematics University of California San Diego
2024	Doctor of Philosophy in Mathematics, University of California San Diego

ABSTRACT OF THE DISSERTATION

Efficient Grid-Based Algorithms for Visibility Problem in Level Set Framework

by

Chuqing Shi

Doctor of Philosophy in Mathematics

University of California San Diego, 2024

Professor Li-Tien Cheng, Chair

This research explores the static visibility problem, focusing on identifying visible regions in environments with fixed obstacles and a stationary viewpoint. The problem is formulated as the determination of visibility boundaries within the level set framework. To address this, a novel grid-based algorithm is introduced, designed to improve computational efficiency while maintaining high accuracy. This method integrates phase flow techniques, enables localized computations near the visibility interface with the growing method, and further reduces the

computational burden associated with high-resolution grids with tube structures. Additionally, a new Heaviside function approximation, inspired by advanced Delta function methodologies, is proposed. This approach is conjectured to have second-order accuracy in calculating the volume of visibility regions. All accuracy and computational workload considerations are verified through numerical experiments.

Chapter 1

Background and Introduction

The visibility problem has been a subject of extensive research in computational geometry, computer graphics, and physics. Essentially, the visibility problem seeks to determine which parts of a given environment are visible from a specific viewpoint, and which are occluded by obstacles [9]. In this dissertation, we only consider the static visibility problem, when the observer remains stationary and the environment unchanged.

The visibility problem remains a vibrant area of research due to evolving application demands. Visibility analysis is essential for tasks such as rendering realistic scenes in virtual environments [32, 11], placing surveillance cameras [2], simulating etching process in photolithography [24], and designing efficient communication networks [7]. A wide variety of approaches and algorithms have been developed to address the visibility problem, and these can be broadly categorized into two primary types based on how the problem is presented: explicit and implicit.

With explicit representation, obstacles are clearly defined and directly mapped in the environment and often heavily make use of geometric features — line segments, polygons, polyhedrons, trianglated surfaces [5], or more complex forms. These obstacles are explicitly modeled as barriers to visibility, and the problem becomes one of determining which areas of the environment are obstructed or not obstructed by these shapes when viewed from specific points. Classical approaches such as visibility graphs [19], line-of-sight checks [12], geometric data

structures, and other computational geometry techniques [6] are often employed to solve this problem in environments with explicitly represented obstacles.

On the other hand with implicit representation, obstacles are not directly represented as distinct geometric entities but are instead encoded through environmental features, such as height maps, occupancy grids, or probabilistic representations [34]. In this case, visibility is inferred based on the presence or absence of obstacles within these implicit representations. Level-set frameworks [20, 13], occupancy grids [23], and probabilistic mapping models can be used to determine visibility in such settings. Compared to the well-established and sophisticated research on the visibility problem using explicit representations, the use of implicit representations is relatively new and offers the potential for novel and advantageous ideas.

Among the implicit approaches, the level set framework [20, 13] plays a significant role due to its ability to model complex and irregularly shaped obstacles that are difficult to represent with discrete or explicit forms, in a continuous and highly flexible manner. In this framework, obstacles are implicitly represented by a scalar function, often a signed distance function, where the zero-level set defines the boundaries between free space and obstacles. Thanks to this, level set methods developed for 2D can be easily extended to 3D, with the visibility boundary encoded as a level set contour. Moreover, the level set method allows for a multi-resolution [21, 28] representation of obstacles, meaning that the level set function can be adapted to different scales. This can be useful for handling both fine-grained details of small obstacles and larger, coarser representations of the environment, thus improving the efficiency of visibility algorithms.

Despite advancements, existing algorithms for level set-based visibility analysis are sometimes lacking. One example is in accurate and compact schemes for the approximation of the Delta and Heaviside functions [36, 8, 27]. Geometric information concerning regions of visibility often require integrating quantities over obstacle boundaries or visible regions, which are represented implicitly by the level set function. The Delta Function is used to localize computations to the interface, and the Heaviside function represents regions inside or outside obstacles for visibility computation. An especially interesting case can be found in the problem

of optimizing what can be seen, which relates to maximizing the area or volume of the visible regions.

An especially pervasive issue of level set methods is high computational complexity in naive formulations. The accuracy of level-set methods depends on grid resolution. High-resolution grids improve precision but dramatically increase computational cost as such a grid exists over the ambient space. Adaptive mesh refinement techniques have been explored to balance these trade-offs [25, 3]. Hybrid methods combining level set techniques with traditional geometric [14] or machine learning approaches [15] are also promising avenues [1] to go along with more traditional tube-based approaches [22]. These methods have the potential to significantly enhance the computational efficiency of level-set approaches, making them even more practical, especially in real-world applications.

This dissertation proposes a new grid-based algorithm for determining the visibility status of points at reduced computational cost and with high accuracy near visibility interfaces that separate visible and nonvisible points. The algorithm is built upon the level set framework, integrating concepts from the phase flow method [35] for solving numerical propagation in ordinary differential equations, and is specifically tailored to primarily operate local to visibility interfaces. Additionally, a novel approximation of the Heaviside function is proposed, drawing inspiration from Peter Smereka's work [26] on Delta function approximations, for second-order accurate area or volume calculations of regions, such as regions of visibility. Numerical experiments are employed to justify the efficacy of the algorithms.

The remainder of the dissertation is organized as follows. Chapter 2 introduces the visibility problem within the level-set framework and develops a new algorithm for detecting visibility boundaries. The algorithm is constructed step by step, integrating ideas, techniques, and data structures presented throughout the sections. Chapter 3 presents numerical experiments and results demonstrating the effectiveness and accuracy of the proposed method. Chapter 4 proposes the new approach to Heaviside function approximation and illustrates its effective integration with visibility computations. Lastly, conclusions and discussions for future research

are in Chapter 5.

Chapter 2

Visibility Algorithm

In this chapter, we step up the visibility problem in the level set framework and introduce a new algorithm for detecting the visibility boundary. The algorithm is developed step by step, integrating ideas, techniques, and data structures presented in the sections. Each section contributes incrementally to the algorithm's construction, with its final form presented in the later sections of the chapter.

2.1 Problem Set Up in the Level Set Framework

Given obstacles and an viewpoint (also referred to as the observer or eye-point), we want to determine the visible region, meaning the region in the given environment that is not occluded by any obstacle. This is the central objective of the visibility problem. As shown in Figure 2.1 from online source [33], all points within the yellow area can be reached by a direct, unbroken line of sight from the observer. Therefore, the yellow region represents the visible region, while the red contour marks the visibility interface, which delineates the precise boundaries separating visible and occluded regions. Essentially, the key to distinguishing visible and invisible regions lies in detecting the visibility interface.

The problem for detecting the visibility interface could be viewed in a dynamic way, where the interface of the blocked region is evolved throughout time and transported gradually from the boundary of the obstacle. Therefore, the idea of transport equation is brought up, and

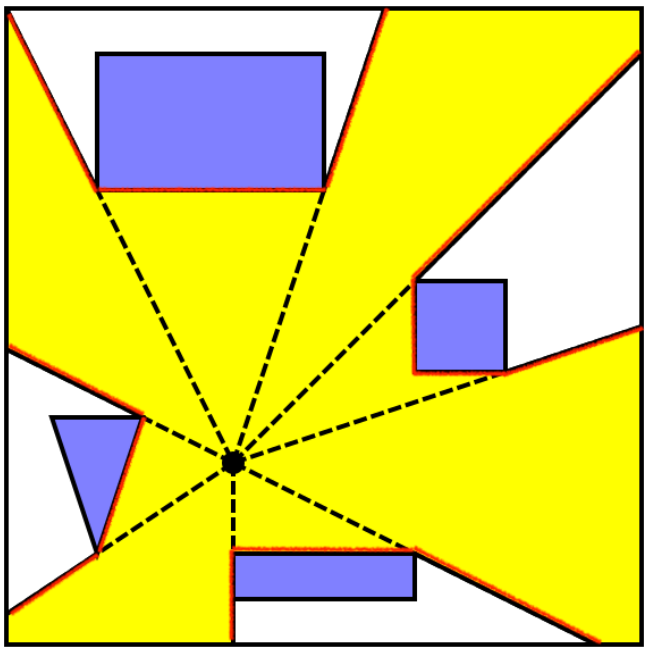


Figure 2.1. The key question in the visibility problem: What regions in a given environment are visible to an observer, and what regions are blocked (occluded) by obstacles? The black dot represents the viewpoint, the purple shapes indicate obstacles, the yellow area denotes the visible region, and the red line outlines the visibility interface.

the use of level set functions to represent obstacles is natural.

The level-set framework makes a great tool for modeling time-varying objects with change in topology. It is primarily used in modeling visibility propagation where a wavefront representing visibility expands from a viewpoint or source [28]. With the help of a level set function $\phi(\mathbf{x}) : \mathbb{R}^d \to \mathbb{R}$, where d is typically 2 or 3 based on the dimension of interest, the surface of the obstacles can be represented by its zero-level set $\{\mathbf{x} : \phi(\mathbf{x}) = 0\}$, while the interior of the obstacles corresponds to $\{\mathbf{x} : \phi(\mathbf{x}) < 0\}$. As a remark, the availability of level set functions ϕ representing obstacles is ensured by the existence of signed distance functions, which naturally provide a representation of obstacle boundaries that is Lipschitz continuous almost everywhere.

The above level set function $\phi(\mathbf{x})$ could be viewed as the initial value of a transport-like problem, where a class of visibility functions $\{\psi(\mathbf{x},t) : \mathbb{R}^d \times \mathbb{R}^+ \to \mathbb{R}\}$ would flow over time, and ultimately settle down to some visibility function $\psi(\mathbf{x}) := \psi(\mathbf{x},\infty)$ to represent the visibility region, where $\{\mathbf{x} : \psi(\mathbf{x}) = 0\}$ is the visibility interface, and $\{\mathbf{x} : \psi(\mathbf{x}) < 0\}$ means invisible region.

Consider a single point observer at location $\mathbf{x}^*$ in the $\mathbb{R}^d$ Euclidean space, d could be 2 or 3 based on the dimension. In this context, there is an implicit vector field $\mathbf{v}$ based on the nature of straight-line propagation of light. Imagine there are light rays shooting from the eye-point, so the vector field has all flow lines radiated away from $\mathbf{x}^*$ isotropically at a steady rate, that is

$$\mathbf{v}(\mathbf{x},t) = \frac{\mathbf{x}-\mathbf{x}^*}{||\mathbf{x}-\mathbf{x}^*||}. \tag{2.1}$$

Therefore the following transport equation is formulated with initial value:

$$\frac{\partial \psi}{\partial t} + \mathbf{v}(\mathbf{x},t) \cdot \nabla \psi = 0, \quad t > 0, \tag{2.2}$$

$$\psi(\mathbf{x},0) = \phi(\mathbf{x}). \tag{2.3}$$

As the propagation advances, however, even with a tiny time step $t_0 > 0$, the zero-level set

of $\psi(\mathbf{x},t_0)$ will enter the obstacle's interior, causing the shape to gradually erode. A point is considered invisible whenever any point along the ray from the eye-point to itself is occluded. Therefore, the erosion shall be pushed out to preserve the original obstacle boundary that blocks the points farther away from the eye-point. A union of the insides can fulfill the need, that is, to update the new invisible area as a union of the current interface interior with the interface interior from previous time. Based on our set up that negative values of ϕ or ψ means invisibility, the "union of insides" could be enforced by minimization of the current state ψ with its previous. For clarity, an iterative algorithm to calculate $\psi(\mathbf{x})$ is outlined accordingly below.

1. Discretize (2.2) in time t with any numerical method, and initialize with (2.3).

2. Iterate until convergence. At step k:

   - Advance for $\psi(\mathbf{x},t_k)$ using any numerical method of choice;
   - Update $\psi(\mathbf{x},t_k)$ with $\min(\psi(\mathbf{x},t_k),\psi(\mathbf{x},t_{k-1}))$;

3. Set $\psi(\mathbf{x})$ to be the last $\psi(\mathbf{x},t_K)$ out of the loop.

To our advantage, the homogeneous transport equation (2.2) can be solved with the method of characteristics [10]. $\psi(\mathbf{x},t)$ is constant along characteristics, i.e. trajectories satisfying the ODE

$$\frac{d\mathbf{x}}{dt} = \mathbf{v}(\mathbf{x},t). \tag{2.4}$$

Hence solving for ψ at some $(\mathbf{x},t)$ actually involves going "backward" along characteristics until initial time (zero in our case) is hit, i.e. to solve an initial value problem with regard to (2.4). The value of $\psi(\mathbf{x},t)$ is then determined by its initial condition accordingly. With (2.1), the above autonomous ODE reveals that each point $\mathbf{x}$ has its characteristics as the flow line tracing from $\mathbf{x}^*$ to itself. Therefore, the initial value problem (2.2) with (2.3) has solution

$$\psi(\mathbf{x},t) = \phi(\mathbf{x}-t\mathbf{v}), \quad 0 \le t \le ||\mathbf{x}-\mathbf{x}^*||. \tag{2.5}$$

Together with minimization of results along the way, the modified procedure is equivalent to a minimization of $\phi(\mathbf{x})$ along the ray tracing back from $\mathbf{x}$ to $\mathbf{x}^*$, that is

$$\psi(\mathbf{x},t) = \min_{L(\mathbf{x}-t\mathbf{v},\mathbf{x})} \phi(\mathbf{x}), \quad 0 \le t \le ||\mathbf{x}-\mathbf{x}^*||, \tag{2.6}$$

where $L(\mathbf{x}-t\mathbf{v},\mathbf{x})$ is the flow line segment from $\mathbf{x}-t\mathbf{v}$ to $\mathbf{x}$. Since the visibility problem has its concentration merely on the visibility interface, $\phi(\mathbf{x})$'s positive value on any point beyond the tracing ray is of no use. After all, as time t goes on until convergence, the actual visibility level set function $\psi(\mathbf{x})$ can be represented by a minimization problem as below:

$$\psi(\mathbf{x}) = \min_{L(\mathbf{x}^*,\mathbf{x})} \phi(\mathbf{x}), \tag{2.7}$$

where $L(\mathbf{x}^*,\mathbf{x})$ is the flow line segment connecting $\mathbf{x}^*$ and $\mathbf{x}$.

With property (2.5) of the transport equation and the above analysis, the previous algorithm to calculate $\psi(\mathbf{x})$ is refined to a more precise one outlined below.

1. Discretize time t with step size τ, and initialize with (2.3).
2. Iterate until $\mathbf{x}-t_k\mathbf{v}$ passes beyond eye-point. At step k:
   - Update $\psi(\mathbf{x},t_k)$ with $\min(\psi(\mathbf{x}-\tau\mathbf{v},t_{k-1}),\psi(\mathbf{x},t_{k-1}))$;
3. Set $\psi(\mathbf{x})$ to be the last $\psi(\mathbf{x},t_K)$ out of the loop.

In fact, it is exactly a discretization of the equivalent minimization problem (2.7). In its key iterating step.

$$\min(\psi(\mathbf{x}-\tau\mathbf{v},t_{k-1}),\psi(\mathbf{x},t_{k-1})) = \min(\phi(\mathbf{x}-t_k\mathbf{v}),\psi(\mathbf{x},t_{k-1})), \tag{2.8}$$

where the right-hand side is a typical and straight forward iterating step to look for minimum along the ray tracing back from $\mathbf{x}$ to the eye-point $\mathbf{x}^*$.

In summary, we have set up the visibility problem through help of level set functions, and constructed it in the framework of a transport equation. Modifications are added to the equation to fully represent the nature of the problem, and it is ultimately reformulated as a minimization problem on each individual point of interest.

2.2 Ray Tracing and Phase Flow Method

Based on the problem setup, visibility function ψ could be evaluated point by point individually on the domain, as a minimization of values of ϕ along the ray tracing back from $\mathbf{x}$ to the eye-point $\mathbf{x}^*$. To get full visibility information on the entire domain, Algorithm 1 is put forward in response.

Algorithm 1: algorithm for finding visibility boundary: trivial

```
1 function visibility (φ, x*);
  Input  : a level set function φ representing obstacles, and an eye-point location x*.
  Output : a visibility function ψ representing the visibility boundary.
2 Initialize with (2.3) and discretize time t with step size τ;
3 for each x on domain do
4 |   while x − t_k v does not pass beyond x* do
5 |   |   ψ(x, t_k) = min(φ(x − t_k v), ψ(x, t_{k−1}));
6 |   end
7 end
```

As a remark, the condition "$\mathbf{x} - t_k\mathbf{v}$ does not pass beyond $\mathbf{x}^*$" can be verified by checking if $t_k < ||\mathbf{x} - \mathbf{x}^*||$. In most cases, the initial ϕ is only provided on a gridded domain, as in the case of a numerical signed distance function efficiently computed on the grid using the fast marching method [30] or fast sweeping method [29]. Consequently, all evaluations of ϕ along the eye rays must be approximated using interpolation schemes, such as bilinear interpolation in the two-dimensional scenario.

Algorithm 1 is showcased as transparently as it can be, with an outer loop throughout each point of interest, and utilizing the idea from (2.8). Line 5 of the algorithm shows independence of points in the evaluations of ψ's, since new evaluations on $\mathbf{x}$ are not influenced by any previous

ψ calculations on other points. Henceforth, parallel computing could be introduced in the system to take advantage of the independence of points to execute the inner loop concurrently.

It is worth trying to rearrange the nested loop for further improvement of the algorithm, considering that the inner loop in time is basically making use of locations closer to the observer on the ray tracing back to the eye-point. There is more flexibility to utilize existing information on the rays tracing back when location is the inner loop. Thus, the algorithm is rewritten as Algorithm 2, and referred to as the ray tracing method from now on.

Algorithm 2: algorithm for finding visibility boundary: ray tracing

```
1  function visibility (φ, x*);
   Input  : a level set function φ representing obstacles, and an eye-point location x*.
   Output : a visibility function ψ representing the visibility boundary.
2  Initialize with (2.3) and discretize time t with step size τ;
3  while t_k < T do
4      for each x on domain do
5          if x − t_k v does not pass beyond x* then
6              ψ(x, t_k) = min(ψ(x − τv, t_{k−1})), ψ(x, t_{k−1}));
7          else
8              ψ(x, t_k) = ψ(x, t_{k−1});
9          end
10     end
11 end
```

In practice, the threshold T could be set as any number greater than $\max_{\mathbf{x}\in D} ||\mathbf{x}-\mathbf{x}^*||$ where D is the domain, so that all ψ's are finalized upon termination. When the input ϕ is provided only on a gridded domain, all evaluations of $\psi(\mathbf{x},t)$ on non-gridpoint shall be approximated by interpolation schemes.

