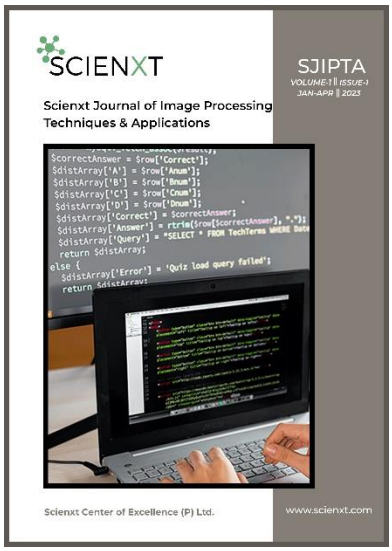

Computing visual field of vision on 2.0-grid systems using new algorithms

Pawan Kumar Bajwa[1], Harsh Girdharilal[*2]

Government Engineering College, Nilokhari, Karnal, Haryana

Email: harsh_girhdar.657@gmail.com

Corresponding Author: Harsh Girdharilal

Abstract:

Determining if an object can be seen from another is critical in a lot of video games. Checking an individual's view of a scene in a video game can be described as having a "field of vision" (FOV). Using a FOV, it is possible to easily measure the visibility of many items from a particular location. It analyses the constraints of the existing methods for FOV computing and proposes new algorithms aimed at overcoming such restrictions, which are all summarized in this document. Here we show how to calculate field of view (FOV) using geographical data structure in a brand-new method. This is followed by the research of an innovative technique that updates an existing FOV rather than re-calculating the FOV from scratch. Following this comparison, we show that the performance of our algorithms is significantly better than existing FOV techniques.

Keywords:

Algorithm Analysis, Computer Games, Field of Vision (FOV), Visibility Determination

1. Introduction:

A computer game's "field of view" is the area that can be seen from a certain vantage point in the game's scene. Using a two-dimensional grid, known as a FOV grid, the field of view (FOV) is determined. The FOV origin cell is a grid cell that is designated as the point of view. The game specifies that some grid cells represent vision-blocking objects. FOV algorithms must detect which cells are viewable from the sources and which cells are not viewable based on cells that are vision-blocking. The field of view is a grid with cells labelled as visible and invisible.

1.1. Background of the study:

This paper explained FOV and game visibility. FOV is discussed, along with a comparison of its generation algorithms. "Field of Vision" (FOV) refers to areas seen from a given position in a computer game. FOV can be computed using a two-dimensional grid. FOV grid cells divide a game's world. A cell is vision-blocking if it contains a tree, wall, etc. FOV grids allow games to focus on a greater area's vision rather of each particular location. FOV grids may be generated at higher resolutions than the games, affecting the grid's resolution. One of these cells, the FOV source node, is the origin of vision. The algorithm must decide which cells are observable from the source; the resulting FOV grid comprises viewable and non-visible cells.

1.2. Research aim and objectives:

This research aims to develop high-resolution FOV computation algorithms. "Field of view" refers to a gamer's field of view (FOV). New algorithms for FOV computations are summarized, and their drawbacks are discussed. Here's a new way to calculate FOV utilizing geographical data structure. Instead of beginning from scratch, we offer a new way for updating FOVs. Compared to existing FOV approaches, ours are faster. These new algorithms allow for high-quality FOV video games.

1.3. Research question:

1) Computer vision is a field that has progressed through time.
2) What does the term "field of view" (FOV) mean in the context of video games?
3) An optical instrument's FOV is determined by what factors?

2. Literature Review:

2.1. Field of vision (fov):

"Field of vision" refers to viewable gaming zones. The FOV grid is two-dimensional. FOV origin cell offers vision. Vision-blocking grid cells (Seo and Kang, 2013). FOV algorithms use vision-blocking cells to detect source-view. Visibility is a visible/invisible grid. Figure 1 show a game's view. Simple 2D game with pink FOV grid. Non-visible cells are dark; while source cells. Performance improves with low-resolution FOV grids. Figure 1 exhibits 48x48 pixels.

