Magic Rays: The Trickery Behind the "3D Graphics" of *Wolfenstein 3D*

Tyler Pringle, MATH 400, 5/12/2024

Abstract

In this paper, I discuss the 2D ray casting employed in id Software's video game *Wolfenstein 3D*. I first go over the mathematical concepts behind the technology that allow 3D perspective to be simulated without actually modeling 3D objects. I then go into depth on how ray casting came to be the driver behind the creation of *Wolfenstein 3D*, giving much focus to the role of programmer John Carmack, who developed the core of the game's ray casting engine. I conclude by reflecting on the impact of John Carmack and id Software's contributions to the video game industry and their use of math to push '90s computer hardware to its limits.

1 Why care about 3D graphics in video games?

Though 3D graphics have a wide range of applications, including films, computer-aided design, and physics simulations, if you ask anybody what they associate 3D graphics the most with, it is likely that they will answer video games. This should be no surprise, as video games have grown alongside the development of 3D graphics and as 3D video games continue to stay at the forefront of popular culture. The first 3D video game consoles, such as Sony's PlayStation, the Nintendo 64, and the Sega Saturn, were released in the 1990s and were some of the first introductions many people had to the technology of 3D graphics. And today, video games remain at the top of people's minds when 3D graphics are mentioned, as games like the third-person shooter *Fortnite* dominate discussion of today's popular culture.

A struggle for the video game designer and developer has and continues to be how they can translate their own unbridled vision of the game in their head into a viable product that consumers can actually play. This often means that, while developers may want the most realistic graphics technologically possible for their game, they ultimately have to consider the limitations of the hardware that the average video game player uses. Better graphics inevitably mean that there are more surfaces to take into account for processes like lighting. The more surfaces there are in a scene,

Fig. 1: A screenshot from *Metal Gear Solid*, released in 1998, showcasing the primitive, blocky nature of early 3D computer graphics.

Fig. 2: A screenshot from *Fortnite* taken in 2024.

the more a computer has to calculate how light bounces off of each and every one of those surfaces. If a computer has to make too many calculations at once, it will start to slow down, meaning the game will slow down and may even stop. A game slowing down, stopping, or crashing is absolutely unacceptable for a consumer who just wants to have fun continually playing a game, so developers have to do whatever they can to stop that from happening. Thus, there is always a compromise to be made between what looks the best and what runs the smoothest on the hardware a game's players will be using. We can also envision this as a battle between developers and hardware: what can developers force hardware to do in order to translate as much as their vision into a game's final product?

The developers at id Software were constantly asking this question and seeking to cover new ground in the video game industry. id Software, founded in 1991, is renowned for releasing some of the most iconic and influential games in history, such as *Wolfenstein 3D*, *Doom*, and *Quake*, all of them spawning their own series of games. What the developers at id realized is that forcing hardware to do what they want oftentimes requires some clever implementation of mathematical concepts. In this paper, we will discuss one particular technique that id Software developers used to mimic 3D graphics and triumph over the limitations of hardware: ray casting, utilized in *Wolfenstein 3D* (among other id Software games). This paper will go over some of the math behind this technique and the history of its implementation within *Wolfenstein*. To this end, this paper hopes to paint a picture of tireless developers applying mathematical concepts to overcome the limits of the technology of their time.

2 The Ray Casting Technique

2.1 Basic Overview

Ray casting, rather than being a singular technique, is a broad category of techniques that themselves all fall under the umbrella of ray tracing. According to Wikipedia, a ray-tracing algorithm basically works by "tracing a path from an imaginary eye through each pixel in a virtual screen, and calculating the color of the object visible through it" [1]. When talking about video games, though, **ray casting** is a technique used to create a 3D projection based on a 2D map. With this technique, video game developers are able to mimic 3D perspective without actually having to model any 3D objects or scenes. Ray casting requires fewer calculations than the use of actual 3D surfaces, which makes ray casting an ideal technique for creating the illusion of fast 3D graphics on low-end hardware. Or, rather, a developer's ideal solution for introducing fast pseudo-3D graphics to video game players in the 1990s.

