Proceedings of CONF-SEML 2026 Symposium: Multimodal Data Acquisition: Applications in Physiological and Behavioral Research
DOI: 10.54254/2755-2721/2026.AD31573

A Survey on 2D Visibility Algorithms: Ray Casting, Rectangle-Based FOV and Recursive Shadowcasting

Wenjun Huang[1*], Ziming Weng[1]

[1]University of Nottingham, Ningbo, China
**Corresponding Author. Email: 3108879135@qq.com*

Abstract. Field of view (FOV) algorithms are essential in determining the visible area of a player in 2D games. These algorithms dynamically calculate the visible areas while occluding these hidden areas, and play an important role in games such as roguelikes and stealth games. This survey summarizes three 2D FOV algorithms: ray casting, rectangle-based FOV, and recursive shadowcasting. The ray casting algorithm casts rays to determine which area was hidden from the player, which is a basic FOV algorithm. Rectangle-based FOV optimizes computation for large 2D grids by representing obstacles as rectangles, also using a quadtree to improve the access speed. Recursive shadowcasting efficiently computes the visible area by dividing the grid into 8 octants and recursively splitting the view when obstacles are encountered. This survey also mentioned how to adapt the recursive shadowcasting algorithm to 2.5D and 3D environments.

Keywords: FOV algorithms, 2D games, Ray casting, Recursive Shadowcasting

1. Introduction

Field of view (FOV) algorithms are critical in 2-dimensional game development, determining which area or object should be displayed on the game screen. These algorithms simulate real-world vision by calculating visible areas in the map dynamically, and display them on the screen if it is not blocked by any vision-blocking cells. Video games often use a FOV grid, in order to calculate a large block of area at once to reduce the overhead of computing [1]. FOV algorithms play a critical role in affecting player's strategy and immersion, especially for games such as roguelikes or stealth games.

These algorithms have historically been discussed in isolation–technical papers often focus on the details in the implementation of the algorithm, while academic papers focus on broader computational geometry contexts. This paper focuses on three algorithms: ray casting, rectangle-based FOV, and recursive shadowcasting.

Ray casting is a foundational approach to calculate visible area, through casting rays in different directions, and if any vision-blocking object blocks the ray, areas behind that object will be marked as not visible. As for Rectangle-based FOV, it optimizes the performance in a large 2D grid map by abstracting obstacles into rectangles and using quadtree data structures to improve searching speed. And recursive shadowcasting is an efficient algorithm in both 2D and 3D environments by

© 2026 The Authors. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (https://creativecommons.org/licenses/by/4.0/).

DOI: 10.54254/2755-2721/2026.AD31573

partitioning the view into angular sectors, and recursively tracking shadow boundaries to determine the visibility.

This survey aims to provide an overview of these algorithms, introduce their core principles and basic implementation details, analyses the applicable scope of these algorithms, and provide a brief introduction to these algorithms for 2D game developers.

2. Background

The field of view is defined as the set of all positions in a game environment that are observable from a specific viewpoint. Its primary functionality is to simulate the visual perception of the real world. FOV for games would dynamically determine which area or object is visible to the viewpoint, often the player, and enables games to simulate the logic of "line of sight". However, a trade-off exists between accuracy and performance, where accuracy determines how accurately the algorithm simulates the occlusion for each area in the game map, which will certainly affect the player's immersion and decision, and performance reflects the extra overhead of calculating the visible area.

Additionally, certain phenomena observed in grid-based FOV algorithm, such as pillar shadows, corner peeking, and diagonal wall handling, illustrate the behavior differences be- tween algorithms approaches rather than representing computational error. For instance, single-cell pillars produce varying shadow angles depending on the algorithm, corner peeking may or may not occur at T-junctions, and visibility through diagonal walls differs according to the implementation of the algorithm [2].