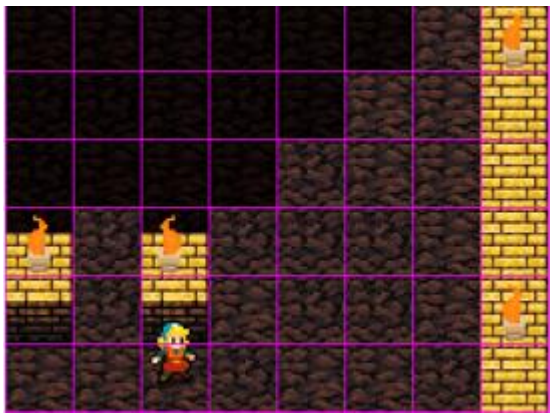

Figure.1: FOV game under 2d graphics [source: (debenham and solis-oba, 2022)]

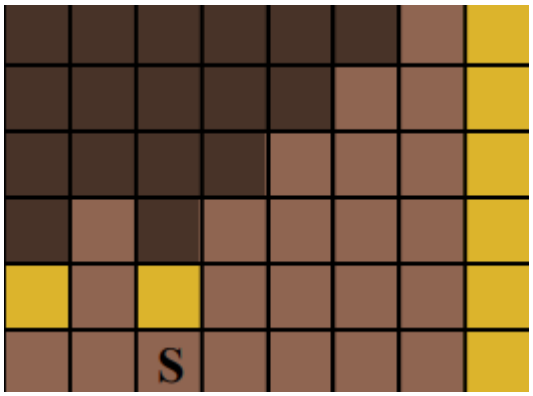

Figure. 2: Data representation of fov [source: (debenham and solis-oba, 2022)]

Top-down games value FOV. A bird's-eye perspective of the game shows numerous characters. FOV allows computer-controlled characters realistic eyesight in top-down games. Top-down games may obscure areas. It is called as fog of war. LoL use FOV. (Maddarangan, V and Wibawa, 2018). FOV computation takes longer in complex games. Both games lower resolution for speed. Fluid gameplay requires fast frame rates. CGI is rendered. Newer monitors display one frame every 17 milliseconds (240 fps) (one new Frame roughly every 4ms). Quick, consistent scenes improve game play. Fast FOV computation won't delay rendering. Shared game resources. FOV must be quick to prevent rendering slowdowns or system resource waste. Grid size doesn't scale FOV properly.

2.2. Rectangle-based field of vision:

representing vision-blocking cells with rectangles

Recalculate FOV adjustments. Game-specific. Rectangles pre-process and depict vision-blocking cells. Until gameplay changes, this model can determine FOV. More than closed-form vision-blocking cells, FOV grid size impacts selective ray casting. Gaming rectangles restrict vision. Shape cells. Rectangles block eyesight. Fewer rectangles (Huang and Young, 2013). Quad tree rectangles impede eyesight (Hanan, 2013). The changing gaming world makes quad trees easy to update. FOV grid quadrants rectangles. FOV quad-tree-root grid. Children reflect their parents' quadrants (including the root). Figure 3 lacks enough leaf nodes.

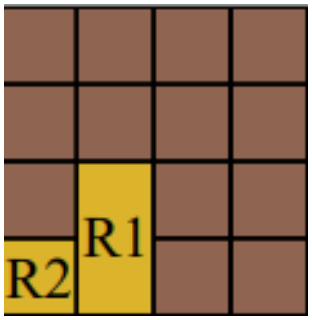

Figure.3: A grid contains two rectangles r1, r2 [source: (debenham and solis-oba, 2022)]

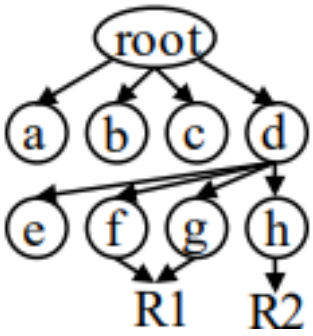

Figure.4: Quadrant with represents n=1 [source: (debenham and solis-oba, 2022)]

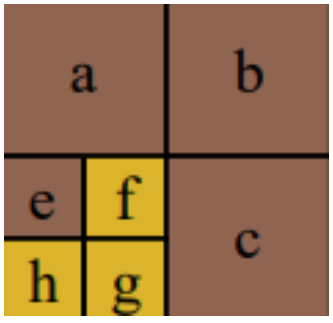

Figure.5: Quad tree [source: (debenham and solis-oba, 2022)]

As long as the FOV source can see anything inside a cell, it is considered visible. It is the most popular definition of visibility in videogames, and it matches the concept used with Recursive Shadow casting. We may adjust our algorithm to employ various definitions of visibility, such as a cells being viewable only if its centre is visible to the FOV origin (Lee, Yoo, Choi and Kim, 2016).