Fig. 3: *Wolfenstein 3D* uses ray casting to implement a 3D perspective while using 2D sprites for most other entities, like enemy characters.

Wolfenstein 3D, a first-person shooter developed by id Software and initially released for the DOS family of systems in May 1992, is one of the most popular and influential examples of the ray casting technique. Although *Wolfenstein 3D* was not the very first first-person shooter, the game is considered to be the genre's first "breakaway hit" and provided a model to follow for countless games within the genre [2]. Ray casting allowed the developers of *Wolfenstein* to quickly render pseudo-3D environments for players to move around in. The swift rendering speed was necessary for the fast-paced action that was required for the genre to blossom into an experience that had mass appeal.

2.2 The Math

This next section is adapted from material by Lode Vandevenne [3].

The ray casting technique used in *Wolfenstein 3D* requires a 2D map. Additionally, this map must be based on a 2D square grid structure, so complex geometry beyond 2D squares that aligned with the 2D grid was not possible. The limitations of *Wolfenstein*'s engine also meant that the walls had to be the same height and that height differences in general were impossible (although it is possible to include variable-height walls in a ray casting engine, as 1993's *ShadowCaster* included these and was based on *Wolfenstein*'s engine [4]). So, the engine allowed for rooms based on a 2D grid structure with 2D sprites inhabiting those rooms.

To understand ray casting, we first have to understand the 2D grid structure we are using. In the grid, each square either contains a wall or does not. More specifically, we can encode squares as having a value of 0 for no wall and positive values for the existence of a wall. Different positive values can be employed for different colors or textures.

We can treat our computer screen as being aligned with the 2D Cartesian coordinate system, with the X-axis running along either the top or bottom of the screen and the Y-axis running along the left side of the screen. Thus, we can treat each vertical column of pixels as representing discrete values along a vertical line for a particular X coordinate, like X = 1. With this in mind, we can start casting our rays: for every X coordinate, or vertical column of pixels, we cast out a ray starting at the player that is based on the player's current viewing direction and the particular X-coordinate. Remember that we are dealing with a 2D map, so the player and rays we are casting exist on the 2D grid structure. We then let the ray move forward through the grid until it hits a wall square. Once the ray hits a wall, we calculate the distance of the hit point to the player, using this distance calculation to determine how tall/close

the wall should appear on the screen. If a wall is further away, then its height should appear shorter on the screen.

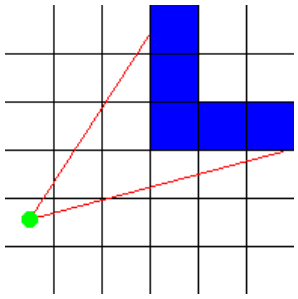

Fig. 4: Top-down view of two rays (red) starting from the player (green) and hitting the blue walls. *(image by Lode Vandevanne)*

The change in height of the walls is what gives the illusion of depth in the perspective. Specifically, *Wolfenstein* utilizes a form of **perspective projection**, where lines that are parallel in the game world appear to converge towards each other at a horizon. Additionally, in *Wolfenstein 3D*, any area underneath the calculated walls is given a solid color for the floor, and the same occurs for any area above the calculated walls.

Although a human can immediately tell where a ray hits a wall, this is not true for computers, as a computer can only check a finite number of positions on the ray. A computer has to let the ray start at the player's position, let the ray move a particular distance, and check if the ray's position is *currently* inside a wall. If the ray is not inside a wall, then the computer has to repeatedly step and check if the ray is inside a wall, stopping once a wall is hit. If we let the ray go a constant distance before checking on its position, there's a chance that the ray could pass through a wall. What we need is an algorithm that guarantees that we will hit a wall.