Notice that, line 6 from Algorithm 2 comprehensively reflects the essential idea of the algorithm, solving the transport equation and minimizing in one time step. Compared to line 5 from Algorithm 1, it is more fundamental, making it easier to adjust for further improvements. The algorithm will produce a class of intermediate maps $\psi(\mathbf{x},t_k)$'s with an increment of τ in time, evaluated on the whole domain. In each t_k loop, evaluating the map involves solving the equation (2.4) with multiple initial conditions. Henceforth, the phase flow method is brought

forward to help.

Phase flow method [35] was first introduced as a fast and accurate approach to compute numerical solutions to non-linear autonomous ordinary differential equations of the following form

$$\frac{d\mathbf{y}}{dt} = F(\mathbf{y}), \quad t > 0, \tag{2.9}$$

where $\mathbf{y} : \mathbb{R} \to \mathbb{R}^n$ and $F : \mathbb{R}^n \to \mathbb{R}^n$ is a smooth multivariate function not dependent on time t. Given an initial condition $\mathbf{y}(0) = \mathbf{y}_0$ and a target time T, the basic initial value problem is to integrate the system and compute $\mathbf{y}(\mathbf{y}_0, T)$, which indicates the position of a particle at time T starting from position $\mathbf{y}_0$.

In many situations, one needs to solve (2.9) with multiple initial conditions. The standard approach is to solve each initial value problem independently for $\mathbf{y}(\mathbf{y}_0, T)$, selecting a time step τ, and recursively applying a local integration rule (such as the explicit Euler method or widely-used fourth order Runge-Kutta method) at discrete times $\tau, 2\tau, 3\tau, ..., T$ for each initial condition $\mathbf{y}_0$. It may be extremely costly considering the amount of integrations needed on the time steps in every initial value case. Phase flow method is designed to bypass the complexity caused by the independent but repetitive procedures.

For a fixed time t, a phase map $g_t : \mathbb{R}^n \to \mathbb{R}^n$ is defined as a map on initial value $\mathbf{y}_0$, that is $g_t(\mathbf{y}_0) = \mathbf{y}(\mathbf{y}_0, t)$. The family of all phase maps $\{g_t, t \in \mathbb{R}^+\}$ is called the phase flow. The phase flow has a one parameter group structure, $g_{t'} \circ g_t = g_{t'+t}$ which is important in the study of autonomous ODEs. Compare to the standard approach, the phase flow method aims to construct the complete phase map g_T at time T. It operates by initially constructing a phase map for a small time using a standard ODE integrator and builds up the phase map for larger times with the help of a local interpolation scheme together with the group property of the phase flow. The key idea is basically listed in the three steps below.

1. Discretization: Select a small time step $\tau > 0$ such that $T = 2^s\tau, s \in \mathbb{Z}^+$, and a grid size $h > 0$ to make uniform or quasi-uniform grid on the domain.

2. Initialization: Compute an approximate phase map $\tilde{g}_\tau$ at time τ. At grid point $\mathbf{y}_0$, $\tilde{g}_\tau(\mathbf{y}_0)$ is computed by applying a standard ODE integration rule with a single time step of length τ. At any other point, $\tilde{g}_\tau$ is evaluated via local interpolation.

3. Loop: For $k = 1, ..., s$, construct $\tilde{g}_{2^k\tau}$ using the group relation $\tilde{g}_{2^k\tau} = \tilde{g}_{2^{k-1}\tau} \circ \tilde{g}_{2^{k-1}\tau}$. At grid point $\mathbf{y}_0$, $\tilde{g}_{2^k\tau}(\mathbf{y}_0) = \tilde{g}_{2^{k-1}\tau}(\tilde{g}_{2^{k-1}\tau}(\mathbf{y}_0))$. At any other point, $\tilde{g}_{2^k\tau}$ is evaluated via local interpolation. At the end of the loop, $\tilde{g}_T$ is the approximate phase map at time T.

In fact, the phase flow algorithm in theory keeps the number of stages s as a constant, and thus may adopt powers with base number $B \in \mathbb{Z}^+$ other than 2. The group relation in the loop then looks like $g_{B^k\tau} = \underbrace{g_{B^{k-1}\tau} \circ g_{B^{k-1}\tau} \circ ... \circ g_{B^{k-1}\tau}}_{B \text{ times}}$. The "time doubling" idea shown above with fixed base 2, in contrary, is actually not as accurate. Yet it reduces the complexity even more, by a log factor in time, compared to the standard approach. So it is very typical and useful for practical purposes, and we still refer to it as the phase flow method.

Below is a comparison of steps in the standard approach versus phase flow approach in solving the (2.9) ODE. Note that, the critical observation that drives the idea of phase flow method is the breakdown of time T into smaller steps, based on the one parameter group structure of the operator family $\{g_t, t \in \mathbb{R}^+\}$. The "divide and conquer" paradigm behind makes it easy to generate to other scenarios.

$g_\tau(\mathbf{y}_0)$: τ integration
$g_{2\tau}(\mathbf{y}_0) = g_\tau(g_\tau(\mathbf{y}_0))$: τ integration
$g_{3\tau}(\mathbf{y}_0) = g_\tau(g_{2\tau}(\mathbf{y}_0))$: τ integration
$g_{4\tau}(\mathbf{y}_0) = g_\tau(g_{3\tau}(\mathbf{y}_0))$: τ integration
... **Standard**

$g_\tau(\mathbf{y}_0)$: τ integration
$g_{2\tau}(\mathbf{y}_0) = g_\tau(g_\tau(\mathbf{y}_0))$: $g_\tau$interpolation
$g_{4\tau}(\mathbf{y}_0) = g_{2\tau}(g_{2\tau}(\mathbf{y}_0))$: $g_{2\tau}$interpolation
$g_{8\tau}(\mathbf{y}_0) = g_{4\tau}(g_{4\tau}(\mathbf{y}_0))$: $g_{4\tau}$interpolation
... **Phase Flow with B=2**

To solve visibility problem, as discussed in the previous section, when we go "backward"

along characteristics curve until initial time 0, we are actually solving an autonomous ODE $\frac{d\mathbf{x}(t)}{dt} = -\mathbf{v}(\mathbf{x})$, with $\mathbf{v}(\mathbf{x}) = \frac{\mathbf{x}-\mathbf{x}^*}{||\mathbf{x}-\mathbf{x}^*||}$ considering the straight-line propagation of light. Hence, notations from the previous discussion of phase flow method can be adapted for use here. At any fixed time t, denote the phase map $g_t : \mathbb{R}^d \to \mathbb{R}^d$ as $g_t(\mathbf{x}_0) = \mathbf{x}(\mathbf{x}_0, t)$, representing the "backward" characteristics curve location at time t starting from $\mathbf{x}_0$. In our case, $g_t(\mathbf{x}) = \mathbf{x} - t\mathbf{v}$, with $\mathbf{v}$ from (2.1). Additionally, for convenience in notation, use $\psi_t(\mathbf{x})$ to denote $\psi(\mathbf{x}, t)$, meaning the minimum of ϕ along the characteristics curve segment from $\mathbf{x}$ to $g_t(\mathbf{x})$, that is $L(\mathbf{x} - t\mathbf{v}, \mathbf{x})$ in our case according to (2.6). So $\psi_t(\mathbf{x}) = \min_{[0,t]} \phi(\mathbf{x}(t))$.

Thanks to the following property of minimization, for any trajectory $\mathbf{x}(t) : \mathbb{R} \to \mathbb{R}^n$ and any function $f : \mathbb{R}^n \to \mathbb{R}$,

$$\min_{[0,t+t']} f(\mathbf{x}(t)) = \min\left(\min_{[0,t]} f(\mathbf{x}(t)), \min_{[t',t+t']} f(\mathbf{x}(t))\right), \text{ when } t' \leq t, \tag{2.10}$$

hence the $\{\psi_t\}$ operator family exhibits a recursive structure as below

$$\psi_{t+t'}(\mathbf{x}) = \min\left(\psi_t(\mathbf{x}), \psi_t(g_{t'}(\mathbf{x}))\right), \text{when } t' \leq t.$$

Such structure supports the break down of time T into smaller time pieces, with just an evolution in location, which is analytically available and easily accessible. In light of this, insights from the phase flow method could be transferable to improve the ray tracing method. To illustrate this, the following presents a comparison of steps in the standard ray tracing versus ray tracing with

the phase flow idea in solving the visibility problem.

$\psi_\tau(\mathbf{x})$: τ minimization
$\psi_{2\tau}(\mathbf{x}) = \min(\psi_\tau(\mathbf{x}), \psi_\tau(g_\tau(\mathbf{x})))$
$\psi_{3\tau}(\mathbf{x}) = \min(\psi_{2\tau}(\mathbf{x}), \psi_{2\tau}(g_\tau(\mathbf{x})))$
$\psi_{4\tau}(\mathbf{x}) = \min(\psi_{3\tau}(\mathbf{x}), \psi_{3\tau}(g_\tau(\mathbf{x})))$
... **ray tracing**

$\psi_\tau(\mathbf{x})$: τ minimization
$\psi_{2\tau}(\mathbf{x}) = \min(\psi_\tau(\mathbf{x}), \psi_\tau(g_\tau(\mathbf{x})))$
$\psi_{4\tau}(\mathbf{x}) = \min(\psi_{2\tau}(\mathbf{x}), \psi_{2\tau}(g_{2\tau}(\mathbf{x})))$
$\psi_{8\tau}(\mathbf{x}) = \min(\psi_{4\tau}(\mathbf{x}), \psi_{4\tau}(g_{4\tau}(\mathbf{x})))$
... **ray tracing with phase flow**

Note that, all the reasoning above to deduce an integrated algorithm only requires $\mathbf{v}$ in the transport equation (2.2) independent of time t. If $\mathbf{v}(\mathbf{x})$ differs from the straight-line propagation form as in (2.1), meaning the sunlight travels in a non-direct path, possibly influenced by refraction, reflection, or diffraction, etc., the phase flow method for ODE could be directly applied to calculate numerical $g_t(\mathbf{x})$, and it would fit perfectly in the structure of the new algorithm. Fortunately, when only straight-line propagation of light is considered, as the main focus our research, $g_t(\mathbf{x})$ is analytically available. Accordingly, the integrated algorithm is presented below in Algorithm 3, referred to as the ray tracing method with phase flow.

It is important to note that the key step utilizing the phase flow idea is in line 13. Each loop of evaluations on $\mathbf{x}$ requires local interpolation based on the previous phase map. Compared to the previous ray tracing algorithm, the number of phase maps evaluated on the computational domain is drastically decreased by a log factor, from $\{\psi_\tau, \psi_{2\tau}, \psi_{3\tau}, ..., \psi_{k\tau}, ..., \psi_T\}$ to $\{\psi_\tau, \psi_{2\tau}, \psi_{2^2\tau}, ..., \psi_{2^s\tau}, ..., \psi_T\}$.

The algorithm 3 provided above adopts the "time doubling" scheme with fixed base number B=2. It is easy to generate it to other $B \in \mathbb{Z}^+$ based on the fact that

$$\psi_{Bt}(\mathbf{x}) = \min\left(\underbrace{\psi_t(\mathbf{x}), \psi_t(g_t(\mathbf{x})), ..., \psi_t(g_{(B-1)t}(\mathbf{x}))}_{B \text{ terms}}\right),$$

Algorithm 3: algorithm for finding visibility boundary: ray tracing with phase flow

function visibility $(\phi, \mathbf{x}^*)$;

Input : a level set function ϕ on a uniform grid representing obstacles, and an eye-point location $\mathbf{x}^*$.

Output : a visibility function ψ on the grid representing the visibility boundary.

Discretize time t with step size τ, and introduce $s \in \mathbb{Z}^+$ representing levels in time;
for *each* $\mathbf{x}$ *on domain* **do**
    if $\mathbf{x} - \tau\mathbf{v}$ *does not pass beyond* $\mathbf{x}^*$ **then**
        $\psi(\mathbf{x}, \tau) = \min(\phi(\mathbf{x} - \tau\mathbf{v}), \phi(\mathbf{x}))$;
    else
        $\psi(\mathbf{x}, \tau) = \min(\phi(\mathbf{x}^*), \phi(\mathbf{x}))$;
    end
end
while $2^s\tau < T$ **do**
    for *each* $\mathbf{x}$ *on domain* **do**
        if $\mathbf{x} - 2^{s-1}\tau\mathbf{v}$ *does not pass beyond* $\mathbf{x}^*$ **then**
            $\psi(\mathbf{x}, 2^s\tau) = \min(\psi(\mathbf{x} - 2^{s-1}\tau\mathbf{v}, 2^{s-1}\tau), \psi(\mathbf{x}, 2^{s-1}\tau))$;
        else
            $\psi(\mathbf{x}, 2^s\tau) = \psi(\mathbf{x}, 2^{s-1}\tau)$;
        end
    end
end

resulting from utilizing (2.10) in an iterative manner. Below, a more general ray tracing algorithm with phase flow is outlined in the three steps, in a format consistent with the phase flow method.

1. Discretization: Select a small time step $\tau > 0$ and an base constant $B \in \mathbb{Z}^+$, such that $T = B^S\tau, S \in \mathbb{Z}^+$. Select a grid size $h > 0$ to make uniform or quasi-uniform grid on the domain.

2. Initialization: Compute an approximate phase map $\tilde{\psi}_\tau$ at time τ. At grid point $\mathbf{x}_0$, $\tilde{\psi}_\tau(\mathbf{x}_0)$ is computed by minimum of $\phi(\mathbf{x}_0 - \tau\mathbf{v})$ and $\phi(\mathbf{x}_0)$. At any other point, $\tilde{\psi}_\tau$ is evaluated via local interpolation.

3. Loop: For $k = 1, ..., S$, construct $\tilde{\psi}_{B^k\tau}$ using $\tilde{\psi}_{B^{k-1}\tau}$. At grid point $\mathbf{x}_0$,

$$\tilde{\psi}_{B^k\tau}(\mathbf{x}_0) = \min\Big(\tilde{\psi}_{B^{k-1}\tau}(\mathbf{x}_0), \tilde{\psi}_{B^{k-1}\tau}(\mathbf{x}_0 - B^{k-1}\tau\mathbf{v}), \tilde{\psi}_{B^{k-1}\tau}(\mathbf{x}_0 - 2B^{k-1}\tau\mathbf{v}), ..., \tilde{\psi}_{B^{k-1}\tau}(\mathbf{x}_0 - (B-1)B^{k-1}\tau\mathbf{v})\Big).$$

   At any other point, $\tilde{\psi}_{B^k\tau}$ is evaluated via local interpolation. At the end of the loop, $\tilde{\psi}_T$ is the approximate phase map at time T.

An illustration of operations needed in one dimension in cases of $B = 2$ and $B = 3$ is provided in the Figure 2.2. To cover all the 11 grid points, the stages needed are 4 and 3 respectively. Each black dot represents a stored data of the approximate phase map at a certain stage. Dots from higher stages are determined by dots from lower stages through connecting lines.

For ease of reference, $\tilde{\psi}_{B^k\tau}$, $k = 0, 1, ..., S$ can be written as $\tilde{\psi}_{k+1}$ to indicate our approximation at level $k+1$ when the context of base number B is clear. The following proposition guarantees the accuracy and complexity of our ray tracing method with phase flow.

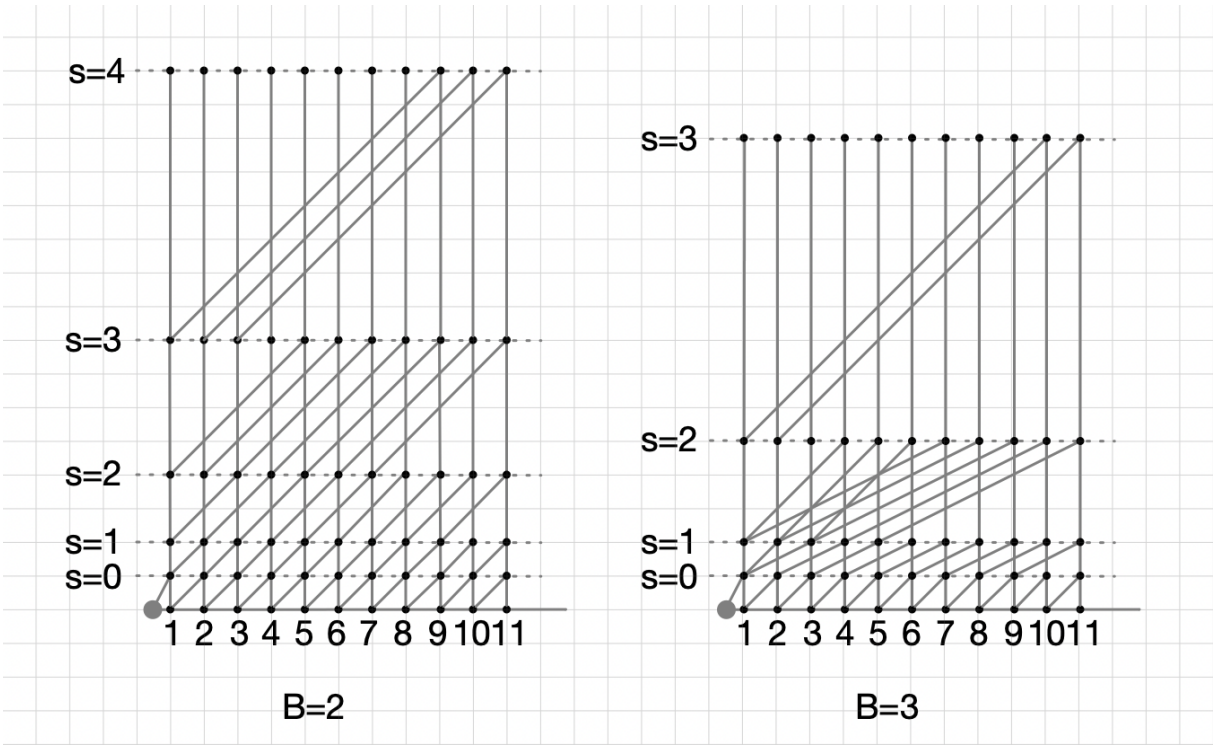


Figure 2.2. Illustration of operations of Algorithm 3 in one dimension.

Proposition 2.2.1. *Suppose that the local interpolation scheme is multi-linear interpolation. Define the approximation error at level l by*

$$\varepsilon_l = \max_{\mathbf{x}\in D_h} |\tilde{\psi}_l(\mathbf{x}) - \psi_l(\mathbf{x})|. \tag{2.11}$$

Then Algorithm 3 enjoys the following properties:

1. *The approximation error obeys*

$$\varepsilon_{S+1} \leq C \cdot ((S+1)h^2 + \tau^2) \tag{2.12}$$

for some positive $C > 0$. When B is set to constant, the accuracy is $O(h^2 \cdot \log(\tau^{-1}) + \tau^2)$; when S is set to constant, the accuracy is $O(h^2 + \tau^2)$.