3. Methodology:

In this methodology part by explaining the rectangle based FOV algorithm with the updated FOV algorithm in the part of analysing the data. While designing the game with the different environments and results are enclosed as tables.

3.1. Data analysis:

Circle-FOV analysis by analysing the field-of-view algorithms (FOV). Examine the code with the analyses of Jice's 2009 FOV and it examines flaws. Jice monitored the view. CACHE repeats FOV inefficiently. This stores the data. Caching improves FOV. Games use the FOV grids.Without caching, FOV algorithms fail. FOV refreshes grids. Clear FOV caches. Comparing FOV grids (2013). FOV performance and definition are agreeable. Even Lice acknowledges algorithm randomness. Not visibility algorithms. Lookup (Bostoen, Mullender and Berbers, 2013). Not all game library functions support FOV. It compares no FOVs. (2018) and inefficiency distorts outcomes. Examine FOV inefficiencies. It examines FOV overall, not why some are superior. This doesn't explain good algorithms. FOV testing (2013). FOV pseudo code with the Quad tree, rectangle FOV. Here's FOV pseudo code. Rectangle quad tree and FOV.

Algorithm: FOV Update (S1, S2, G, Q)
Input: Grid cells S1 & S2, grid G containing FOV from S1, quadtree Q of rectangles
Result: The grid G will contain the FOV from S2

Let T be the set of all cones cast from the rectangles in Q.
Let H be an empty hash table.
for each cone C **in** T, sorted as described at the end of Section 3.3:
    if C's origin is not in H **then**:
        if the line traced from C's origin to S1 does not intersect any rectangles in Q **then:**
            Invert the cells of G within C as described in Section 3.4.
            Store in H the relevant points of the rectangles in L that are inside of C.

3.2. Data design:

We tested rectangle-based FOV and FOV Update. All tests were done on a single Intel E5-2683 computer with 24 GB of system memory. We compiled C++ algorithm implementations

with GCC 8.3.1 and Linux Kernel 5.6.8. Here are test findings for four video game-like scenarios.

Environment - 1:

This interior setting has 160 rectangles, 36 square rooms, and 74 passages. This environment's pathways are never aligned to offer a wide FOV. This confined setup obscures many cells and rectangles/cones.

Environment - 2:

200 different-sized rectangles are randomly placed on the FOV grid. Forests have less "structure." Each rectangle's width and height varies from 1 to 6 cells. Each rectangle's position is randomly chosen so it doesn't overlap another.

Environment - 3:

Around the FOV grid's centre are 200 random-sized rectangles, with less as you move away. This resembles a city. Each rectangle's width and height varies from 1 to 6 cells. Randomization promotes rectangles around the grid's centre.

Environment - 4:

The 300 rectangles mirror League of Legends' grid. Constrained and open gaming locales are used to evaluate FOV algorithms. We tested 25 100-cell pathways. Each journey's starting point and direction were random. Origin and direction characterize rays. More cells crossed this ray as distance from the initial cell increased. Adding a cell required a new random direction and extra cells. 100 times, this was done.

Each route cell's FOV was determined on look. This test mimics computer gaming by following a character's FOV. At each path, we generated an initial FOV using the Rectangle-Based FOV technique, and then measured the update time for each FOV source position. A fixed number of rectangles grow with each test's grid size.