Therefore, what we should do instead is check every side of a wall that a ray will encounter along the way. We can give each grid square a length/width of 1 so that every line on the grid (e.g. the sides of walls) matches up with an integer value. This makes the step size vary as the ray moves forward. There are different ways of calculating the step size as the ray travels through the grid, but that is beyond the scope of this paper.

To actually represent the direction of the players and the rays, there are a few different methods a developer could use, two of which I will quickly mention. One involves representing the directions as Euclidean angles and treating the field of view (an angle indicating how much of the world a player can see from their first-person view) as

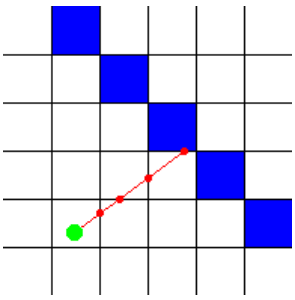

Fig. 5: Here, we check on the ray every time it might hit a wall, e.g. every time it intersects with one of the lines on the grid. *(image by Lode Vandevanne)*

another angle. Another method utilizes vectors, treating the position of the player as a vector and their viewing direction as another angle. With this method, we also implement a line/vector that is (almost always) perpendicular to the viewing direction vector, and we treat it as the camera plane. With this camera plane, we can represent each ray as the sum of the player's position vector, their direction vector, and a multiple of the camera plane vector. This camera plane represents the screen, so we can easily match up a ray with the x-coordinate, or column of pixels, it should correspond to.

3 History of the Technique

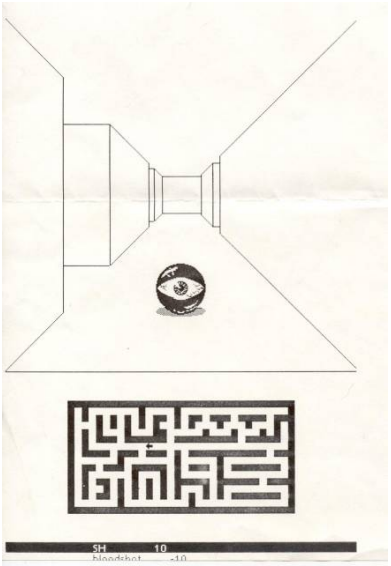

Fig. 6: Screenshot of 1973's *Maze War* on a Xerox Alto workstation. This game is rendered using a technique called wireframe rendering, *not* ray casting.

3.1 Background

It should first be made clear that the developers at id Software were not the first to create a game with a first-person 3D perspective: that recognition goes to Steve Colley, Greg Thompson, and Howard Palmer of NASA who developed *Maze War* in 1973 [5]. The graphics in this game, while quite primitive even by the time of *Wolfenstein*'s release, bear quite a bit of resemblance to the very rectangular and boxy nature of *Wolfenstein*'s rooms. While *Maze War* did not utilize ray casting, opting instead for wireframe rendering, the game provided a foundation for 3D graphics in video games that first-person shooter games would build upon in the coming decades.

Fig. 7: Screenshot of *WayOut* running on an Atari 800, showcasing the simplistic solid-color walls used in this game's ray casting engine.

Wolfenstein 3D was not even the first to utilize what we can call 2D ray casting to implement a 3D perspective. As early as 1982, Paul Allen Edelstein's *WayOut* used ray casting to represent its world [6]. Of course, the game was still a far cry from *Wolfenstein*, with only blue walls and one other entity (the 'Cleptangle' antagonist) to render. Nonetheless, *WayOut* proved ray casting's ability to render a first-person 3D perspective with smooth 360-degree rotation on consumer hardware, which was quite a marvel in 1982. For the more sophisticated ray casting of *Wolfenstein 3D*, though, more improvements to the ray casting algorithm and somewhat more advanced tech would be needed.