Lemma 2.2.1. *Given two sets of unsorted numbers of same size* $\{x_1, x_2, ..., x_n\}$ *and* $\{y_1, y_2, ..., y_n\}$.

$$|\min_i x_i - \min_j y_j| \leq \max_k |x_k - y_k|, \quad i, j, k \in \{1, 2, ..., n\}.$$

Proof. Suppose $x_{im} = \min_i x_i$ and $y_{jm} = \min_j y_j$.

When $x_{im} \leq y_{jm}$,

$$|\min_i x_i - \min_j y_j| = \min_j y_j - x_{im} \leq y_{im} - x_{im} \leq \max_k |x_k - y_k|.$$

When $x_{im} > y_{jm}$,

$$|\min_i x_i - \min_j y_j| = \min_i x_i - y_{jm} \leq x_{jm} - y_{jm} \leq \max_k |x_k - y_k|.$$

□

Proof of Proposition 2.2.1. We begin by proving claim 1. Below are some notation setups and observations:

- Let d be the dimension of the ambient space;
- Let $\mathbf{x}_i$ denote grid points of the discretized computational domain D_h.
- For function $f(\mathbf{x}) : \mathbb{R}^d \to \mathbb{R}$, let f_i denote its value on grid point $\mathbf{x}_i$. Let I_f denote the multi-linear interpolant of f based on f_i. There are some useful properties of the multi-linear interpolation operator I.
  - For any f,

$$\min_i f_i \leq I_f(\mathbf{x}) \leq \max_i f_i, \qquad |I_f(\mathbf{x})| \leq \max_i |f_i|, \qquad \mathbf{x} \in D. \tag{2.13}$$

– For any f and g,

$$I_{f+g}(\mathbf{x}) = I_f(\mathbf{x}) + I_g(\mathbf{x}). \tag{2.14}$$

– Suppose ∇f is Lipschitz continuous, then the interpolation error at any point $\mathbf{x} \in D$

$$|I_f(\mathbf{x}) - f(\mathbf{x})| \leq C_f h^2, \tag{2.15}$$

for some positive constant C_f.

The following notations only applies to the case when $B = 2$. All other cases will follow similar reasoning.

- Let $\mathbf{y}_{i,l}$ denote points utilized in evaluating $\tilde{\psi}_l(\mathbf{x}_i)$ at level $l = 1, 2, ..., S+1$:

$$\mathbf{y}_{i,l} = \begin{cases} \mathbf{x}_i - \tau \mathbf{v}_i, & \text{if } l = 1, \\ \mathbf{x}_i - 2^{l-2}\tau \mathbf{v}_i, & \text{if } l \geq 2,\ 2^{l-2}\tau \leq ||\mathbf{x}_i - \mathbf{x}^*||, \\ \mathbf{x}^*, & \text{otherwise,} \end{cases}$$

where $\mathbf{v}_i = \frac{\mathbf{x}_i - \mathbf{x}^*}{||\mathbf{x}_i - \mathbf{x}^*||}$. The notation can be generalized to any points $\mathbf{x} \in D$ as $\mathbf{y}_l(\mathbf{x})$, when $\mathbf{x}_i$ is replaced by $\mathbf{x}$, and likewise, $\mathbf{v}_i$ by $\mathbf{v}(\mathbf{x})$ as in (2.1).

- $\tilde{\psi}_l(\mathbf{x}_i)$ denotes our approximate visibility level-set function from Algorithm 3 at level l evaluated on grid point x_i:

$$\tilde{\psi}_l(\mathbf{x}_i) = \begin{cases} \phi(\mathbf{x}_i), & \text{if } l = 0, \\ \min\left(\tilde{\psi}_{l-1}(\mathbf{x}_i),\ I_{\tilde{\psi}_{l-1}}(\mathbf{y}_{i,l})\right), & \text{if } l = 1, 2, ..., S+1. \end{cases} \tag{2.16}$$

The notation can be generalized to any points $\mathbf{x} \in D$ as $\tilde{\psi}_l(\mathbf{x})$, when $\mathbf{x}_i$ is replaced by $\mathbf{x}$, and likewise, $\mathbf{y}_{i,l}$ by $\mathbf{y}(\mathbf{x})$.

- Let $\hat{\psi}_l(\mathbf{x}_i)$ denote the "ideal" visibility level-set function at level l evaluated on grid point x_i, if no interpolation is needed:

$$\hat{\psi}_l(\mathbf{x}_i) = \begin{cases} \phi(\mathbf{x}_i), & \text{if } l = 0, \\ \hat{\psi}_l(\mathbf{x}_i) = \min\left(\hat{\psi}_{l-1}(\mathbf{x}_i),\ \hat{\psi}_{l-1}(\mathbf{y}_{i,l})\right), & \text{if } l = 1,2,...,S+1. \end{cases} \tag{2.17}$$

  The notation can be generalized to any points $\mathbf{x} \in D$ as $\hat{\psi}_l(\mathbf{x})$, when $\mathbf{x}_i$ is replaced by $\mathbf{x}$, and likewise, $\mathbf{y}_{i,l}$ by $\mathbf{y}(\mathbf{x})$.

- Let $\psi_l(\mathbf{x})$ denote the exact visibility level-set solution at level l according to previous simplified notations and (2.6), that is

$$\psi_l(\mathbf{x}) = \begin{cases} \phi(\mathbf{x}), & \text{if } l = 0, \\ \min_{L(\mathbf{x}-2^{l-1}\tau\mathbf{v},\mathbf{x})} \phi(\mathbf{x}), & \text{if } l = 1,2,...,S+1. \end{cases}$$

  where $\mathbf{v}$ is short for $\mathbf{v}(\mathbf{x})$ as in (2.1).

  Since

$$L(\mathbf{x}-2^{l-1}\tau\mathbf{v},\mathbf{x}) = L(\mathbf{x}-2^{l-2}\tau\mathbf{v},\mathbf{x}) \cup L(\mathbf{x}-2^{l-1}\tau\mathbf{v},\mathbf{x}-2^{l-2}\tau\mathbf{v}), \quad l \geq 2,$$

$$\psi_l(\mathbf{x}) = \begin{cases} \phi(\mathbf{x}), & \text{if } l = 0, \\ \min_{L(\mathbf{x}-\tau\mathbf{v},\mathbf{x})} \phi(\mathbf{x}), & \text{if } l = 1, \\ \min\left(\psi_{l-1}(\mathbf{x}),\ \psi_{l-1}(\mathbf{y}_l(\mathbf{x}))\right), & \text{if } l = 2,...,S+1. \end{cases} \tag{2.18}$$

- Let $E^I_{i,l}$ denote the error introduced by local interpolations in the algorithm, evaluated at $\mathbf{x}_i$,

$$E^I_{i,l} = \tilde{\psi}_l(\mathbf{x}_i) - \hat{\psi}_l(\mathbf{x}_i).$$

  The notation can be generalized to any points $\mathbf{x} \in D$ as $E^I_l(\mathbf{x})$, when $\mathbf{x}_i$ is replaced by $\mathbf{x}$.

- Let $E_{i,l}^M$ denote the error introduced by discretized minimization in the initialization step of the algorithm, evaluated at $\mathbf{x}_i$,

$$E_{i,l}^M = \hat{\psi}_l(\mathbf{x}_i) - \psi_l(\mathbf{x}_i).$$

The notation can be generalized to any points $\mathbf{x} \in D$ as $E_l^M(\mathbf{x})$, when $\mathbf{x}_i$ is replaced by $\mathbf{x}$.

- Let $E_{i,l}$ denote the error of the algorithm, evaluated at $\mathbf{x}_i$,

$$E_{i,l} = \tilde{\psi}_l(\mathbf{x}_i) - \psi_l(\mathbf{x}_i) = E_{i,l}^I + E_{i,l}^M.$$

Then

$$\varepsilon_l = \max_i |E_{i,l}| \leq \max_i |E_{i,l}^I| + \max_i |E_{i,l}^M|.$$

We claim that there exist constants $C_1 > 0, C_2 > 0$ large enough, such that for any $l = 0, 1, 2, ..., S+1$,

$$|E_{i,l}^I| \leq C_1 l h^2. \tag{2.19}$$

$$|E_{i,l}^M| \leq C_2 \tau^2. \tag{2.20}$$

Proof of (2.19) is done by induction on l:

i) When $l = 0$, $|E_{i,0}^I| = 0$.

ii) When $l \geq 1$, assume $|E_{i,l-1}^I| \leq C_1(l-1)h^2$. Based on (2.16) and (2.17),

$$|E_{i,l}^I| = |\min\left(\tilde{\psi}_{l-1}(\mathbf{x}_i),\ I_{\tilde{\psi}_{l-1}}(\mathbf{y}_{i,l})\right) - \min\left(\hat{\psi}_{l-1}(\mathbf{x}_i),\ \hat{\psi}_{l-1}(\mathbf{y}_{i,l})\right)|$$

By Lemma (2.2.1), $$\leq \max\left(|\tilde{\psi}_{l-1}(\mathbf{x}_i) - \hat{\psi}_{l-1}(\mathbf{x}_i)|,\ |I_{\tilde{\psi}_{l-1}}(\mathbf{y}_{i,l}) - \hat{\psi}_{l-1}(\mathbf{y}_{i,l})|\right).$$

The first term is $|E_{i,l-1}^I|$, upper bounded by $C_1(l-1)h^2$. For the second term, when $\mathbf{z} \in D$,

$$
\begin{aligned}
|I_{\tilde{\psi}_{l-1}}(\mathbf{z}) - \hat{\psi}_{l-1}(\mathbf{z})| &= |I_{\hat{\psi}_{l-1}+E^I_{l-1}}(\mathbf{z}) - \hat{\psi}_{l-1}(\mathbf{z})| \\
\text{By(2.14),} \qquad &= |I_{E^I_{l-1}}(\mathbf{z}) + I_{\hat{\psi}_{l-1}}(\mathbf{z}) - \hat{\psi}_{l-1}(\mathbf{z})| \\
&\leq |I_{E^I_{l-1}}(\mathbf{z})| + |I_{\hat{\psi}_{l-1}}(\mathbf{z}) - \hat{\psi}_{l-1}(\mathbf{z})| \\
\text{By(2.13),} \qquad &\leq \max_i |E^I_{i,l-1}| + |I_{\hat{\psi}_{l-1}}(\mathbf{z}) - \hat{\psi}_{l-1}(\mathbf{z})| \\
\text{By(2.15),} \qquad &\leq C_1(l-1)h^2 + C_{\hat{\psi}_{l-1}}h^2 \\
&\leq C_1 l h^2
\end{aligned}
$$

if $C_{\hat{\psi}_{l-1}} \leq C_1$. Therefore, the second term $|I_{\tilde{\psi}_{l-1}}(\mathbf{y}_{i,l}) - \hat{\psi}_{l-1}(\mathbf{y}_{i,l})| \leq C_1 l h^2$.

Altogether, $|E^I_{i,l}| \leq C_1 l h^2$. Therefore, (2.19) is proved for all $l = 0, 1, 2, ..., S+1$.

Proof of (2.20) is done by induction on l as well:

i) When $l = 0$, $|E^M_{i,0}| = 0$.

ii) When $l \geq 1$, compare $\hat{\psi}_l(\mathbf{x})$ and $\psi_l(\mathbf{x})$ from (2.17) and (2.18). The major difference results from its initial level $l = 1$, where we have

$$
E^M_1(\mathbf{x}) = \min(\phi(\mathbf{x}), \phi(\mathbf{x} - \tau\mathbf{v})) - \min_{L(\mathbf{x}-t\mathbf{v},\mathbf{x})} \phi(\mathbf{x}).
$$

Suppose $\nabla\phi$ is Lipschitz continuous, so is its directional derivative on $L(\mathbf{x} - t\mathbf{v}, \mathbf{x})$. With the one dimensional Taylor's Theorem, there exists constant $C_2 > 0$ such that

$$
|E^M_1(\mathbf{x})| \leq C_2\tau^2, \ \forall \mathbf{x} \in D.
$$

In the case $\mathbf{x} \in D_h$, it proves the base step $|E^M_{i,1}| \leq C_2\tau^2$ for the following induction.

iii) When $l \geq 2$, assume $|E^M_{i,l-1}| \leq C_2\tau^2$. Based on (2.17) and (2.18),

$$\begin{aligned} |E^M_{i,l}| &= |\min\left(\hat{\psi}_{l-1}(\mathbf{x}_i),\ \hat{\psi}_{l-1}(\mathbf{y}_{i,l})\right) - \min\left(\psi_{l-1}(\mathbf{x}_i),\ \psi_{l-1}(\mathbf{y}_{i,l})\right)| \\ \text{By Lemma (2.2.1),} \quad &\leq \max\left(|\hat{\psi}_{l-1}(\mathbf{x}_i) - \psi_{l-1}(\mathbf{x}_i)|,\ |\hat{\psi}_{l-1}(\mathbf{y}_{i,l}) - \psi_{l-1}(\mathbf{y}_{i,l})|\right) \\ &\leq \max\left(|E^M_{i,l-1}|,\ |E^M_1(\mathbf{y}_{i,l})|\right) \\ &\leq C_2\tau^2. \end{aligned}$$

Therefore, (2.20) is proved for all $l = 0, 1, 2, ..., S+1$.

Combining with all previous reasoning, we see that

$$\varepsilon_l \leq C_1 l h^2 + C_2\tau^2.$$

As a remark for base number B other than 2, the only difference is that all the minimization in the loop would be on B numbers instead of 2, where the B evaluations are from B number of $\mathbf{y}_l(\mathbf{x})$'s for each point.

We now prove claim 2. The gridded domain has $O(h^{-d})$ number of grid points. The one-step initialization of the algorithm takes a constant number of operations per grid point and is thus of the order of $O(h^{-d})$. As we have seen, the construction of $\tilde{\psi}_T$ is divided into S stages, and each stage requires applying the repetitive operations (minimization and interpolation) B times. Therefore, the total complexity is $O(B \cdot S \cdot h^{-d})$. □

2.3 Seeding and Growing Method

Algorithms from the previous sections evaluate the visibility function ψ on the whole computational domain to find the visible/ invisible region, causing an excess of redundant data at grid points far from the visibility interface. To further alleviate the problem's complexity, we are concentrating specifically on grid points next to the visibility interface. A key observation is

that, the visibility interface of any connected obstacle is always continuous. Therefore, given a connected obstacle region, assuming the availability of a "seed" cell crossing its visibility boundary, we can "grow" a queue of candidate cross-boundary cells, and thus only evaluate ψ on a limited grid points.

We call the procedure to prepare for the "seed" cell the seeding process. It looks for a grid cell crossing the visibility interface, based on the obstacle level set function ϕ. The seeding process then provides a feed to launch the succeeding growing method. Notice that, among all the grid cells crossing the obstacle boundary, the one closest to the eye-point has to be on the visibility boundary as well. Henceforth, the seeding process basically contains two steps: finding all grid cells that cross the obstacle boundary, and identifying the cell with minimum distance to the eye-point. The former step traverses all grid cells, and check if all vertices of the grid cell share the same ϕ sign. In the d dimensional discretized computational domain D_h, its complexity is $(2^d - 1)h^{-d}$. The later step can be realized by building a min-heap in $O(h^{-(d-1)})$ complexity.

The following procedure that "grows" a set of candidate cross-boundary cells from the "seed" is called the growing process. It is implemented in a queue structure, initiated with a seed provided by the seeding process. ψ is evaluated at all vertices of the top cell from the queue. Taking advantage of continuity of the visibility function, if there is a change in sign of ψ values among vertices on a face, it means the visibility boundary cuts through that face, and the queue "grows" by appending a neighboring cell sharing the interface-cutting face. After checking all faces of the top cell, it is popped out, and the procedure loop through the queue until it is empty.

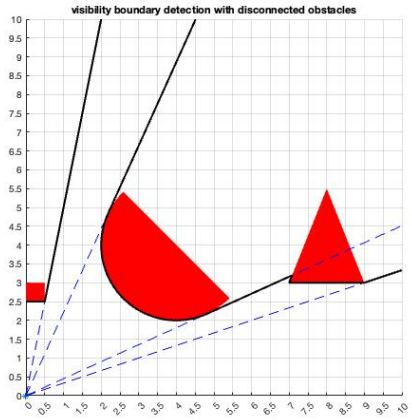
When there are several disconnected obstacles, as illustrated in the Figure 2.3, the seeding process should be able to resume after the growing method scan through a connected visible region. To fulfill this, we maintain the ordering provided by the min-heap from the seeding process, and nest the growing process in a seeding loop. Pop out the root cell from the min heap, if it is not yet marked as read from the queue, and it is not completely blocked, then it could serve as a seed to initiate the next round of growing process. The Seeding-Growing loop is then iterated until the heap is empty. Algorithm 4 is presented below to emphasize the structure

used in locating the grid points needed for evaluations. The algorithm is mentioned later in the dissertation as the seeding and growing method.

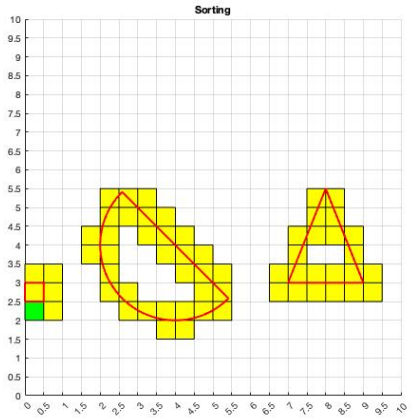
Algorithm 4: algorithm for finding visibility boundary: seeding and growing method

```
1  function visibility (φ, x*);
   Input  : a level set function φ on a uniform grid representing obstacles, and an
            eye-point location x*.
   Output : visibility function ψ values on certain grid points next to the visibility
            boundary.
2  min-heapify all grid cells crossing the boundary of obstacles according to distance to
     eye-point;
3  while heap is not empty do
4      get the root cell;
5      evaluate ψ at its vertices;
6      if it is not once_in_queue and it is not fully invisible then
7          make it a seed;
8          add the seed cell to the queue and mark it once_in_queue;
9          while queue is not empty do
10             get the top cell;
11             evaluate ψ at its vertices;
12             for each neighboring cell do
13                 if it is not once_in_queue and the shared face cuts through the
                     interface then
14                     add it to the queue, and mark it once_in_queue;
15                 end
16             end
17             pop out the top cell;
18         end
19     end
20     remove the root cell and re-balance the heap;
21 end
```

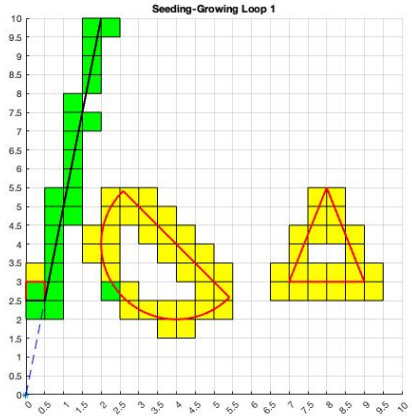
In Algorithm 4, the method used to evaluate ψ is not specified. The seeding and growing method can actually work together with each of the three algorithms introduced before with some modifications. For example, in the case of ray tracing method, Algorithm 2 can be adapted for recursion at specified grid points and utilized in Algorithm 4 to evaluate ψ. In comparison, the composite algorithm does sacrifice some complexity to maintain a min-heap for the cross-obstacle-boundary cells and a queue for the cross-visibility-boundary cells, but it dramatically



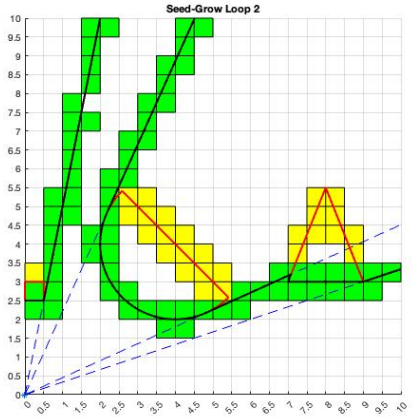
(a) A show case with disconnected obstacles. Red fill color means the interior of obstacles. Black solid lines represent the actual visibility interface.