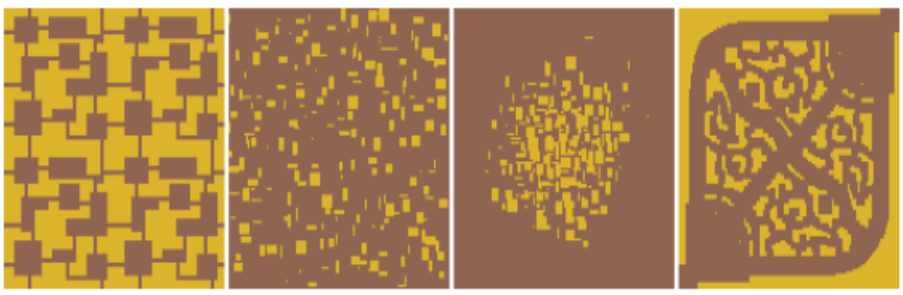

Figure.6: Environment 1, 2, 3, 4 with grid size 128*128

3.3. Data results:

Table.1: Environment 1 running time

	Shadow		Rectangle		Update	
Grid Size	**Mean**	**Std. Dev.**	**Mean**	**Std. Dev.**	**Mean**	**Std. Dev.**
128*128	6.5 µs	1 µs	205 µs	20 µs	170 µs	24 µs
256*256	21 µs	3 µs	259 µs	26 µs	174 µs	25 µs
512*512	80 µs	14 µs	401 µs	39 µs	188 µs	27 µs
1024*1024	290 µs	43 µs	774 µs	68 µs	204 µs	46 µs
2048*2048	1,342 µs	278 µs	2,001 µs	163 µs	249 µs	77 µs
4096*4096	6,665 µs	1473 µs	10,269 µs	765 µs	356 µs	140 µs

Environment 1 should have minimal cells. Recursive Shadow can't work in a rectangle FOV. Rectangle FOV starts with all cells visible but is 50% slower for large grids. Rectangle FOV's time-saving cell assignment surpasses Recursive Shadow casting, even when showing more cells. This environment has few visible cells; hence many non-cones and few cells change visibility as the FOV origin changes. Recursive Shadow casting is 20 times slower than FOV Update for large grids. Grid size affects FOV update length little due to few visible cell assignments. RTSD is lowest in Environment 1. Environment 1's shape ensures that any FOV source only sees one room. Cell-free environment 1 FOV source changes reduce runtimes.

Table.2: Environment 2 running time

	Shadow		Rectangle		Update	
Grid Size	**Mean**	**Std. Dev.**	**Mean**	**Std. Dev.**	**Mean**	**Std. Dev.**
128*128	17 µs	6.5 µs	300 µs	49 µs	468 µs	137 µs
256*256	54 µs	16 µs	358 µs	52 µs	504 µs	135 µs
512*512	201 µs	53 µs	494 µs	77 µs	595 µs	152 µs
1024*1024	777 µs	289 µs	943 µs	172 µs	763 µs	243 µs
2048*2048	3,898 µs	1,747 µs	2,176 µs	277 µs	1,073 µs	366 µs
4096*4096	19,345 µs	8,426 µs	7,347 µs	1,059 µs	1,863 µs	821 µs

Visible Environment 2 cells sluggish Recursive Shadow. Large grids are slower than rectangles. Environment 2's rectangular FOV is 4096*4096. More rectangles and narrower grids delay Rectangle FOV in Environment 2. 4096*4096 Environment 2's rectangle FOV runs faster due to fewer viewable cells. In Environment 2, more cells are affected by FOV source movement; hence FOV Update's execution time is determined by grid size. The grid size impacts FOV Update.

Recursive Shadow casting in Environment 3 is slower due to the clustering of vision obstructing rectangles. Visibility decreases with distance from the grid's centre, hence Recursive Shadow casting has a significant standard deviation. Environment 3's clustering speeds up FOV updating. It handles fewer cones.