However, by 1991, there were not many improvements made to the 2D ray casting technique. 1987 brought *MIDI Maze*, another ray-cast game with smooth rotation and networking capabilities that amazed people of the day (up to 16 people could play at once to shoot at each other) [5]. Graphically, though, the game was only somewhat better than *WayOut*: instead of the walls being a single solid color, the walls were different shades of gray. It was not until John Carmack of id Software began experimenting in 1991 that we saw major improvements in the ray casting technique.

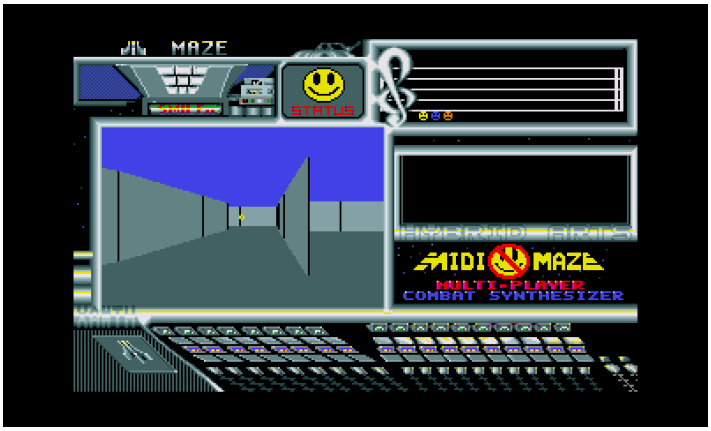

Fig. 8: Screenshot of 1987's *MIDI Maze*.

3.2 id Software's Work

id Software had its more informal start in late 1990 with just four employees at the software company Softdisk: John Romero, John Carmack, Tom Hall, and Adrian Carmack [7]. They had their official start on February 1, 1991, after releasing a few episodes of their 2D platformer series *Commander Keen*, and they continued on making games (some of which were still made under contractual obligation to Softdisk). Around this time, programmer John Carmack wanted to experiment with 3D computer graphics and figure out how he could render authentic, smooth 3D scenes on the average home computer. Thus, after id's official founding, Carmack quickly began work on a 3D engine that used ray casting to render the world and 2D sprites to represent enemies and other characters. According to Carmack, this engine took about 6 weeks to develop and became the 3D first-person shooter *Hovertank 3D*, which was released in April 1991 [8].

Carmack had to work with a lack of resources on the implementation of ray casting, which explains why the ray-cast graphics of *Hovertank 3D* were not that much of a graphical jump from something like *MIDI Maze*. Either way, after the release, John

Fig. 9: John Carmack's first foray into ray casting with the game *Hovertank 3D*.

Carmack was actually shown a demo of *Ultima Underworld*, an RPG with a similar 3D perspective that implemented **texture mapping** (mapping a picture onto a surface rather than just a solid color). The engine for *Ultima Underworld* did have to sacrifice some speed for the texture-mapped 3D perspective, though. Upon seeing this demo, Carmack was determined to create an engine that included faster texture mapping [9]. This led to Carmack working to enhance the *Hovertank* to include just such a thing, and he spent another 6 weeks on this endeavor. The final product of this project was *Catacomb 3-D*, another first-person shooter that id released in November 1991. With texture-mapped graphics that ran at a nice frame rate on average home consumer hardware, Carmack and the rest of the team were getting closer to their breakthrough pseudo-3D title.

Fig. 10: John Carmack quickly modified his engine for *Hovertank 3D* to include texture mapping, showcased here in the walls of *Catacomb 3-D*.

After the team at id Software decided to work on a fast-paced 3D action game, they decided to base the game off of 1981's action-adventure game *Castle Wolfenstein*. They acquired the rights to the name and began work on what would become *Wolfenstein 3D* on December 15, 1991 [10]. Carmack spent another four weeks improving the raycasting engine from *Catacomb 3-D*, with improvements ranging from support for doors to faster and higher-resolution graphics. From here, the level designers, John Romero and Tom Hall, were able to design levels using the 2D tile editor that was utilized for their *Commander Keen* platformers, as the ray casting engine would handle the 3D projection of the 2D maps Romero and Hall created [8]. Ultimately, id Software would finish up their work on the game in Spring 1992 and initially release *Wolfenstein* on May 5, 1992.