(b) All grid cells crossing the boundary of obstacles are highlighted and put in the min-heap. The one cell in green is a seed.



(c) The one seed grows to a connected visibility boundary in green cells. The outlying cell in green is the next seed.



(d) All cells in green are the ones cross the visibility boundary (once_in_queue). The cells in yellow are auxiliary ones from the seeding process, but never used.

Figure 2.3. Illustration of the seeding and growing process in 2 dimension.

reduces the number of ψ evaluations: from the scale of the whole computational domain, to the size of the visibility interface, that is a $O(h^{-1})$ factor. Taking the advantage of ray tracing method with phase flow as in Algorithm 3, Algorithm 4 would further cut down the operations needed per ψ evaluation at a grid point.

2.4 Tube and Cube Structure

The function ϕ, which represents the obstacle, usually accentuates its zero-level set by avoiding excessive oscillations far from it. It is commonly observed that at grid points adjacent to the visibility interface, the value of ψ, according to (2.7), is minimized near the interface, particularly when ϕ corresponds to the signed distance function to the obstacle boundary. With this in mind, we propose a 'tube' structure to delineate the region near the actual visibility interface. Within this region, new evaluations of ψ depend solely on the ϕ values inside the tube, thereby reducing computational complexity to a further extent.

The tube is formed by a set of grid cells on the computational domain. It is basically designed to contain the grid cells next to the visibility interface up to a certain width. New evaluations of ψ would then only depend on ϕ values in the tube. The tube evolves step by step along with the seeding and growing method, and keeps taking advantage of the existing information at the same time.

To ensure the tube evolves properly for reliance in upcoming steps, we want to avoid skipping any grid cells closer to the observer when populating the current tube. Henceforth, the seeding and growing processes introduced in Algorithm 4 needs to be tailored, to maintain an ordering of grid cells according to their distance to the eye-point.

The min heap for the seeding process and queue for the nested growing process are then combined together as a min-priority queue, which can be implemented with a min-heap. Use the min heap from the seeding process to start with. The structure then evolves just like the queue in the growing process, while the top cell of the queue is now the element with the minimum priority, that is the root cell of the min-heap. The min priority is kept by constantly re-balancing the heap when adding and removing from the the min heap. In each loop checking out the root cell of the min-heap, the tube expands to contain a ball of grid cells, centered at the root cell, with a radius as the width of the tube. Then evaluations of ψ only depends on ϕ values within in the tube. The following Algorithm 5 outlines the steps of the seeding and growing method

altered and incorporated with the tube.

Algorithm 5: algorithm for finding visibility boundary: modified seeding and growing method with tube

```
function visibility (φ, x*);
Input  : a level set function φ on a uniform grid representing obstacles, and an
         eye-point location x*.
Output : visibility function ψ values on certain grid points next to the visibility
         boundary.
min-heapify all grid cells crossing the boundary of obstacles according to distance to
  eye-point;
while heap is not empty do
    get the root cell;
    update the tube to contain grid cells centered around the root cell up to a width;
    evaluate ψ at vertices of root cell with information within the tube;
    for each neighboring cell do
        if it is not once_in_heap and the shared face cuts through the interface then
            add it to the heap, mark it once_in_heap, and re-balance the heap;
        end
    end
    remove the root cell and re-balance the heap;
end
```

Compared to the Algorithm 4, Algorithm 5 sacrifices a bit time complexity in maintaining the min-priority queue. Let N_O and N_I represent the number of cells crossing the obstacle boundary and cells crossing the visibility interface, respectively. Usually, $N_I > N_O$. Algorithm 4 needs $O(N_O)$ time complexity to build the min-heap, $O(N_O \cdot \log N_O)$ to clear the min-heap, and $O(N_I)$ operations for the queue. On the other hand, Algorithm 5 needs $O(N_O)$ to build the min-heap, and the time complexity upper bounded by $O((N_O + N_I) \cdot \log N_O)$ to maintain the min-heap. The additional complexity does not pose a significant challenge when the obstacles are sparsely distributed and not too close to the observer, resulting in minimal obstruction.

In terms of complexity from evaluations, Algorithm 5 cuts down the number of ϕ function calls significantly, and hence decreases operations in evaluating ψ. This especially helps in the case when obstacles are far from the observer. In the meantime, the method used to evaluate ψ is not specified as in Algorithm 4. In the case of using ray tracing method with phase flow, the

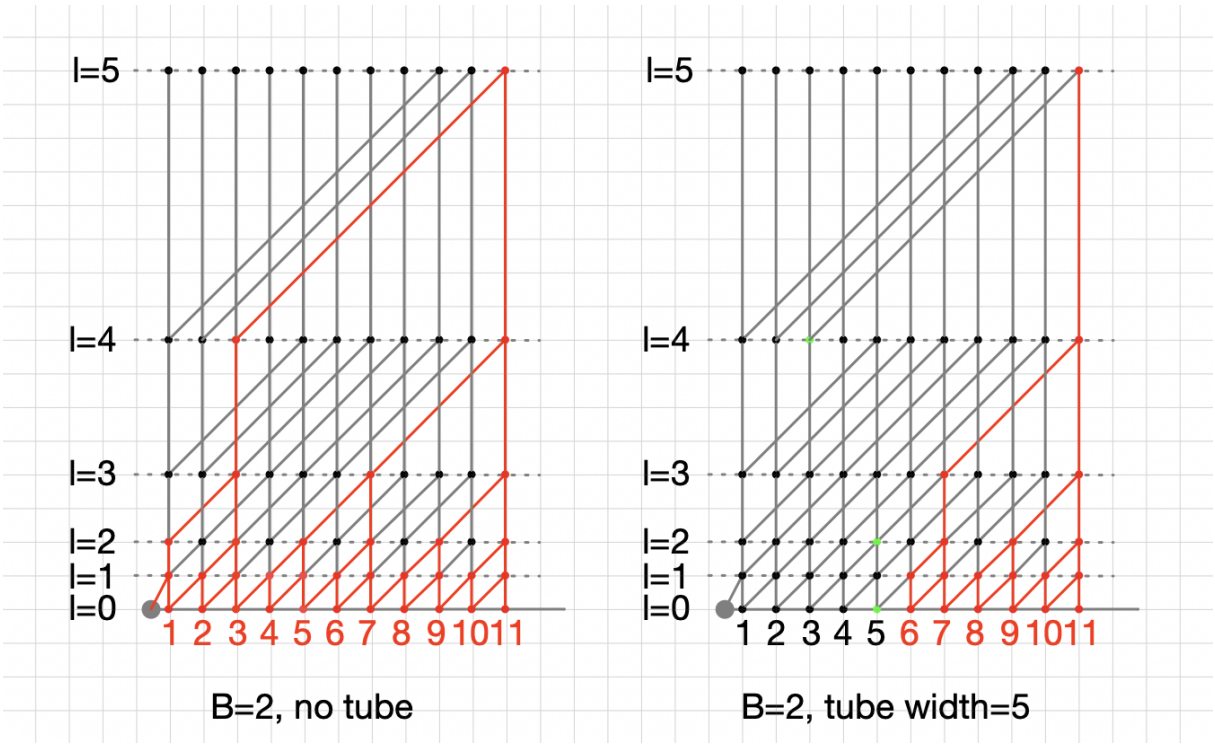


Figure 2.4. Illustration of operations with and without tube when using ray tracing method with time doubling phase flow on the grid point # 11. All red dots represent data calculated and stored in evaluating ψ at point # 11. Each branching of the red lines from a higher level to a lower level indicates an operation.

width of the tube is naturally set as the number of levels $S+1$ from the phase flow. Figure 2.4 shows how the introduction of the tube leads to a reduction in operations in a one dimensional case.

Although the tube brings benefits by eliminating number of evaluations, due to its limitation in width, Algorithm 5 can not perform well when obstacles are obscured by some other obstacles far away. This is because the tube is too narrow to pass the signal that the region is already obstructed by something farther along. To address this, a structure called ”cube” is introduced to work in conjunction with the tube, serving as a framework for querying visibility information outside the tube.

The cube develops from the eye-cell, i.e. the grid cell containing the eye-point, and it advances by width in L-infinity norm to form concentric squares around the eye-cell. Figure 2.6

illustrates the cube up to the layer with a width of 8, advancing in the direction of the gray arrows pointing outward. Each cell contains information *Vis* indicating if the cell is completely blocked, or else. The visibility information is updated along with the growing method, and advances layer by layer until it covers the whole domain.

To better integrate the cubic structure with existing growing method, a new metric called the "cubic distance" is proposed as an alternative to the Euclidean metric for use in the min-priority queue. It is modified based on the L-infinity distance between grid cells to the eye-cell. The new cubic distance should help the growing method progress along with outgoing layer of the cube, and maintain some ordering within the same cubic layer. A viable cubic distance between a grid cell containing $\mathbf{x}$ and the eye-cell is defined as below:

$$||index(cell\mathbf{x}) - index(cell\mathbf{x}^*)||_\infty + \frac{||index(cell\mathbf{x}) - index(Proj(cell\mathbf{x}^*))||_1}{(d-1) * ||index(cell\mathbf{x}) - index(cell\mathbf{x}^*)||_\infty + 1},$$

where the $Proj(cell\mathbf{x}^*))$ is the project of eye-cell onto the cubic face containing $\mathbf{x}$. The definition guarantees the cubic distance between the eye-cell and cells on the layer with width w ranges in $[w, w+1)$, and the distance increases as the cell moves farther from the center of the face it stands on, as illustrated by the horizontal arrows in Figure 2.6. As a remark, the level sets of the cubic distance to the center eye-cell form star-shaped patterns, meaning the cubic distance keeps viable ordering for the characteristics. This implies that when the cube extends to a specific grid cell, all the cells along the ray tracing back to the eye-cell have a smaller Euclidean distance to the eye-cell and are contained within the existing cube. Thanks to this property, the new distance measure serves as an effective replacement for the Euclidean metric in sorting operations involved in both the seeding and growing phases within the tube.

The update of *Vis* is done along with the advancement of the Cube. To start with, all cells are default to be visible ($+\infty$). On a given cubic layer, as the growth with the tube progresses, each time the root cell of the min-heap is checked, the *Vis* value is updated based on ψ approximations at its vertices. If the cell is completely invisible, *Vis* is marked as $-\infty$.

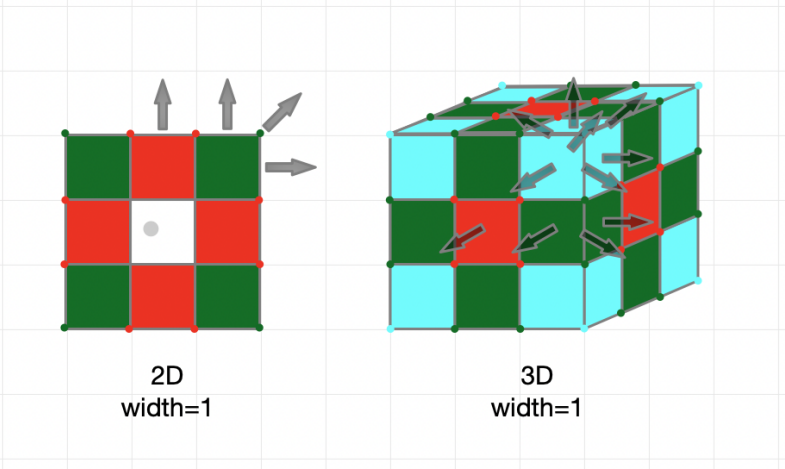


Figure 2.5. Illustration of *Vis* customization for cells that cross the visibility boundary, considering a cube where the second layer of cells intersects the boundary. Gray arrows indicate that the *Vis* determination in the current cell affects cells in the following layer. Red-filled cells customize its *Vis* using the sign of ψ at either of the two red-marked vertices on the cell's outgoing edge. Green-filled cells use the sign at a single corner vertex. This logic extends to 3D. Red cells determine *Vis* using the sign of ψ at any of the four red vertices on their outgoing face. Green cells use two green vertices on an edge, and cyan cells rely on a single corner vertex.

Otherwise, it is further customized when the cell crosses the visibility boundary, facilitating the update of *Vis* in the next cubic layer. Customization of *Vis* is based on the location of the cell in the current cubic layer, as depicted in Figure 2.5 and detailed in the following scheme: The cell is marked as ± 1 or 0 based on the sign of the ψ value at any one of its vertices that is shared by all the surrounding cells in the next cubic layer, as any such vertex yields the same result.

After the growing process finishes with all the cells in the min-heap from the current cubic layer, the cube is ready to advance to the next layer. If the untouched cell on the current layer has its preceding cell from the previous cubic layer with a negative *Vis*, it is completely obstructed, and marked as $-\infty$ accordingly. Over all, after *Vis* is updated to cover the whole domain, all cells with $Vis = -\infty$ are completely blocked, and all cells with finite *Vis* values are the ones cross the visibility interface. Figure 2.6 demonstrates cubic layers, and how *Vis* is updated alongside the growing method in a 2 dimensional scenario.

The cubic *Vis* information is used in conjunction with the tube for ψ evaluation. When the ψ evaluation requires data outside of the tube, if *Vis* value of the involving grid cell shows

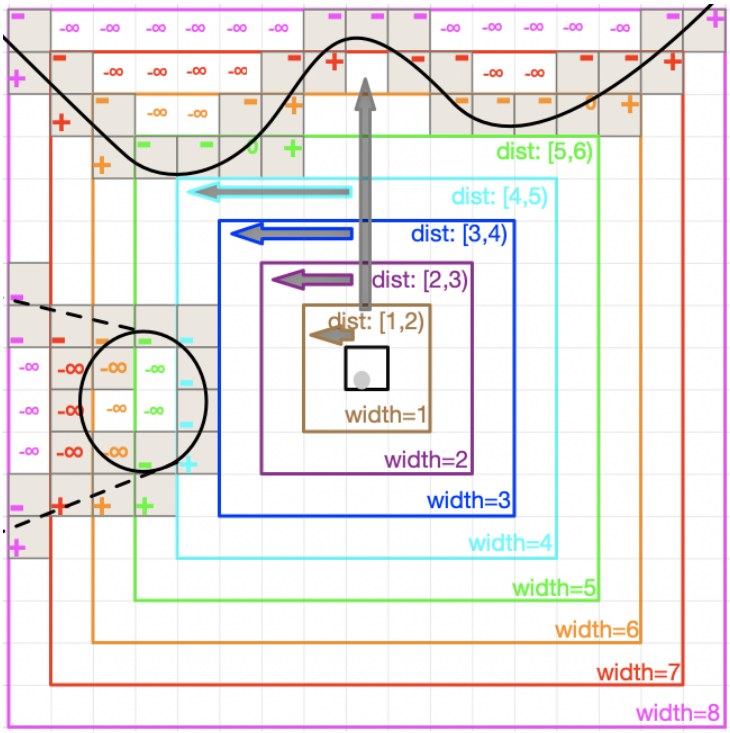


Figure 2.6. Illustration of *Vis* update with cubic layers in 2 dimension. The black curve indicates boundary of the obstacle, dotted line represents boundary of visible region, and the gray dot represents the eye-point. All shaded cells are once in the min-heap from the growing process. At gray cells, *Vis* is either $-\infty$ or customized with ± 1 or 0. When customized, vertices of a cell are scanned counterclockwise from its lower left corner. At cells not in shade, *Vis* is set as $-\infty$ when its preceding cell from the previous layer has negative value. Otherwise, it is visible.

it completely blocked, then the ψ evaluation is void, since anything behind a blocked obstacle is invisible as well. Sub-figures from Figure 2.7 show how the tube and cube progress in two dimensions as the growing method advances, while Figure 2.8 shows the final result once the process is complete. The integration of cubic update of visibility information successfully fixed the problematic case when there are obstacles obscured by others far away, as shown in Figure 2.9.

There is certainly some complexity introduced to maintain the cube in order to address issues caused by the tube alone. However, no additional evaluations of ψ or function calls to ϕ are required in the process, which would otherwise be the primary contributors to complexity.

As a note, upon the completion of the growing procedure with tube and cube, the cubic *Vis* information can be used to query visibility information on the whole computational domain. For cells that cross the visibility interface, the cube provides references to their corresponding ψ values as determined by the algorithm. For cells located far from the interface, the cube stores simpler binary information indicating whether they are visible or invisible. Optimizations can be applied to the current data structure of the cube to enable faster queries and improved storage efficiency.

The ultimate algorithm is shown in Algorithm 7 with the helper function Function 6. The Function 6 is configured to operate on specified point and level, using the ray tracing method with the time doubling phase flow in partnership with tube and cube, adapted for recursive use. Algorithm 7 combines all the structures we introduce in this section, and functions as a seeding and growing method with tube and cube.

In the following chapter, experiments are done with the Algorithm 7 to evaluate its order of accuracy and complexity compared to other methods. The discretization unit τ is by default set as the grid size h, and the width of the tube is set as the number of levels from the phase flow part, roughly in the scale of $\log h^{-1}$.