FOV algorithms can be categorized based on their computational logic, for instance, how they represent spatial information, how they detect occlusion interaction, and how they perform structure calculations. This reflects differences in how these algorithms address the accuracy-performance trade- off, which can be a detailed comparison framework between them. However, the performance of an FOV algorithm largely depends on the environment in which it is applied; various edge cases may occur in certain scenarios. This paper focuses specifically on a 2D grid environment.

3. Ray casting

Ray casting is a fundamental visibility algorithm widely used in 2D games, which is always used to determine the player's FOV [3]. The basic idea of it is simple: starting from a given viewpoint, rays are cast in all directions, and the first obstacle encountered by each ray marks the boundary of visibility. This method effectively approximates the FOV by checking collisions at discrete angles, making it fast and practical in grid-based environments like dungeon exploration games. Since the traditional ray casting algorithm samples visibility at fixed angle intervals, it may overlook small gaps between obstacles or produce artifacts like 'shadow spikes' due to ray distribution. Besides, it relies on grid traversal (e.g., the Bresenham algorithm), making it difficult to achieve precise geometric visibility in polygonal spaces, where precise line-of-sight is more critical than grid-based approximations. A rotational plane sweep algorithm is proposed to address certain limitations of grid-based ray casting in polygonal envi- ronments which is introduced in Ana Batinovic et al's study [4]. The algorithm simulates a continuous angular sweep around the viewpoint, dynamically tracking obstacle edges as they intersect the rotating line of sight. This approach can be viewed as a specialized form of ray casting in polygonal settings, designed to guarantee accurate visibility computations between exact polygon vertices. However, this precision may cause more cost: the

DOI: 10.54254/2755-2721/2026.AD31573

algorithm's reliance on angular sorting and dynamic edge updates makes it more computationally intensive than traditional ray casting.

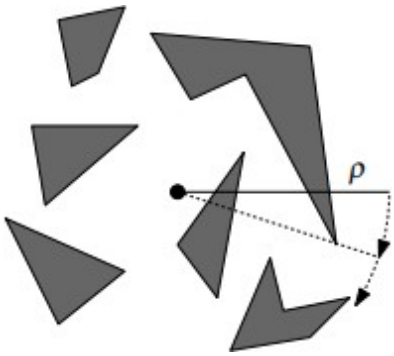

Figure 1. Rational plane sweep algorithm [5]

A side-by-side analysis of these two approaches indicates that ray casting functions as an efficient and widely implemented technique within the gaming sector, emphasizing real- time performance rather than the higher accuracy achievable through rotational sweep methods. As a result, ray casting continues to be the prevailing choice for 2D game development, owing to its computational economy and ease of integration. That said, its inherent constraints in terms of precision and adaptability have motivated the creation of more specialized visibility algorithms for use cases requiring exacting line-of- sight determination.

4. Rectangle-based field of vision

In Evan et al's study, an algorithm called rectangle-based FOV was proposed to increase the computing speed of visible areas in the FOV grid [1]. Due to the game environment changes rather infrequently, using rectangles to represent the vision blocking cells is a more compact representation, and the performance of the visible area computation for a 2D FOV grid is less dependent on the grid size than the ray casting algorithms mentioned in the previous section.

Rectangle-based FOV uses a quadtree to represent the FOV grid to increase the access speed of the grid. For a 2D FOV grid, a quadtree divides the grid into 4 quadrants, each of which serves as a node of the root, and then repeats this process until the entire map is stored in the quadtree. The root node of the quadtree represents the entire FOV grid and each internal node of the quadtree has exactly 4 children, each representing one quadrant of the space that its parent node represents. There will be some situations where the FOV grid cannot be divided into four regions that contain exactly the same number of rectangles, so each node can store more than one rectangle based on the predefined number N. When a node contains fewer than N rectangles, that node will be considered as a leaf node.

In a rectangle-based FOV, whether a cell can be considered visible is based on the definition it used. In Evan et al's study, a cell is considered as visible when any point in that cell is visible from the FOV source, which matches the definition of visibility used in the recursive shadowcasting algorithm [1]. Instead of determining what areas are visible from the FOV source, a rectangle-based FOV focuses on what areas are occluded behind the obstacles.