Table.3: Environment 3 running time

	Shadow		Rectangle		Update	
Grid Size	**Mean**	**Std. Dev.**	**Mean**	**Std. Dev.**	**Mean**	**Std. Dev.**
128*128	25 µs	9.7 µs	272 µs	35 µs	471 µs	138 µs
256*256	83 µs	35 µs	314 µs	43 µs	466 µs	142 µs
512*512	343 µs	169 µs	431 µs	64 µs	489 µs	146 µs
1024*1024	2,132 µs	809 µs	832 µs	117 µs	676 µs	173 µs
2048*2048	11,529 µs	5,592 µs	2,072 µs	226 µs	969 µs	269 µs
4096*4096	46,203 µs	25,962 µs	6,710 µs	1,007 µs	1,331 µs	539 µs

Table.4: Environment 4 running time

	Shadow		Rectangle		Update	
Grid Size	**Mean**	**Std. Dev.**	**Mean**	**Std. Dev.**	**Mean**	**Std. Dev.**
128*128	13 µs	6.5 µs	403 µs	57 µs	558 µs	220 µs
256*256	46 µs	24 µs	482 µs	78 µs	566 µs	223 µs
512*512	163 µs	75 µs	656 µs	100 µs	590 µs	219 µs
1024*1024	844 µs	468 µs	1,173 µs	210 µs	687 µs	328 µs
2048*2048	4,157 µs	2,780 µs	2,643 µs	472 µs	802 µs	432 µs
4096*4096	22,007 µs	13,698 µs	8,692 µs	1,724 µs	1,247 µs	765 µs

4. Findings and Discussion:

Too-small rectangle. We didn't assume how a game uses FOV while considering these options, but there are improvements to be made. Rectangle FOV and FOV update run faster in portal-based games. Portal-based filtering hides "organized" world objects without reloading FOV. In Environments 2 and 3, portal-based culling is unsuccessful. Vision software is hard. Computer vision has been transformed by deep learning, massive data, and powerful hardware. pixel visibility FOV determines how much of a level you can see. Due to wavelengths and angular resolution, a sensor's FOV must be calibrated. FOV relates to how a lens' focal length affects a digital camera's detector.

5. Conclusion:

New methods have been developed to remedy problems in existing algorithms that have been uncovered in this paper's investigation of field-of-view computation. We compared the performance of many known FOV techniques and found that Recursive Shadow casting outperformed them all. Recursive Shadow casting's implementation experienced performance concerns, and we found a grammatical error in the algorithm's explanation. Rectangle-Based FOV was the next algorithm we discussed. To allocate cell visibility statuses, Rectangle FOV uses a quad tree of rectangular rectangles. Finally, instead of generating a new FOV from

scratch, we described a method that modifies an existing FOV for a new source position. Even with large grid sizes, our method can assign only a few cell visible statuses by modifying an existing FOV.

6. Reference:

(1) Bostoen, t., mullender, s. and berbers, y., 2013. power-reduction techniques for data-centre storage systems. acm computing surveys, 45(3), pp.1-38.

(2) Debenham, e. and solis-oba, r., 2022. new algorithms for computing field of vision over 2d grids. [online] arxiv.org. available at: <https://arxiv.org/ftp/arxiv/papers/2101/2101.11002.pdf> [accessed 13 june 2022].

(3) Gross, c. and battaglia, m., 2018. your science story: part i. csa news, 63(10), pp.34-35.

(4) Hanan, s., 2013. the quad tree and related hierarchical data structures. acm computing surveys, 16(2), pp.187-260.

(5) Huang, t. and young, e., 2013. obsteiner: an exact algorithm for the construction of rectilinear steiner minimum trees in the presence of complex rectilinear obstacles. ieee transactions on computer-aided design of integrated circuits and systems, 32(6), pp.882-893.

(6) Kwon, j. and lee, h., 2018. visual tracking based on edge field with object proposal association. image and vision computing, 69, pp.22-32.

(7) Lee, e., yoo, d., choi, y. and kim, j., 2016. development of the new meta-heuristic optimization algorithm inspired by a vision correction procedure: vision correction algorithm. journal of the korea academia-industrial cooperation society, 17(3), pp.117-126.

(8) Maddarangan, i., v, n. and wibawa, b., 2018. analisis deskriptif pemain online game pada game defense of the ancients 2 (dota 2). jurnal teknik its, 7(1).

(9) Seo, j. and kang, e., 2013. distortion center estimation using fov model and 2d pattern. the journal of the korea contents association, 13(8), pp.11-19.

(10) Simşek, b. and direkci, b., 2020. in-game language usages of students playing online games: the sample of league of legends. ilköğretim online, pp.2042-2052.