Function 6: formvisgridpttube function defined recursively to update visibility at the current grid point: utilizing phase flow method with tube and cube

function formvisgridpttube $(\mathbf{x}, l, \tau, V, \text{tube}, \text{Vis}, \phi, \mathbf{x}^*)$;

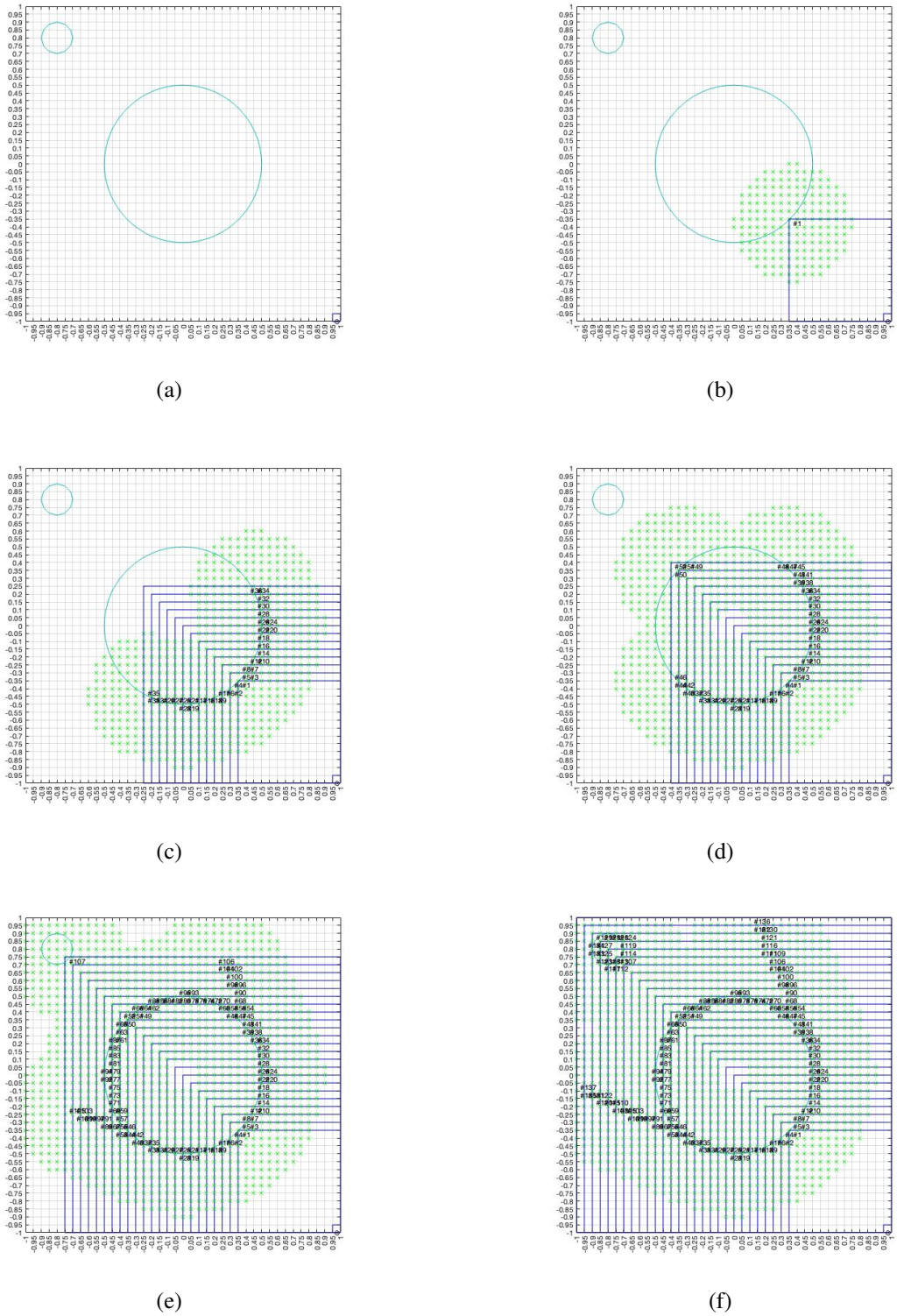
Input : a grid point of interest $\mathbf{x}$, the desired level l in phase flow, the discretization unit τ, stored visibility function values at all levels V, an existing *tube*, visibility information carried by the cube *Vis*, a level set function ϕ on a uniform grid representing obstacles, and an eye-point location $\mathbf{x}^*$.

Output : updated V, *tube*, and *Vis*.

if $V(\mathbf{x}, l)$ *is not evaluated* **then**
  if $l > 1$ **then**
    formvisgridpttube$(\mathbf{x}, l-1)$;
    $valuex \leftarrow V(\mathbf{x}, l-1)$;
  else
    $valuex \leftarrow \phi(\mathbf{x})$;
  end
  if $2^{(l-2)^+} \cdot \tau < ||\mathbf{x} - \mathbf{x}^*||$ **then**
    $\mathbf{y} = a - 2^{(l-2)^+} \cdot \tau \frac{\mathbf{x}-\mathbf{x}^*}{||\mathbf{x}-\mathbf{x}^*||}$;
  else
    $\mathbf{y} = \mathbf{x}^*$;
  end
  if *cell*$\mathbf{y}$ *is in tube* **then**
    if $\mathbf{y} \neq \mathbf{x}^*$ ***and*** $l > 1$ **then**
      formviscelltube$(cell\mathbf{y}, l-1)$;
      $valuey \leftarrow multilinear(V(cell\mathbf{y}, l-1))$;
    else
      $valuey \leftarrow multilinear(\phi(cell\mathbf{y}))$;
    end
    $V(\mathbf{x}, l) = \min(valuex, valuey)$;
  else
    $V(\mathbf{x}, l) = valuex$;
    if $Vis(cell\mathbf{y})$ *shows the cell completely blocked* **then**
      $V(\mathbf{x}, l) = -\infty$;
    end
  end
end

Algorithm 7: visibility algorithm for finding the boundary of visibility area given obstacles: operated by phase flow method with tube, together with growing method and cubic update of visibility info

```
function visibility (φ, x*);
Input  : a level set function φ on a uniform grid representing obstacles, and an
         eye-point location x*.
Output : visibility function ψ values on certain grid points next to the visibility
         boundary, together with Vis data in a cubic structure for fast query of
         visibility status of grid cells on the computational domain.
min-heapify all grid cells crossing the boundary of obstacles according to the cubic
  distance to eye-point;
while heap is not empty do
    get the root cell;
    update the tube to contain grid cells centered around the root cell up to a width;
    run formvisgridpttube at vertices of root cell;
    for each neighboring cell do
        if it is not once_in_heap and the shared face cuts through the interface then
            add it to the heap, mark it once_in_heap, and re-balance the heap;
        end
    end
    update Vis at the root cell;
    remove the root cell and re-balance the heap;
    while the next root cell is not covered by the current cubic layer do
        advance the cube by one layer;
    end
end
update the remaining cubic layers to cover the whole domain.
```



(a) (b) (c) (d) (e) (f)

Figure 2.7. Visualization of the tube and cube's progression in two dimensions as the growing method unfolds, shown sequentially from (a) to (f).

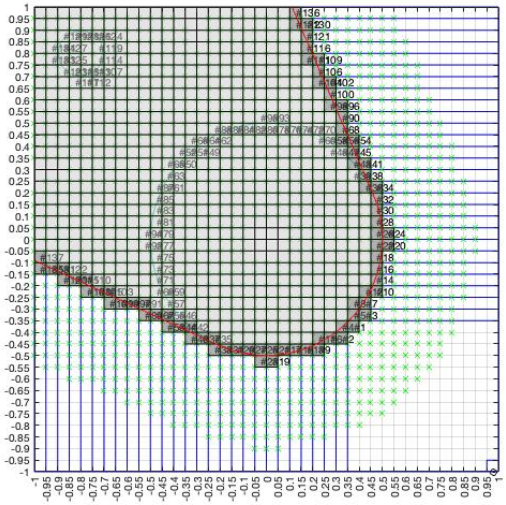


Figure 2.8. Illustration of the final state of the tube and cube in two dimensions after executing Algorithm 7. The blue lines outline the cubes storing visibility information of grid cells. All grid cells in light gray are completely invisible, while the dark gray ones are crossing the visibility interface. The red line is the approximate visibility interface based on visibility function values on the dark gray cells. All evaluations rely on the grid points labeled with green cross, which represent the vertices of grid cells from the tube.

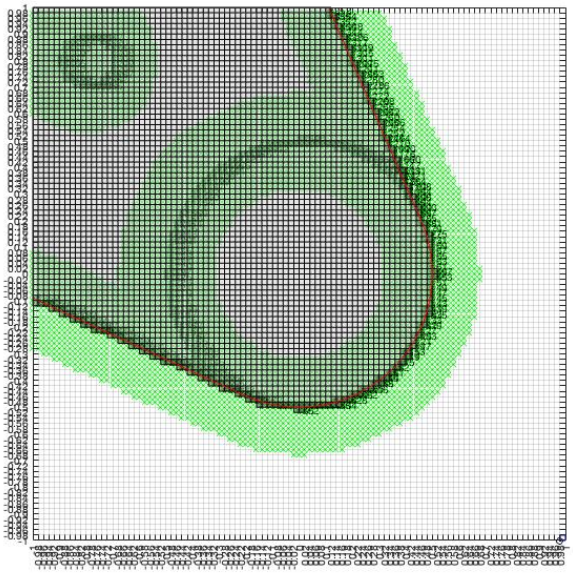
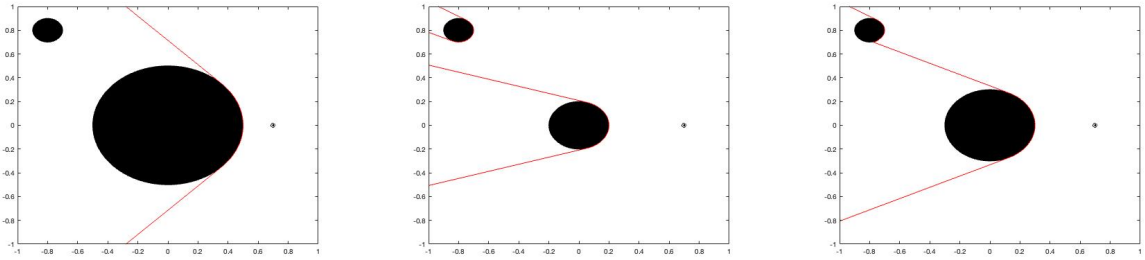


Figure 2.9. Illustration of necessity of cube in Algorithm 7 in 2 dimension. It is a show case when the tube itself is too narrow to pass enough visibility information to the obstacle obstructed by the other one far away. The tube is highlighted in green, and the gray shade marks the information carried by the cube.

Chapter 3

Numerical Results

To ensure consistent scaling for comparison, all experiments are performed within the fixed computational domain $[-1,1] \times [-1,1]$, with the eye-point positioned at $(0.7,0)$. The following three cases encompass the primary scenarios related to vision obstruction: obstacles farther from the eye-point can be completely invisible, completely visible, or partially blocked, as shown in Figure 3.1.



(a) Case1-1:
One circle centered at $(-0.8,0.8)$ with radius 0.1 completely blocked by another circle centered at $(0,0)$ with radius 0.5

(b) Case2:
One circle centered at $(-0.8,0.8)$ with radius 0.1 and another circle centered at $(0,0)$ with radius 0.2 mutually non-occluding

(c) Case3-1:
One circle centered at $(-0.8,0.8)$ with radius 0.1 partially blocked by another circle centered at $(0,0)$ with radius 0.3

Figure 3.1. Three cases for accuracy and complexity testing of Algorithm 7.

3.1 Accuracy

There are two primary approaches for measuring accuracy of our approximation of the visibility interface, referred to as E_1 and E_2. Both methods have their advantages and limitations.

1. E_1 quantifies the distance between the approximate visibility interface and the actual visibility interface. Based on visibility function ψ values from the output of the Algorithm 7 adjacent to the approximate visibility interface, linear interpolation is used to compute the approximate zero-level points on the grid lines, and the distance between these points and the actual visibility interface is defined as E_1. It is denoted as E_1^{inf} when measured in L-infinity norm, and E_1^{ave} when measured in L-1 norm divided by the number of evaluations.

2. E_2 quantifies the difference between the approximate and actual values of the visibility function ψ at grid points adjacent to the approximate visibility interface. It is denoted as E_2^{inf} when measure in L-infinity norm, and E_2^{ave} when measured in L-1 norm divided by the number of evaluations.

The distance between the approximate visibility interface and actual visibility interface is generally a more reliable metric to assess accuracy across different scenarios or cases, since a distance-based metric is more robust to variability caused by influence of different configurations (e.g., obstacle function ϕ choices and scales) on the other metric. However, it is difficult to measure numerically without knowing points right on the approximate visibility interface. E_1 uses linear interpolation of approximate ψ values on the grip points next to the zero-level set, basically a one-step secant method, to approximate the zero-level points on the grid lines. This root-finding method introduces error in the quantification of the measure E_1, but it aligns with the default approach for visualizing a zero-level set in Matlab [16]. To ensure that E_1 is meaningful, it is typically evaluated on the actual analytic visibility function ψ as well for reference, denoted as refE_1, in order to exclude the effects of linear interpolation.

In Algorithm 7, the output visibility function ψ values are only available at end-points of certain grid cells that cross the "approximate" visibility boundary. Therefore E_2 can not catch the difference at grid points adjacent to the "actual" visibility interface, as shown in Figure 3.2, and it may introduce biases or unfairness when used as a comparison metric among different

visibility algorithm. More alarmingly, a significant limitation of E_2 is that the analytic form of ψ is often unavailable in experimental scenarios. This absence necessitates numerical methods, which can introduce computational complexities and errors. Regardless, when analytic ψ is available, E_2 is more practical to calculate than E_1.

In Table 3.1, the E_1^{inf} of Case3-1 shows significant lower order of accuracy, close to $O(h)$. This occurs at the sharp corner of the partially blocked region. When smoothing out with E_1^{ave}, the result looks much better. This justifies that the $O(h)$ accuracy is local when the analytic ψ is not continuously differentiable at the corner. At the same time, Figure 3.3 shows that the E_1^{inf} of Case3-1 is much higher than the other cases, and not much different from the reference error refE_1^{inf} when calculated with its analytic ψ, indicating that the problem is dominated by the error introduced by linear interpolation in E_1.

As a remark, the periodic artifacts shown in Figure 3.3 is a result of finite precision and rounding errors of floating-point arithmetic. Nevertheless, the general trend of convergence of errors is still obvious from the plot, and the dominance of the error introduced by linear interpolation in E_1 less noticeable when averaging is applied, in E_1^{ave}. Therefore the order of accuracy measured in E_1^{ave} makes sense, indicating the Algorithm has an order of accuracy not far from 2. You can also get a sense of the order of accuracy from the Table 3.1 comparing row $N = 100, 200, 400$.

Another noticeable observation from Table 3.1 is that Case 2 shows the best performance, followed by Case 3-1, and then Case 1-1 in terms of E_*^{ave}. First of all, the weaker outcome in Case 1-1 is not a result of blockage or anything caused by the tube and cube in the growing method, since another experiment is preformed with only the one circular obstacle in the front, and the result with N ranging from 65 to 512 is identical, except for arithmetic calculation errors. Instead, comparing the ϕ level sets in the three cases from Figure 3.4, the performance difference may stem from the ϕ scale inside the obstacle, as calculating ψ essentially involves looking of its minimum along lines, and ϕ is only negative inside. To validate this reasoning, another set of three cases is tested for comparison, as shown below in Figure 3.5 and Table 3.2. The result

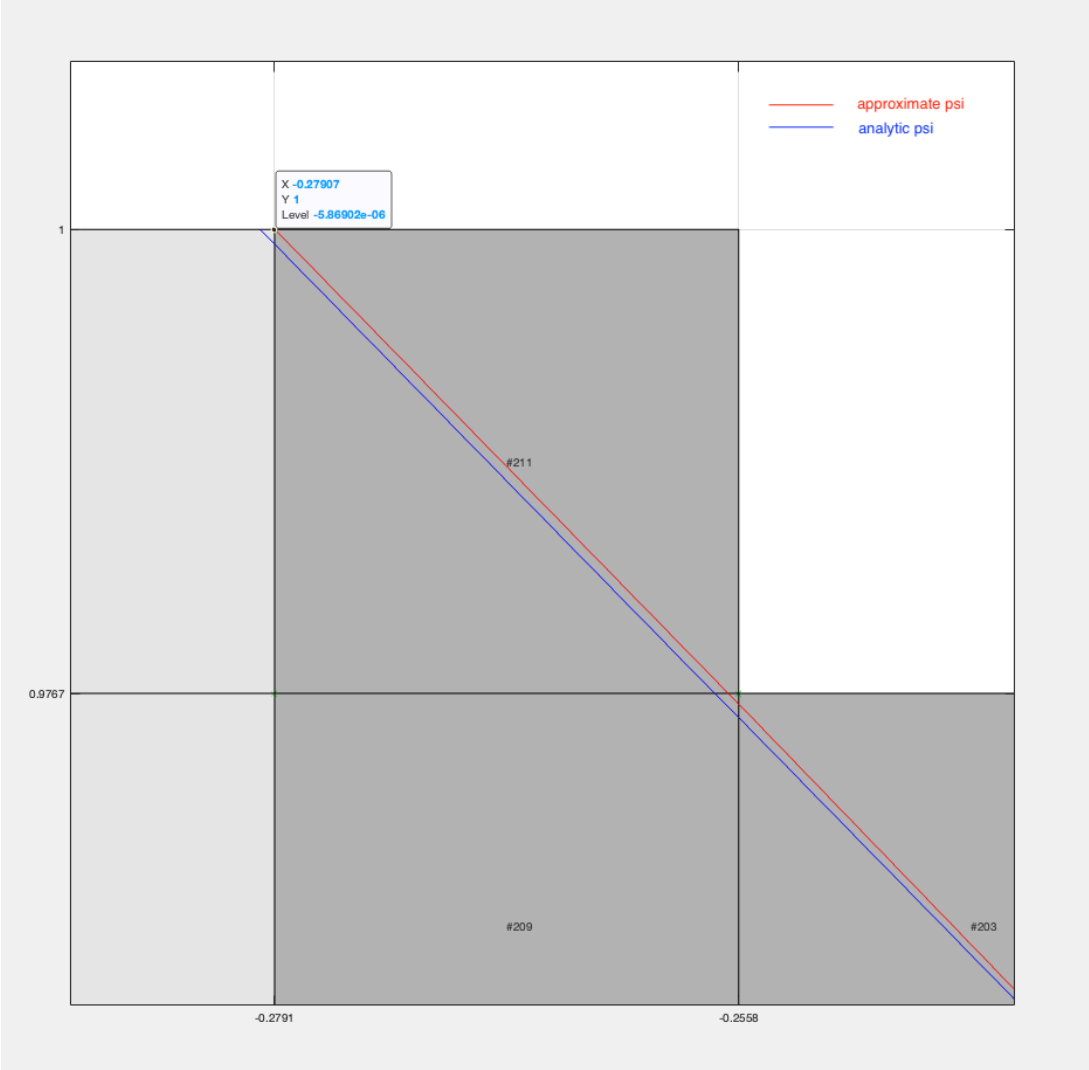
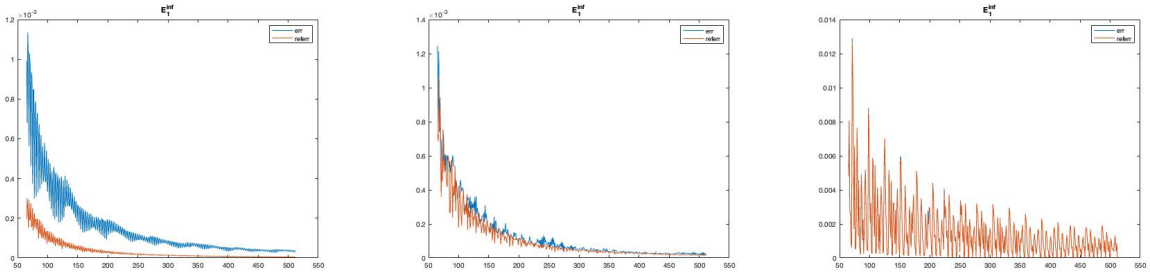
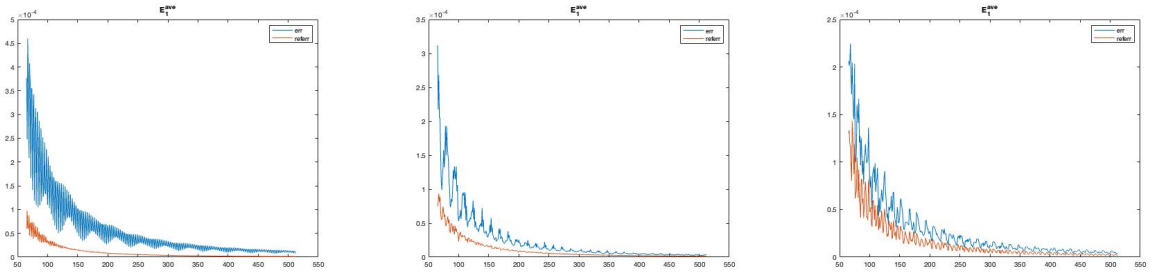


Figure 3.2. Zoom-in picture of the problematic area in an experiment of Case 1-1 with $N = 86$. The blue line indicates the actual zero-level set of the analytic ψ, while the red refers to the approximate one from the Algorithm 7. At grid point $(-0.2791, 1)$, the approximate ψ gives false negative, and consider the grid cell to the left of #211 completely obstructed while it is not. Thus there is no valid data for E_2 evaluation if measured at grid points adjacent to the "actual" visibility interface.

reaffirms the second-order accuracy in E_1^{ave}, and $O(h)$ accuracy in E_1^{inf} for the case with corner on the visibility interface.