Rectangle-based FOV will consider all vision blocking rectangles independently, each select two visible vertices from the aspect of the FOV source and has the furthest distance apart from each other among the visible vertices to the FOV source. Then it will cast a ray from the selected vertices

DOI: 10.54254/2755-2721/2026.AD31573

in the opposite direction of the FOV source, as shown in Figure 2. A cell is considered invisible if it is entirely within the vision-blocking area.

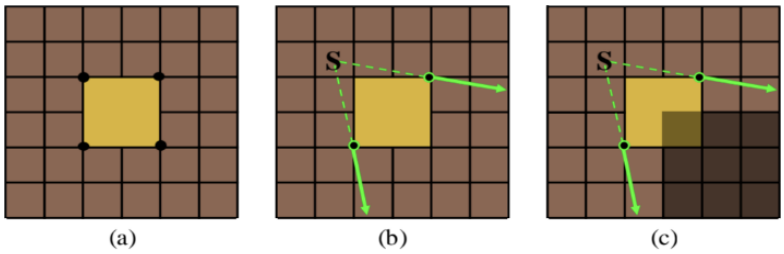

Figure 2. Ray casting in rectangle-based FOV [1]

When rectangles share a common side, certain cells would be labelled as visible incorrectly, as shown in Figure 3. The algorithm will consider each rectangle separately, so rectangles R1 and R2 will be processed in isolation. For cell i and cell ii, the ray casts from R1 and R2 touched them, so they were labelled as visible by the rectangle-based FOV (c). To address the issue, the algorithm will slightly increase the size of one of the two adjacent rectangles. For rectangle x, it belongs to R1 when the algorithm starts to compute the vision blocking areas of R1, and also belongs to R2 when its vision blocking areas are calculated. The overlap rectangle x ensures the vision- blocking areas can be calculated correctly.

After the rectangles had been extended, the visibility status of each grid cell was assigned. Initially, all the cells were assigned as visible, and the algorithm goes through all the vision-blocking rectangles one by one, setting the grid cells to invisible between two ray casts from left to right for each row.

Arranging the vision-blocking rectangles before calculating is an optimization of this algorithm. The algorithm will process the rectangles from the nearest to the furthest to reduce the redundant work of recalculating the vision-blocking areas. If a vision-blocking area is fully occluded by other rectangles, then it will be ignored during the process of the algorithm.

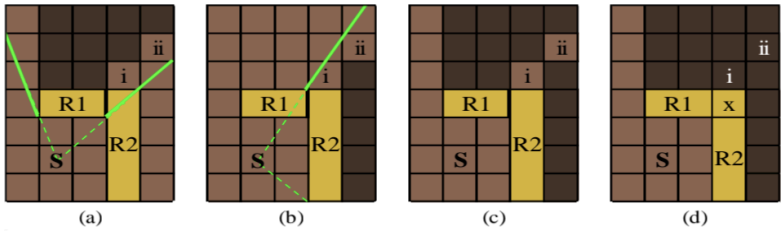

Figure 3. Issue with the algorithm [1]

This algorithm performs exceptionally well in large-scale 2D grid environments. Compared to the ray casting algorithm, it significantly reduces the number of rays, eliminating a great deal of redundant computations. In Evan et al's study, data have shown that rectangle-based FOV's performance in large 2D grid environments is better than recursive shadowcasting algorithms [1]. This is largely attributed to its approach of abstracting obstacles into rectangles and utilizing quadtrees for spatial representation.

Despite its superior performance in large 2D grid environments, the algorithm has limited applicability in other environments. Therefore, it is a favorable choice for 2D games that require visibility algorithms for large grid maps and have high performance demands.