4 Conclusion

Fig. 11: Despite looking primitive now, *Wolfenstein 3D* changed the way the public saw gaming.

Wolfenstein 3D became a breakout hit: through a shareware distribution system, where id would freely give out an extended demo of *Wolfenstein* and sell the rest of the game separately, by mid-1994, id had sold 150,000 real copies of *Spear of Destiny* expansion to the game and estimated "worldwide shareware distribution to top one million" [11, p. 22]. The immersiveness of the 3D ray-cast environments and the gratuitous violence revolutionized the '90s video game industry and showed that the PC

was capable of more than just strategy and flight simulation games [12, p. 105]. None of this would have been possible without the vision of the four small-time programmers looking to shake things up within the field of video games. Of course, vision is not enough to bring about change. You need to know-how to back up your vision, and id had the know-how in spades.

John Carmack more than exemplified this fact through his work to develop the ray casting engine that would eventually be the core of *Wolfenstein 3D*. He found himself wanting to solve the problem of smooth 3D graphics on hardware that was accessible to the average consumer. So, he experimented and delved into the rather unconventional technique of ray casting. Understanding and implementing a technique such as ray casting in a game and having it run smoothly on hardware we now consider insanely primitive was a feat that only a few programmers could aspire to. But John Carmack and the developers at id did it anyway by going through the trouble to understand the math behind implementing a basic 3D perspective and using their ingenuity to have this math be calculated as quickly as possible. After all, limits are meant to be broken, but only by those who understand why and how those limits are there in the first place.

References

[1] Ray tracing (graphics). Page Version ID: 1223357437 (2024). https://en.wikipedia.org/w/index.php?title=Ray_tracing_(graphics)&oldid=1223357437 Accessed 2024-05-13

[2] Shachtman, N.: May 5, 1992: 'Wolfenstein 3-D' Shoots the First-Person Shooter Into Stardom (2008). https://web.archive.org/web/20111025220612/http://www.wired.com/science/discoveries/news/2008/05/dayintech_0505 Accessed 2024-05-13

[3] Vandevenne, L.: Raycasting (2020). https://lodev.org/cgtutor/raycasting.html Accessed 2024-05-13

[4] Peel, J.: The life and times of John Romero, gaming's original rock-star – part 3: Nightmares (2016). https://www.pcgamesn.com/quake/john-romero-interview-part-3 Accessed 2024-05-13

[5] Thor Jensen, K.: Knee Deep in the Dead: The History of First-Person Shooters (2022). https://www.pcmag.com/news/the-complete-history-of-first-person-shooters Accessed 2024-05-14

[6] Evans-Thirlwell, E.: The history of the first-person shooter. PC Gamer (2017). Accessed 2024-05-14

[7] Fahs, T.: The Early Years of id Software (2008). https://www.ign.com/articles/2008/09/23/the-early-years-of-id-software Accessed 2024-05-15

[8] Wolfenstein 3D 20th Anniversary Director's Commentary with John Carmack. Section: DOOM (2017) (2012). https://nl.ign.com/doom-4/50308/video/wolfenstein-3d-20th-anniversary-directors-commentary-with-john-carmack Accessed 2024-05-15

[9] Mallinson, P.: Games that changed the world: Ultima Underworld (2002). https://web.archive.org/web/20071212192612/http://www.computerandvideogames.com/article.php?id=28003 Accessed 2024-05-15

[10] Taylor, D.: An Interview with ID Software. Game Bytes (4) (1992). Accessed 2024-05-15

[11] Lombardi, C.: To Hell and Back Again. Computer Gaming World (120), 20–24 (1994)

[12] NEXT Generation 51, (1999). http://archive.org/details/NEXT_Generation_51 Accessed 2024-05-15