(a) E_1^{inf}: Case1-1 (b) E_1^{inf}: Case2 (c) E_1^{inf}: Case3-1



(d) E_1^{ave}: Case1-1 (e) E_1^{ave}: Case2 (f) E_1^{ave}: Case3-1

Figure 3.3. E_1 and its reference refE_1 on the three cases for accuracy.

3.2 Complexity

There are six quantities used to measure complexity of a iterative visibility algorithm based on the problem set up in our work, as listed below:

- N_1: number of requests of the key iterative function,
- N_2: number of executions of the key iterative function,
- N_3: number of interpolations needed,
- N_4: number of requests of ϕ value, including redundant requests at the same grid point,
- N_5: number of ϕ values involved at different grid points,

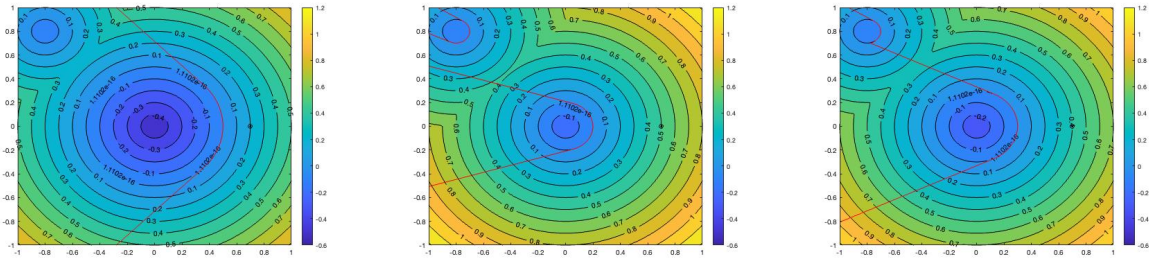
Table 3.1. Accuracy of Algorithm 7 tested in the three cases of two circular obstacles.

N	Case1-1				Case2				Case3-1			
	E_1^{inf}	E_1^{ave}	E_2^{inf}	E_2^{ave}	E_1^{inf}	E_1^{ave}	E_2^{inf}	E_2^{ave}	E_1^{inf}	E_1^{ave}	E_2^{inf}	E_2^{ave}
100	3.335e-4	1.101e-4	1.335e-4	4.508e-5	4.188e-4	5.273e-5	4.413e-4	4.217e-5	3.995e-3	4.254e-4	8.217e-5	3.149e-5
150	2.666e-4	1.024e-4	1.080e-4	4.254e-5	1.784e-4	3.462e-5	2.385e-4	2.022e-5	1.784e-4	2.385e-4	4.238e-5	1.976e-5
200	1.939e-4	6.698e-5	6.859e-5	2.867e-5	1.134e-4	2.557e-5	1.310e-4	1.619e-5	2.907e-3	1.310e-4	2.731e-5	9.007e-6
250	9.022e-5	2.778e-5	3.695e-5	1.113e-5	8.621e-5	1.073e-5	6.015e-5	6.384e-6	4.129e-4	6.014e-5	1.869e-5	7.920e-6
300	6.858e-5	2.167e-5	2.718e-5	8.461e-6	4.702e-5	8.179e-6	4.452e-5	4.936e-6	1.757e-4	4.451e-5	1.296e-5	4.020e-6
350	7.148e-5	2.489e-5	2.513e-5	1.028e-5	3.289e-5	5.500e-6	3.976e-5	3.436e-6	2.457e-4	3.976e-5	9.491e-5	3.959e-6
400	5.436e-5	1.995e-5	2.115e-5	8.509e-6	3.611e-5	4.122e-6	4.087e-5	2.171e-6	9.717e-4	4.087e-5	6.922e-5	2.492e-6
450	3.369e-5	1.087e-5	1.331e-5	4.373e-6	2.627e-5	3.943e-6	2.839e-5	2.839e-6	3.482e-4	2.839e-5	6.081e-5	2.480e-6
500	3.259e-5	9.759e-6	1.194e-5	3.943e-6	2.115e-5	2.897e-6	2.054e-5	2.054e-6	2.066e-5	2.054e-5	4.249e-5	2.032e-6
$O(h^{\alpha})$	1.5633	1.6225	1.5928	1.6045	1.8295	2.0699	1.8422	2.0800	0.9621	1.7795	1.8059	1.8571

Notes:

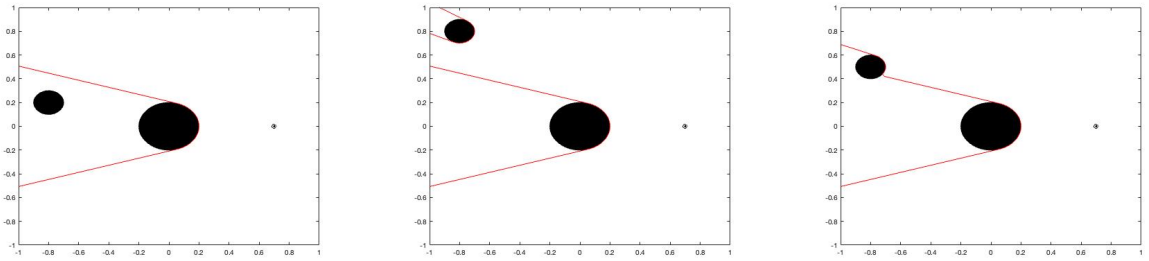
1. E_1 and E_2 refer to the two approaches measuring accuracy. E_1^{inf} and E_1^{ave} are basically used to compare among the three cases, while E_2^{inf} and E_2^{ave} provide extra information with regard to accuracy in each specific case.

2. N denotes the number of divisions along each axis and scales as $O(h^{-1})$. The table presents only specific values of N, while the analysis of order of accuracy is conducted by a log-log regression based on a much larger dataset, with N from 65 to 512.



(a) Case1-1 (b) Case2 (c) Case3-1

Figure 3.4. Level sets of the obstacle function ϕ in the three cases for experiment.



(a) Case1-2:
One circle centered at $(-0.8, 0.2)$ with radius 0.1 completely blocked by another circle centered at $(0, 0)$ with radius 0.2

(b) Case2:
One circle centered at $(-0.8, 0.8)$ with radius 0.1 and another circle centered at $(0, 0)$ with radius 0.2 mutually non-occluding

(c) Case3-2:
One circle centered at $(-0.8, 0.5)$ with radius 0.1 partially blocked by another circle centered at $(0, 0)$ with radius 0.2

Figure 3.5. Three new cases for accuracy testing of Algorithm 7.

Table 3.2. Accuracy of Algorithm 7 tested in the three new cases of two circular obstacles. The accuracy order results in E_1^{ave} for the three cases do not differ as significantly as the previous results shown in Table 3.1.

N	Case1-2		Case2		Case3-2	
	E_1^{inf}	E_1^{ave}	E_1^{inf}	E_1^{ave}	E_1^{inf}	E_1^{ave}
100	1.960e-4	4.222e-5	4.188e-4	5.273e-5	3.031e-4	4.981e-5
150	1.748e-4	2.529e-5	1.784e-4	3.462e-5	8.144e-4	3.256e-5
200	7.780e-5	2.193e-5	1.134e-4	2.557e-5	8.402e-4	2.589e-5
250	8.621e-5	6.181e-6	8.621e-5	1.073e-5	2.353e-4	9.710e-6
300	4.343e-5	6.058e-6	4.702e-5	8.179e-6	1.893e-4	1.236e-5
350	3.289e-5	3.925e-6	3.289e-5	5.500e-6	5.856e-4	7.216e-6
400	2.323e-5	2.613e-6	3.611e-5	4.122e-6	1.012e-4	4.150e-6
450	1.454e-5	2.559e-6	2.627e-5	3.943e-6	1.443e-3	5.616e-6
500	2.115e-5	1.819e-6	2.115e-5	2.897e-6	4.147e-4	3.466e-6
$O(h^\alpha)$	1.7924	2.1340	1.8295	2.0699	1.1855	2.0296

- N_6: number of ψ values calculated at grid points.

In the case of Algorithm 7, the key iterative function refers to Function 6.

Suppose N is the number of divisions along each axis and scales as $O(h^{-1})$. The gridded domain has $O(N^{d-1})$ number of grid points close to the visibility interface, and the width of the tube is set to be $O(\log N)$. Therefore the number of ψ values calculated (N_6) has order of $O(N^{d-1})$ and the different ϕ values needed (N_5) under the growing method with the tube has order of $O(N^{d-1}\log N)$. As we have seen in section 2.2, the construction of $\tilde{\psi}_T$ is divided into S stages, and each stage requires applying the repetitive operations (minimization and interpolation) B times. In our Algorithm 7, the iterative Function 6 uses the time doubling scheme with fixed $B = 2$, so S scales as $O(\log N)$. In general, N_1 to N_4 should all be proportional to $O(N^{d-1}(\log N)^2)$. The Figure 3.6 below illustrates the trend and order of the dataset, with N ranging from 64 to 512, and Table 3.3 provides a more quantitative approximation of the order of complexity, both providing evidence to verify our analysis in the 2D case.

A horizontal comparison of complexity between methods is shown in Figure 3.7, featuring the phase flow + growing method and the trivial method + growing. Together with Figure 3.6,

Table 3.3. Complexity of Algorithm 7 tested in the three cases of two circular obstacles. The order of complexity α and β are approximated by log-log regressions on a data set with N from 65 to 512. The value of α is below 2, and β for N_1 through N_4 is approximately 2 across all three cases.

	Case1-1		Case2		Case3-1	
	$O(N^\alpha)$	$O(N\cdot(\log N)^\beta)$	$O(N^\alpha)$	$O(N\cdot(\log N)^\beta)$	$O(N^\alpha)$	$O(N\cdot(\log N)^\beta)$
N_1	1.6039	1.9682	1.5430	1.7694	1.5981	1.9485
N_2	1.5529	1.8019	1.4927	1.6056	1.5443	1.7729
N_3	1.5119	1.6682	1.4565	1.4872	1.5138	1.6736
N_4	1.7706	2.5127	1.7146	2.3299	1.7715	2.5147
N_5	1.2578	1.0875	1.1964	0.9643	1.2684	1.4320
N_6	1.2478	1.1669	1.1948	1.0288	1.2865	1.5288

it is evident that the phase flow method with tube and cube + growing requires the fewest operations among the three methods, especially when compared to the trivial method at scale. The improvement in the order of complexity is also clearly demonstrated in the comparison.

In terms of running time, Algorithm 7 requires approximately 1 second to run on a 100×100 grid resolution, making it both efficient and promising for practical applications.

3.3 Other Experiments

Due to the lack of an available analytic visibility function ψ in most cases, we cannot conduct extensive experiments for accuracy analysis. However, Algorithm 7 is capable of detecting visibility boundaries in many scenarios, even with complex-shaped obstacles. Below in Figure 3.8 are additional experiments where our algorithm performs exceptionally well, as evidenced by its visual results.

For sophisticatedly contoured obstacles, Figure 3.9 illustrates a butterfly shape [31] represented by pixels. After pre-processing steps such as noise reduction, rotation, normalization, resizing, and thresholding, the butterfly-shaped obstacle is placed at the center of the domain. It is represented by its level set function ϕ, where $\phi \in [-1,0)$ inside the obstacle and padded with 1 outside. The visibility algorithm is then applied, and Algorithm 7 demonstrates excellent

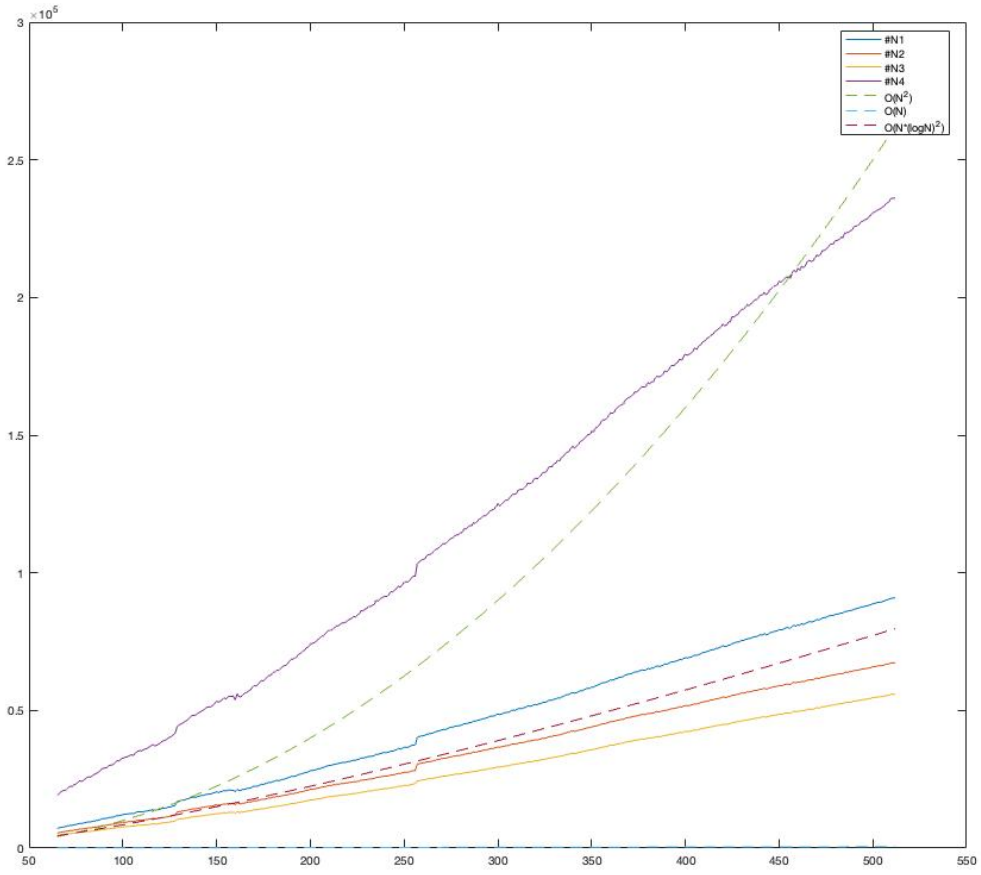
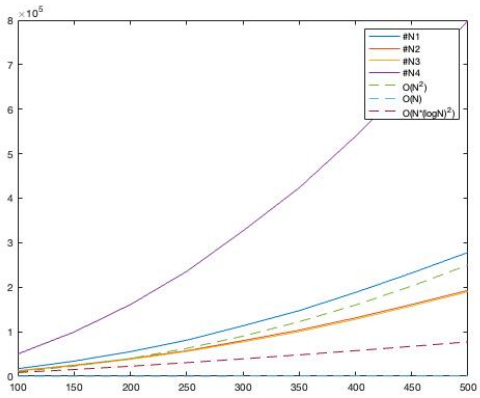
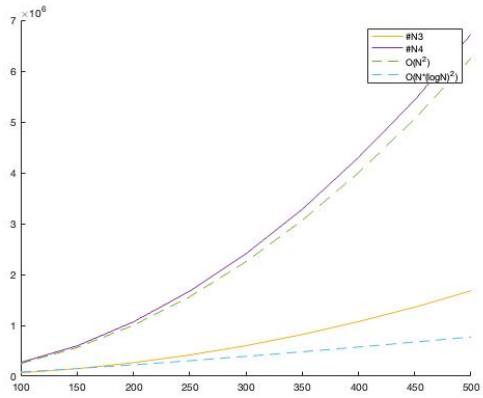


Figure 3.6. Complexity test in Case 2, using of Algorithm 7: phase flow with tube and cube + growing. N_1 through N_4 all scale approximately as $O(N\log^2 N)$.



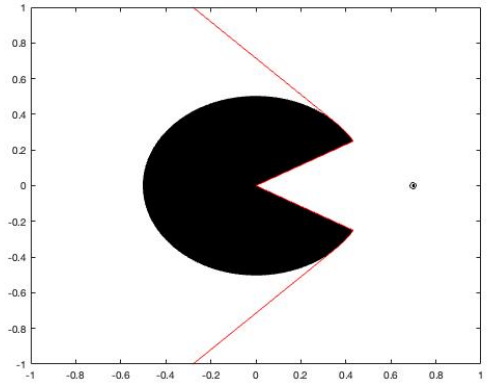
(a) Phase flow + growing



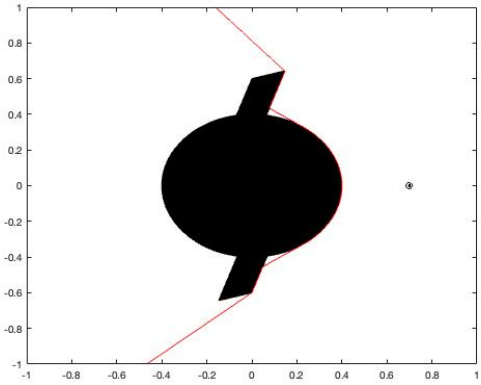
(b) Trivial + growing

Figure 3.7. Complexity comparison between methods. The trivial + growing method requires $O(N^2)$ operations, while the phase flow + growing method has a slightly higher order of complexity than our Algorithm 7, but performs better than the trivial method.