DOI: 10.54254/2755-2721/2026.AD31573

5. Recursive shadowcasting

In Ana Batinovic et al's study, the RSC algorithm is introduced and adapted for 2.5D slicewise processing, cuboid-based edge evaluation, efficient voxel visibility computation, dead-end resolution, and low- complexity gain calculation [4]. This section introduces RSC in a 2D grid and focuses on 2.5D slice-wise processing.

RSC is widely used in lots of 2D grid-based games to calculate the FOV from a specific point. Players navigate a grid-based environment with various obstacles in these games. It is essential to quickly determine the visible area of players in any moment for a gaming experience. It is also difficult to calculate FOV when there exist lots of obstacles or dynamic changes without redundant calculations. However, the RSC provides a useful method that can not only handle complex environments with minimal calculations but also be used in dynamic environments.

RSC divides the grid into eight symmetrical octants, as shown in Figure 4a. Each octant represents a direction sector from viewpoint of the observer. RSC processes these octants independently by casting light rays from the observer to determining visible grid cells. The light rays traverse the octant row by row or column by column , as shown in the Figure 4a. When light rays traverse an octant without detecting any blocked cell, the light cone expands progressively, as show in the Figure 4b. Once a blocked cell (e.g., a wall) is detected, RSC splits the current octant into two parts which are defined by the start and end slopes of the light rays that touch the corners of the blocked cell. It then recursively processes the left part until it reaches the octant boundary, as shown in Figure 4c. Note that even if a ray only grazes the corner of a cell, the entire cell is set to be visible. The result of applying RSC in a grid is shown in the Figure 4d.

The key innovation in RSC is its recursive nature, where visibility in each direction is refined by recursively splitting the current field of view into smaller parts when an obstacle are encountered. This method minimizes redundant checks and is highly efficient, particularly in environments with complex obstacle layouts.

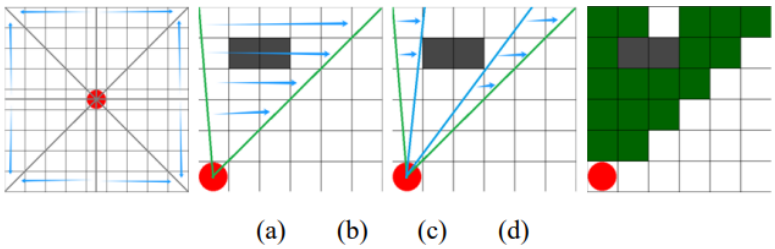

Figure 4. Recursive shadowcasting in a 2D grid

Building on prior research in the field, Ana Batinovic et al demonstrate a novel 2.5D Recursive Shadowcasting (RSC) algorithm designed for three-dimensional exploration by unmanned aerial vehicles (UAVs) [4]. The proposed approach segments the 3D environment into vertically arranged 2D slices, executes the established 2D RSC visibility algorithm separately on each slice, and synthesizes the outcomes to produce a comprehensive 3D visibility model. Several technical innovations underpin the method's effectiveness: adaptive adjustment of inter-slice spacing in response to environmental density, computational bounding via cubic constraints, and the maintenance of cross- slice visibility consistency through occlusion-pattern buffering. By reducing the 3D problem into a series of tractable 2D computations, this strategy confines time complexity to approximately $O((mn \log n)/r\ N_{slices})$, permitting real-time 3D visibility analysis while preserving the

DOI: 10.54254/2755-2721/2026.AD31573

well-documented efficiency of 2D RSC—an essential feature for UAV platforms operating under stringent computational constraints.

Significant performance gains reported in the study are attributed to two principal advances. First, a dynamic slice- spacing mechanism tailors computational load according to the spatial distribution of obstacles. Second, a visibility- propagation technique employing inter-slice buffers substantially reduces redundant processing. Empirical evaluations indicate that this 2.5D framework operates roughly an order of magnitude faster than conventional 3D ray casting, all while maintaining perceptual and functional completeness in explored environments. These characteristics make it a viable candidate for real-time 3D perception tasks on resource- limited autonomous systems.