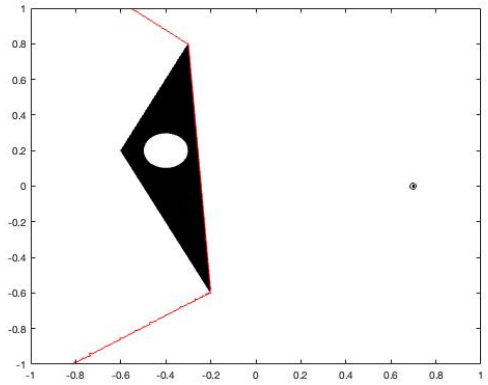
performance in capturing the visibility interface.



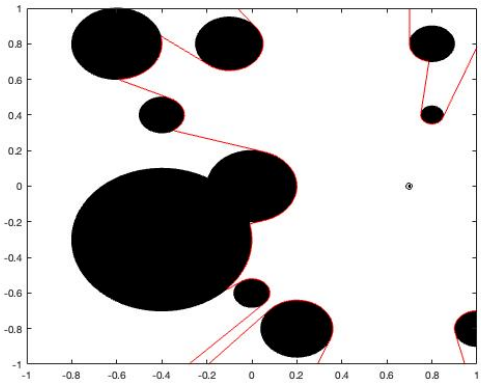
(a) Experiment on a concave-shaped obstacle: the Pac-Man.



(b) Experiment on an irregularly shaped obstacle: a lollipop-like shape resembling the structure of the Greek letter ϕ.

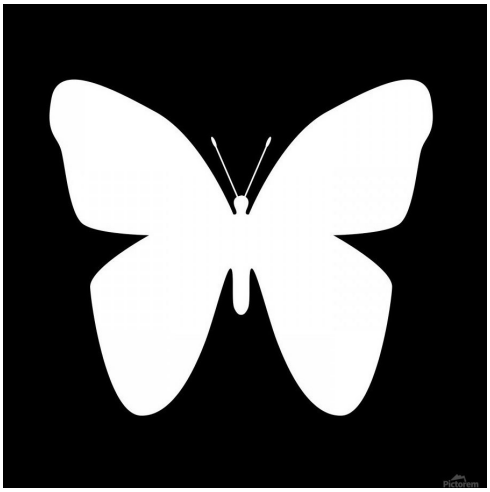


(c) Experiment on a path-connected but not simply-connected obstacle: a triangular shape containing a hollow circular cutout.

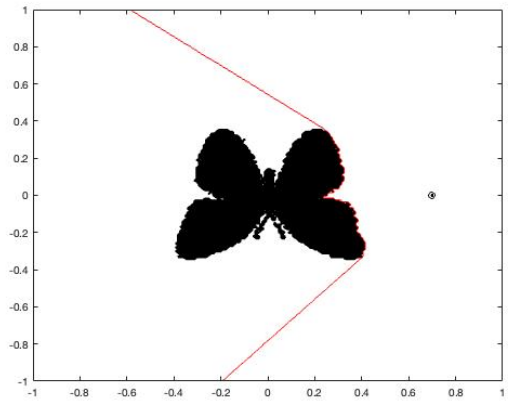


(d) Experiment on multiple occluding obstacles: several circular obstacles scattered across the domain, illustrating layered obstructions.

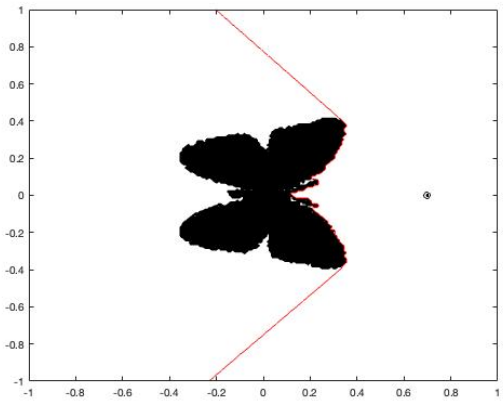
Figure 3.8. Algorithm 7 performs well on complex-shaped obstacles.



(a) A butterfly-shaped gray-scale image.



(b) The visible parts of the butterfly's left wing are accurately captured.



(c) The antennae of the butterfly are properly captured.

Figure 3.9. Algorithm 7 performs well on gray-scale image. The gray-scale image can be loaded as a pixel matrix. The pixel values are mapped linearly to create its level set function ϕ, with 255 mapped to -1, 55 to 0, and 0 to 1. The butterfly shape is then placed in the region $[-0.5, 0.5] \times [-0.5, 0.5]$ as an obstacle.

Chapter 4

Approximation of Volume of visible region

Approximating the volume of the visible region is a fundamental goal in visibility-related optimization problems, because it provides a quantitative measure of how much of the environment can be observed from one or more viewpoints. This metric is instrumental in optimizing viewpoints for maximal coverage [4] and is frequently employed to evaluate the effectiveness of viewpoints.

4.1 Problem Set Up

Consider a subset in the domain $D \in \mathbb{R}^d$ denoted as Ω, with its boundary Γ. Then the volume of the region inside the region Ω is given by

$$V = \int_D H_\Omega \, d\mathbf{x},$$

where H_Ω is a Heaviside step function indicating the region Ω, that is

$$H_\Omega(\mathbf{x}) = \begin{cases} 1, & \text{if } \mathbf{x} \in \Omega, \\ 0, & \text{otherwise.} \end{cases}$$

We use D_h to denote the discretized computational domain, with h the step size. Let $\mathbf{x}_i$ denote grid points of D_h. Our goal is to find a discrete version of H_Ω denoted $\tilde{H}_i$ so that the approximation is

maintained to some order in h, that is,

$$V = \sum_i h^d \tilde{H}_i + O(h^p)$$

where $p > 0$.

In the visibility problem, when Ω represents the visible region, it is characterized by $\{\mathbf{x} : \psi(\mathbf{x}) > 0\}$. Accordingly, Γ refers to the visibility interface. The volume of the visible region is then given by

$$V = \int_D H(\psi(\mathbf{x}))\, d\mathbf{x},$$

where H is the one dimensional Heaviside step function

$$H(x) = \begin{cases} 1, & \text{if } x > 0, \\ 0, & \text{otherwise.} \end{cases}$$

Numerically, given visibility values ψ_i at grid points $\mathbf{x}_i$ over the entire computational domain D_h, we can approximate the volume of visible region as

$$V = \sum_i h^d \tilde{H}(\psi_i) + O(h^p)$$

where $p > 0$. As a remark, it is actually sufficient to consider only the ψ_i data close to the visibility interface, together with binary visible/invisible information elsewhere, as in the format of the output from the algorithm previously presented in the dissertation.

In the volume approximation of visible region, error could only occur close to the visibility interface Γ. With $O(h^{-(d-1)})$ grid points near Γ, setting $\tilde{H}(\psi_i) = H(\psi_i)$ would only guarantee a first order accuracy. Therefore, our goal is to find a higher order approximation of the Heaviside function with a step jump on the zero-level set of a continuous function ψ. A second order Heaviside approximation can be constructed based on the work from Peter Smereka

[26] when he approximates the Delta Function.

4.2 Smereka's Delta Function Approximation

Let $\delta(x)$ be the usual one-dimensional Dirac delta function, and $d(\mathbf{x})$ be the signed distance function to Γ from a point $\mathbf{x} \in D \subset \mathbb{R}^d$, where $d(\mathbf{x})$ is negative inside the region Ω and positive outside. The arc-length in the case $d = 2$ or surface area in the case of $d = 3$ of Γ is given by

$$L = \int_{\Omega} \delta(d(\mathbf{x}))\, d\mathbf{x}.$$

Peter Smereka constructed a discrete Delta function, denoted $\tilde{\delta}_i$ on grid point $\mathbf{x}_i$, using a technique developed by Anita Mayo [17, 18], so that

$$L = \sum_i h^d \tilde{\delta}_i + O(h^2).$$

In Smereka's paper [26], firstly he treats the Delta function as a second order derivative function based on the Green's function for Laplace's equation, and then rewrites its equivalent form with jump conditions. In one dimensional space on the unit interval with $0 < x_\Gamma < 1$ as the singular point,

$$g''(x) = \delta(x - x_\Gamma) \iff \begin{cases} g''(x) = 0, \quad x \neq x_\Gamma, \\ [g]_x = 0, \quad [g']_x = 1, \quad [g'']_x = 0, \end{cases}$$

where $[g]_x$ denotes jump across x_Γ in positive x direction, $[g]_x := \lim_{\varepsilon \to 0} g(x_\Gamma + \varepsilon) - g(x_\Gamma - \varepsilon)$, also denoted as $g_{\Gamma^+} - g_{\Gamma^-}$.

Similarly, in higher dimension, the paper devises the discrete delta function by consider-

ing the following elliptic problem in its equivalent form with regard to jump conditions.

$$\Delta g(\mathbf{x}) = \delta(d(\mathbf{x})) \iff \begin{cases} \Delta g(\mathbf{x}) = 0, \quad \mathbf{x} \notin \Gamma, \\ [g] = 0, \quad [\partial_n g] = -1, \quad [\partial^2_{nn} g] = 0, \end{cases}$$

where $\mathbf{n}$ is the normal direction of Γ pointing outward of Ω, i.e. the direction along which $d(\mathbf{x})$ goes down the fastest, referred to as the outward normal. ∂n and ∂^2_{nn} represents the first and second order directional derivative in the outward normal direction, and $[g]$ denotes jump across the singular boundary Γ in the outward normal direction $\mathbf{n}$, $[g] := \lim_{\varepsilon \to 0} g(\mathbf{x}_\Gamma + \varepsilon \mathbf{n}) - g(\mathbf{x}_\Gamma - \varepsilon \mathbf{n})$.

Then he discretizes the second order term with good care of the jump conditions using techniques from Anita Mayo [17, 18], and that in return, gives a numerical approximation of the delta function. For consistency in terminology, we adhere to Mayo's work, that grid points are categorized into two types: irregular points and regular points. Irregular points are the ones within one mesh distance to the interface. That is to say, if there is an interface point $\mathbf{x}_\Gamma$ between grid points $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$, then $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$ are irregular points. All the other grid points are denoted as regular points.

4.3 Heaviside Function Approximation

Consider the Partial Differential Equation

$$\Delta g = H_\Omega \tag{4.1}$$

with boundary conditions

$$[g] = 0, \left[\frac{\partial g}{\partial n}\right] = 0$$

on $\partial\Omega$ and $g = 0$ on computational domain boundary ∂D.

Start with one dimension:

$$g'' = H_\Omega \tag{4.2}$$

with boundary conditions

$$[g] = 0, \left[g'\right] = 0$$

on $\partial\Omega$, where $n = \pm 1$ is the outward unit normal on $\partial\Omega$, and $g = 0$ on ∂D.

Consider a boundary point x_Γ lying in between grid points x_i and x_{i+1}, and suppose $x_\Gamma = x_i + \alpha h$ where $0 < \alpha < 1$. From a Taylor Series expansion, one has

$$g_{i+1} = g_{\Gamma^+} + (1-\alpha)hg'_{\Gamma^+} + \frac{(1-\alpha)^2h^2}{2}g''_{\Gamma^+} + O(h^3), \tag{4.3}$$

$$g_i = g_{\Gamma^-} - \alpha hg'_{\Gamma^-} + \frac{\alpha^2h^2}{2}g''_{\Gamma^-} + O(h^3). \tag{4.4}$$

Since there are no jumps in g for $x_{i-1} < x_\Gamma$, then

$$g_{i-1} = g_i + hg'_i + \frac{h^2}{2}g''_i + O(h^3). \tag{4.5}$$

Combining (4.5) together with first and second order derivatives of (4.4),

$$\begin{aligned} g_{i-1} &= g_i - hg'_i + \frac{h^2}{2}g''_i + O(h^3), \\ &= g_i - hg'_{\Gamma^-} + \alpha h^2 g''_{\Gamma^-} + \frac{h^2}{2}g''_{\Gamma^-} + O(h^3), \\ &= g_{\Gamma^-} - (1+\alpha)hg'_{\Gamma^-} + \frac{(1+\alpha)^2h^2}{2}g''_{\Gamma^-} + O(h^3). \end{aligned} \tag{4.6}$$

Subtracting (4.4) from (4.3) gives

$$g_{i+1} - g_i = n[g] - \alpha hn\left[g'\right] + hg'_{\Gamma^+} + \frac{(1-\alpha)^2h^2}{2}g''_{\Gamma^+} - \frac{\alpha^2h^2}{2}g''_{\Gamma^-} + O(h^3). \tag{4.7}$$

And subtracting (4.4) from (4.6) gives

$$g_{i-1} - g_i = -hg'_{\Gamma^-} + \frac{(2\alpha+1)h^2}{2} g''_{\Gamma^-} + O(h^3). \tag{4.8}$$

Adding (4.7) and (4.8), and combining with second order derivative of (4.4),

$$g_{i+1} - 2g_i + g_{i-1} = n[g] + (1-\alpha)hn\left[g'\right] + \frac{(1-\alpha)^2h^2}{2} n\left[g''\right] + h^2 g''_i + O(h^3),$$

which can be reorganized as

$$g''_i = \frac{g_{i+1} - 2g_i + g_{i-1}}{h^2} - \frac{n[g]}{h^2} - \frac{(1-\alpha)n[g']}{h} - \frac{(1-\alpha)^2}{2} n\left[g''\right] + O(h).$$

Now, with $[g] = 0, [g'] = 0$, and the jump condition $[g''] = -1$, we have

$$g''_i = \frac{g_{i+1} - 2g_i + g_{i-1}}{h^2} + \frac{(1-\alpha)^2}{2} n + O(h). \tag{4.9}$$

Note the added factor of

$$\frac{(1-\alpha)^2}{2} n.$$

More generally, noting that

$$g_{\Gamma^\pm} - g_{\Gamma^\mp} = \pm n[g],$$

suppose the boundary point x_Γ is in between $g_{i\pm1}$ and g_i,

$$\begin{aligned}
g_{i\pm1} - g_i =& g_{\Gamma^\pm} \pm (1-\alpha)hg'_{\Gamma^\pm} + \frac{(1-\alpha)^2h^2}{2}g''_{\Gamma^\pm} - g_{\Gamma^\mp} \pm \alpha h g'_{\Gamma^\mp} - \frac{\alpha^2h^2}{2}g''_{\Gamma_\mp} + O(h^3),\\
=& \pm n[g] + (1-\alpha)hn\left[g'\right] \pm (1-\alpha)hg'_{\Gamma^\mp} \pm \frac{(1-\alpha)^2h^2}{2}n\left[g''\right] + \frac{(1-\alpha)^2h^2}{2}g''_\Gamma \pm\\
& \alpha h g'_{\Gamma^\mp} - \frac{\alpha^2h^2}{2}g''_{\Gamma^\mp} + O(h^3),\\
=& \pm n[g] + (1-\alpha)hn\left[g'\right] \pm \frac{(1-\alpha)^2h^2}{2}n\left[g''\right] \pm hg'_{\Gamma^\mp} + \frac{(1-2\alpha)h^2}{2}g''_{\Gamma^\mp} + O(h^3),\\
=& \pm n[g] + (1-\alpha)hn\left[g'\right] \pm \frac{(1-\alpha)^2h^2}{2}n\left[g''\right] \pm \left(hg'_i \pm \alpha h^2 g''_i\right) + \frac{(1-2\alpha)h^2}{2}g''_i + ...,\\
=& \pm n[g] + (1-\alpha)hn\left[g'\right] \pm \frac{(1-\alpha)^2h^2}{2}n\left[g''\right] \pm hg'_i + \frac{h^2}{2}g''_i + O(h^3),
\end{aligned}$$

with unknown g'_i and g''_i; if there is no boundary in between $g_{i\pm1}$ and g_i, we have

$$g_{i\pm1} - g_i = \pm hg'_i + \frac{h^2}{2}g''_i + O(h^3).$$

Thus, in the one-dimensional case, we have two equations from $\pm$, and two unknowns g'_i and g''_i. This allows us to solve for g''_i. In our case of jump conditions, we have

$$g_{i\pm1} - g_i = \mp\frac{(1-\alpha)^2h^2}{2}n \pm hg'_i + \frac{h^2}{2}g''_i + O(h^3),$$

or

$$g_{i\pm1} - g_i = \pm hg'_i + \frac{h^2}{2}g''_i + O(h^3).$$

This means if there is a boundary point $x_\Gamma \in (x_{i-1}, x_{i+1})$,

$$(g_{i+1} - g_i) + (g_{i-1} - g_i) = h^2 g''_i + h^2\left(A_i^+ + A_i^-\right) + O(h^3),$$

where

$$A_i^{\pm} = \begin{cases} \mp\frac{(1-\alpha)^2}{2} n & \text{, if interface point lies between } x_i, x_{i\pm1} \\ 0 & \text{, otherwise.} \end{cases}$$

So

$$\frac{g_{i+1} - 2g_i + g_{i-1}}{h^2} = H_\Omega(x_i) + A_i^+ + A_i^- + O(h). \tag{4.10}$$

Notice that, on the left hand side of (4.10) is the central difference approximation for the second derivative of g at x_i, denoted as $g_h''(x_i)$. The $O(h)$ error term is only nonzero when x_i is an irregular point. Otherwise on regular point,

$$g_h''(x_i) = H_\Omega(x_i) + O(h^2). \tag{4.11}$$

Combining (4.10) and (4.14),

$$g_h''(x_i) = H_\Omega(x_i) + A_i^+ + A_i^- + O_I(h) + O(h^2), \tag{4.12}$$

where $O_I(h)$ term is only nonzero at irregular points. This is the discrete form of (4.2), and the right hand side represents the discrete Heaviside function plus error terms.

In higher dimensions $\mathbb{R}^d$, if x_i is an irregular point, then

$$g_{i\pm e_j} - g_i = \pm\frac{n_j}{|n_j|}[g] + (1-\alpha)h\frac{n_j}{|n_j|}\left[g_{x_j}\right] \pm \frac{(1-\alpha)^2h^2}{2}\frac{n_j}{|n_j|}\left[g_{x_jx_j}\right] \pm h\left(g_{x_j}\right)_i + \frac{h^2}{2}\left(g_{x_jx_j}\right)_i + ..,$$

where e_j denotes the unit standard vector, and g_{x_j} denotes partial derivative of g in the direction of e_j. Otherwise when x_i is a regular point,

$$g_{i\pm e_j} - g_i = \pm h\left(g_{x_j}\right)_i + \frac{h^2}{2}\left(g_{x_jx_j}\right)_i + O(h^3).$$

Assuming the jump terms can be calculated, this leads to $2d$ equations and the $2d$

unknowns $\left(g_{x_j}\right)_i, \left(g_{x_j x_j}\right)_i$. Using better notation, with $s = \pm 1$ and noting

$$s\frac{n_j}{|n_j|} = -\operatorname{sign}(\phi_i),$$

we can write

$$\begin{aligned} g_{i+se_j} - g_i =& -\operatorname{sign}(\phi_i)[g] - s(1-\alpha)h\operatorname{sign}(\phi_i)\left[g_{x_j}\right] - \frac{(1-\alpha)^2h^2}{2}\operatorname{sign}(\phi_i)\left[g_{x_j x_j}\right] + \\ & \operatorname{sh}\left(g_{x_j}\right)_i + \frac{h^2}{2}\left(g_{x_j x_j}\right)_i + O(h^3) \\ =& -\operatorname{sign}(\phi_i)\left([g] + s(1-\alpha)h\left[g_{x_j}\right] + \frac{(1-\alpha)^2h^2}{2}\left[g_{x_j x_j}\right]\right) + \\ & \operatorname{sh}\left(g_{x_j}\right)_i + \frac{h^2}{2}\left(g_{x_j x_j}\right)_i + O(h^3), \end{aligned}$$

when there is an interface.

Note that

$$\left[\frac{\partial g}{\partial n}\right] = [\nabla g \cdot n] = [\nabla g] \cdot n,$$

so

$$[\nabla g] \cdot n = 0.$$

Also, $[g] = 0$ implies

$$0 = [\nabla g \cdot \tau] = [\nabla g] \cdot \tau$$

for any $\tau \cdot n = 0$. Altogether, this means

$$[\nabla g] = 0.$$

Finally, note

$$[\Delta g] = -1,$$

and since $[\nabla g \cdot \tau] = 0$ for all $\tau \cdot n = 0$,

$$[\nabla(\nabla g \cdot \tau) \cdot \eta] = 0$$

for all τ, η orthogonal to n, and

$$[\nabla(\nabla g \cdot n) \cdot \eta] = 0$$

as well. This means, since $[\nabla g] = 0$, that

$$\eta^T \left[\nabla^2 g\right] \tau = 0,$$
$$\eta^T \left[\nabla^2 g\right] n = 0.$$

Now let

$$\left\{\tau^{(1)}, \ldots, \tau^{(d-1)}, n\right\}$$

be an orthogonal basis for $\mathbf{R}^d$. Note there are

$$1 + 2 + \cdots + d = \frac{d(d+1)}{2}$$

unknowns in the upper triangular portion of $\left[\nabla^2 g\right]$, and

$$(d-1) + (d-2) + \cdots + 1 + (d-1) + 1 = \frac{d(d+1)}{2}$$

equations. The equations can be labeled as the following:

- For $1 \leq k \leq \ell \leq d-1$,

$$\sum_i \tau_i^{(k)} \tau_i^{(\ell)} \left[g_{x_i x_i}\right] + \sum_{i<j} \left(\tau_i^{(k)} \tau_j^{(\ell)} + \tau_j^{(k)} \tau_i^{(\ell)}\right) \left[g_{x_i x_j}\right] = 0;$$

- For $1 \leq k \leq d-1$,

$$\sum_i \tau_i^{(k)} n_i \left[g_{x_i x_i}\right] + \sum_{i<j} \left(\tau_i^{(k)} n_j + \tau_j^{(k)} n_i\right) \left[g_{x_i x_j}\right] = 0;$$

- And,

$$\sum_i \left[g_{x_i x_i}\right] = -1.$$

Observe that, when

- For $1 \leq i < j \leq d$, $\left[g_{x_i x_j}\right] = -2 n_i n_j$;
- For $1 \leq i \leq d$, $\left[g_{x_i x_i}\right] = -n_i^2$.

we have

$$\left[\nabla^2 g\right] = n n^T,$$

so

$$\eta^T \left[\nabla^2 g\right] v = \eta^T n n^T v = 0$$

for all $\eta \cdot n = 0$, and the observation is the solution. This means

$$g_{i+se_j} - g_i = \operatorname{sign}(\phi_i) \frac{(1-\alpha)^2 h^2}{2} n_j^2 + \operatorname{sh}\left(g_{x_j}\right)_i + \frac{h^2}{2} \left(g_{x_j x_j}\right)_i + O(h^3),$$

if there is an interface point in between. And it follows that

$$g_{i+e_j} - 2g_i + g_{i-e_j} = h^2 \left(g_{x_j x_j}\right)_i + h^2 \left(A_{ij}^+ + A_{ij}^-\right) + O(h^3),$$

where

$$A_{ij}^{\pm} = \begin{cases} \operatorname{sign}(\phi_i) \frac{(1-\alpha)^2}{2} n_j^2 & \text{, if interface point lies between } x_i, x_{i \pm e_j} \\ 0 & \text{, else.} \end{cases}$$

So

$$\sum_{j=1}^{d} \frac{g_{i+e_j} - 2g_i + g_{i-e_j}}{h^2} = H_\Omega(x_i) + \sum_{j=1}^{d} \left(A_{ij}^{+} + A_{ij}^{-}\right) + O(h). \tag{4.13}$$

On the left hand side of (4.13) is the standard center differenced approximation for the Laplacian of g at x_i, denoted as $\Delta_h g_i$. The $O(h)$ error term is only nonzero when x_i is an irregular point. Otherwise on regular point,

$$\Delta_h g_i = H_\Omega(x_i) + O(h^2). \tag{4.14}$$

Therefore,

$$\Delta_h g_i = H_\Omega(x_i) + \sum_{j=1}^{d} \left(A_{ij}^{+} + A_{ij}^{-}\right) + O_I(h) + O(h^2), \tag{4.15}$$

where $O_I(h)$ term is only nonzero at irregular points. This is the discrete form of (4.1), and the right hand side represents the discrete Heaviside function plus error terms.

4.4 Accuracy and Numerical results

From the previous analysis, we consider the accuracy from

$$\begin{aligned}
V &= \int_D H(\psi(\mathbf{x}))\, d\mathbf{x}, \\
&= \int_D \Delta g \, d\mathbf{x}, \\
&= \sum_i h^d \Delta_h g_i + O(h^2), \\
&= \sum_i h^d (\tilde{H}(\psi_i) + O_I(h) + O(h^2)) + O(h^2), \\
&= \sum_i h^d \tilde{H}(\psi_i) + O(h^2),
\end{aligned}$$

since there are $O(h^{-(d-1)})$ irregular points and $O_I(h)$ only occurs at irregular points. In practice, $A_{ij}^{\pm}$ from the $\tilde{H}(\psi_i)$ will be approximated to some level of accuracy in h, and that results from the error approximating α and n_j. Since this error only occurs at irregular points as well, as long as n_j is approximated up to first order on the interface, and α with $O(1)$ accuracy, the overall

accuracy would be of $O(h^2)$.

With

$$n_j = -\frac{D_{e_j}\psi_i}{||\nabla^\varepsilon \psi_i||}, \tag{4.16}$$

where

$$D_{e_j}\psi_i = \frac{\psi_{i+e_j} - \psi_{i-e_j}}{2h}, \quad ||\nabla^\varepsilon \psi_i|| = \sqrt{\sum_j (D_{e_j}\psi_i)^2 + \varepsilon},$$

and with

$$\alpha = \left|\frac{\psi_{i-1}}{D^-_{e_j}\psi_i}\right| \cdot \frac{1}{h},$$

where

$$D^-_{e_j}\psi_i = \frac{\psi_i - \psi_{i-e_j}}{h},$$

as generated from the 2D case analysis of Smereka's Paper [26], n_j and α are up to the required order of accuracy at the interface, which makes the Heaviside approximation second order accurate. In practice, $||\nabla\psi_i||$ tends not to be close to 0 at most of the irregular points around the interface, so (4.16) is usually replaced by

$$n_j = -\frac{D_{e_j}\psi_i}{\max(||\nabla\psi_i||, \varepsilon)}$$

instead for better accuracy result.

In a 3D experiment, the Heaviside function is used to approximate the volume of a ball centered at $(0,0,0)$ with radius 6 in the domain of $[-9,9]\times[-9,9]\times[-9,9]$. A loglog regression is done based on a data set with N from 30 to 300. The result is shown in Figure 4.1. It indicates an order less than 2, but still better than a first order approach.

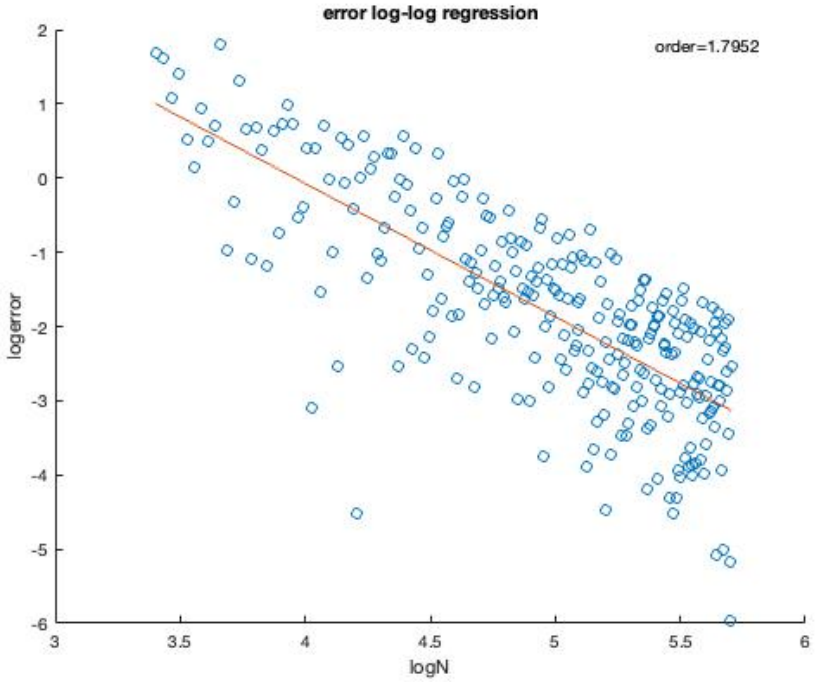


Figure 4.1. Accuracy of the Heaviside Approximation in approximating volume of a ball in 3D.

Chapter 5

Conclusion and Future Research

This study addresses the visibility problem within the level set framework. A novel grid-based algorithm is proposed, striking a balance between computational efficiency and high accuracy, along with a new method for approximating the Heaviside function to improve accuracy in evaluating the volume of visible regions and optimizing related visibility calculations. At the mean time, several aspects of the proposed methodology warrant further discussion and exploration to optimize its performance and broaden its applicability.

The phase flow techniques currently employed in the algorithm use a fixed base number B. While this approach simplifies the computational process and reduces complexity, it imposes a limitation on the algorithm's ability to achieve its full potential in terms of accuracy. Future research should explore the impact of fixing the stage number S instead. This adjustment might lead to a more refined phase flow representation, enhancing the resolution of visibility boundaries without significantly increasing computational costs.

The growing method employed in this study demonstrates excellent localized computation near visibility interfaces. However, its compatibility with multi-resolution representations remains unexplored. Multi-resolution approaches have the potential to enhance computational efficiency by dynamically adjusting resolution based on local complexity. Investigating how the growing method can be integrated with such representations could unlock further efficiencies, particularly for environments with heterogeneous visibility characteristics.

The cube structure used in the algorithm plays a critical role in querying visibility information. The structure could be further optimized to enhance the algorithm's query speed and computational efficiency, especially in higher dimensions.

A limitation of the current method lies in its storage requirements. The algorithm's reliance on high-resolution grids and auxiliary structures for accurate computations can lead to substantial memory usage. Addressing this issue requires designing more efficient data structures that strike a balance between storage efficiency and computational performance. Techniques such as sparse data representations or hierarchical grids may offer promising directions for improvement.

The current Heaviside approximation approach is conjectured to achieve second-order accuracy, which suggests a promising balance between computational efficiency and precision. While preliminary results indicate favorable performance, rigorous theoretical analysis is necessary to establish the conjectured accuracy and validate the approach in diverse scenarios.

Bibliography

[1] Beatriz Flamia Azevedo, Ana Maria AC Rocha, and Ana I Pereira. Hybrid approaches to optimization and machine learning methods: a systematic literature review. *Machine Learning*, pages 1–43, 2024.

[2] Robert Bodor, Andrew Drenner, Paul Schrater, and Nikolaos Papanikolopoulos. Optimal camera placement for automated surveillance tasks. *Journal of Intelligent and Robotic Systems*, 50:257–295, 2007.

[3] Di Chen, Prashant Kumar, Yukinori Kametani, and Yosuke Hasegawa. Multi-objective topology optimization of heat transfer surface using level-set method and adaptive mesh refinement in openfoam. *International Journal of Heat and Mass Transfer*, 221:125099, 2024.

[4] Li-Tien Cheng and Yen-Hsi Tsai. Visibility optimization using variational approaches. *Communications in Mathematical Sciences*, 3(3):425–451, 2005.

[5] Satyan Coorg and Seth Teller. Real-time occlusion culling for models with large occluders. In *Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 83–ff, 1997.

[6] Mark De Berg. *Computational geometry: algorithms and applications*. Springer Science & Business Media, 2000.

[7] Leila De Floriani, Enrico Puppo, and Paola Magillo. Geometric structures and algorithms for geographical information systems. *Handbook on Computational Geometry. Elsevier Science*, page 6, 1998.

[8] Björn Engquist, Anna-Karin Tornberg, and Richard Tsai. Discretization of dirac delta functions in level set methods. *Journal of Computational Physics*, 207(1):28–51, 2005.

[9] Subir Kumar Ghosh. *Visibility algorithms in the plane*. Cambridge university press, 2007.

[10] Kurt Glasner. The method of characteristics, 2009. Accessed from University of Arizona Department of Mathematics.

[11] Aaron Hertzmann and Denis Zorin. Illustrating smooth surfaces. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 517–526, 2000.

[12] Ibrahim Ibrahim, Joris Gillis, Wilm Decré, and Jan Swevers. An efficient solution to the 2d visibility problem in cartesian grid maps and its application in heuristic path planning. *arXiv preprint arXiv:2403.06494*, 2024.

[13] Chiu-Yen Kao and Richard Tsai. A level set formulation for visibility and its discretization. *UCLA CAM report*, 2006.

[14] Chiu-Yen Kao and Richard Tsai. Properties of a level set algorithm for the visibility problems. *Journal of Scientific Computing*, 35(2):170–191, 2008.

[15] Luis Ángel Larios-Cárdenas and Frederic Gibou. A hybrid inference system for improved curvature estimation in the level-set method using machine learning. *Journal of Computational Physics*, 463:111291, 2022.

[16] The MathWorks, Inc., Natick, Massachusetts, USA. *MATLAB Documentation*, 2024. Available at https://www.mathworks.com/help/matlab/.

[17] Anita Mayo. The fast solution of poisson's and the biharmonic equations on irregular regions. *SIAM Journal on Numerical Analysis*, 21(2):285–299, 1984.

[18] Anita Mayo. The rapid evaluation of volume integrals of potential theory on general regions. *Journal of Computational Physics*, 100(2):236–245, 1992.

[19] Joseph O'Rourke. *Art Gallery Theorems and Algorithms*, chapter 7: Visibility Graphs. Oxford University Press, 1987. Accessed: 2024-12-07.

[20] Stanley Osher, Ronald Fedkiw, and K Piechor. Level set methods and dynamic implicit surfaces. *Appl. Mech. Rev.*, 57(3):B15–B15, 2004.

[21] Nikos K Paragios and Rachid Deriche. A pde-based level-set approach for detection and tracking of moving objects. In *Sixth International Conference on Computer Vision (IEEE Cat. No. 98CH36271)*, pages 1139–1145. IEEE, 1998.

[22] Danping Peng, Barry Merriman, Stanley Osher, Hong Kai Zhao, and Myungjoo Kang. A pde-based fast local level set method. *Journal of Computational Physics*, 155(2):410–438, 1999.

[23] Bernt Schiele and James L Crowley. A comparison of position estimation techniques using occupancy grids. *Robotics and autonomous systems*, 12(3-4):163–171, 1994.

[24] James A Sethian and David Adalsteinsson. An overview of level set methods for etching, deposition, and lithography development. *IEEE Transactions on Semiconductor Manufacturing*, 10(1):167–184, 1997.

[25] Deepak Singh, Helmer André Friis, Espen Jettestuen, and Johan Olav Helland. Adaptive mesh refinement in locally conservative level set methods for multiphase fluid displacements in porous media. *Computational Geosciences*, 27(5):707–736, 2023.

[26] Peter Smereka. The numerical approximation of a delta function with application to level set methods. *Journal of Computational Physics*, 211(1):77–90, 2006.

[27] John D Towers. Finite difference methods for approximating heaviside functions. *Journal of Computational Physics*, 228(9):3478–3489, 2009.

[28] Y-HR Tsai, L-T Cheng, Stanley Osher, Paul Burchard, and Guillermo Sapiro. Visibility and its dynamics in a pde based implicit framework. *Journal of Computational Physics*, 199(1):260–290, 2004.

[29] Yen-Hsi Richard Tsai, Li-Tien Cheng, Stanley Osher, and Hong-Kai Zhao. Fast sweeping algorithms for a class of hamilton–jacobi equations. *SIAM journal on numerical analysis*, 41(2):673–694, 2003.

[30] John N Tsitsiklis. Efficient algorithms for globally optimal trajectories. *IEEE transactions on Automatic Control*, 40(9):1528–1538, 1995.

[31] Unknown. Butterfly shape. https://www.pictorem.com/261974/Butterfly%20Shape.html, 2024. Accessed: 2024-12-04.

[32] Arthur Van Bilsen and Ronald Poelman. 3d visibility analysis in virtual worlds: the case of supervisor. In *Proceedings of construction applications of virtual reality (CONVR) 2009 conference Sydney*, pages 5–6, 2009.

[33] Wikipedia contributors. Visibility polygon — wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Visibility_polygon, 2024. Accessed: 2024-12-07.

[34] Kai M Wurm, Armin Hornung, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. Octomap: A probabilistic, flexible, and compact 3d map representation for robotic systems. In *Proc. of the ICRA 2010 workshop on best practice in 3D perception and modeling for mobile manipulation*, volume 2, page 3, 2010.

[35] Lexing Ying and Emmanuel J Candes. The phase flow method. *Journal of Computational Physics*, 220(1):184–215, 2006.

[36] Sara Zahedi and Anna-Karin Tornberg. Delta function approximations in level set methods by distance function extension. *Journal of Computational Physics*, 229(6):2199–2219, 2010.