Though originally intended for robotic exploration, the slice-based 2.5D RSC methodology offers promising extensions to interactive media and game development. Extending efficient 2D visibility analysis across vertical layers can optimize 3D field-of-view (FoV) computations in voxel- based worlds, procedurally generated dungeons, and real-time strategy maps. Such an adaptation could support performant implementations of fog-of-war, dynamic illumination, and AI- driven vision subsystems, all while retaining the computational economy inherent to the RSC approach. By blending dimensional abstraction with spatial fidelity, this 2.5D paradigm provides developers with a pragmatic alternative—striking a functional equilibrium between the precision of full 3D ray tracing and the speed of purely 2D shadow casting. Its utility is especially pronounced in game environments featuring destructible terrain, multi-level architecture, or large- scale procedural generation, where real-time performance is often a critical design constraint.

6. Conclusion

Selecting a field-of-view algorithm in 2D game development typically involves weighing several competing factors: precision, computational cost, and alignment with the game's specific interactive requirements. Ray casting, rectangle-based FoV, and recursive shadowcasting each address visibility through distinct computational models, and their respective strengths fundamentally shape player experience and gameplay mechanics.

Ray casting supports high-fidelity line-of-sight determination, yet incurs significant computational overhead. This method remains indispensable in contexts demanding exact visibility simulation—such as tactical stealth games—where physically plausible occlusion and environmental awareness justify the performance trade-off.

Rectangle-based FoV approaches emphasize execution speed within grid-aligned environments, making them well- suited to classic roguelikes and tile-based RPGs. By leveraging quadtrees for spatial representation and concentrating on occlusion relationships rather than per-cell visibility, these methods reduce dependency on map scale and excel in largely static, regularly structured worlds. However, their reliance on grid topology inherently limits applicability to games with free-form or continuous spatial representations, even when enhanced with adjacency optimizations.

Recursive Shadowcasting emerges as a versatile choice for 2D grid-based games, particularly procedurally generated roguelikes, due to its efficiency in handling complex obstacle layouts and large environments. By decomposing visibility into octants and using recursive region splitting, the dependence on grid size is reduced, ensuring real-time performance even in dynamic or expansive worlds. Moreover, its extension to 2.5D via slice-wise processing unlocks potential in 3D game development from voxel-based games to massive open worlds, where it balances computational complexity and visibility accuracy, supporting features like dynamic lighting and AI vision in resource-constrained scenarios.

DOI: 10.54254/2755-2721/2026.AD31573

Acknowledgment

Our team would like to express our sincere gratitude to Professor William Nace for his guidance, suggestions, consistent support, reviewing the draft, and providing feedback throughout the project. We also wish to thank our Teaching Assistant, Vicky, for her assistance, clarifying core concepts, and providing timely feedback for this survey.

Wenjun Huang and Ziming Weng contributed equally to this work and should be considered co-first authors.

References

[1] Evan R.M. Debenham et al. "Efficient Field of Vision Algorithms for Large 2D Grids". In: International Journal of Computer Science and Information Technology (2021). DOI: 10.5121/ijcsit.2021.13101.

[2] J. C. Wilk. Comparative study of field of view algorithms for 2D grid based worlds. RogueBasin. Avail- able online: https: //www.roguebasin.com/index.php/Comparative study of field of view algorithms for 2D grid based worlds, accessed on 2026-01-14. 2009.

[3] J. C. Wilk. Ray casting. RogueBasin. Available online: https: //roguebasin.com/index.php/Ray casting, accessed on 2026-01-14. 2009.

[4] Ana Batinovic et al. "A Shadowcasting-Based Next-Best- View Planner for Autonomous 3D Exploration". In: IEEE robotics automation letters (2022). DOI: 10.1109/lra.2022.3146586.

[5] "Computational geometry: Algorithms and applications". eng. In: Computers mathematics with applications (1987) 36.6 (1998), p. 139. ISSN: 0898-1221.