To focus on individual software engineers in the same light as other professions can be fraught, as the disciple is by nature group oriented. Whether this is due to the complexity of the projects undertaken, the theory required, or some other reason, to assign sole credit to any individual in most cases neglects the work of others involved. Nevertheless, in this essay I’ll attempt to discuss one individual’s contribution to the industry - seeing that his career has had a disproportionately significant impact - but acknowledge the relevant work of others involved.

This individual is John Carmack, a software engineer famous for his work in the field of 3D graphics. In many ways he pioneered the area which has now become the bedrock of the multi-billion dollar games industry. While much of his work was in the early days of 3D games, many of the techniques he created or popularized are commonly utilised to this day; in the games industry and beyond. This essay will explore some of his most influential and interesting ideas, and their impact on the industry.

Firstly, this essay will look at his impact on the process of rendering (translating the existing 3D models into a 2D matrix to be displayed on the screen). The earliest 3D games rendered an area by directly translating a 3D environment into a 2D matrix to be displayed on the screen. This method on main issue: it made it difficult to calculate which surfaces were facing the camera and which were hidden from view; information essential for lighting and optimisation purposes as we will see later. So Carmack found a different approach first described by Albrecht Durer: ray-casting. The idea was to ‘cast’ virtual rays of light from the camera onto the 3D environment to capture it. Now, only surfaces that are visible are rendered, improving render speed in this respect. However this didn’t make up for the cost of calculating rays for any decent screen resolution. As a result, Carmack had to limit the casting to a ray from each column of screen pixels, which mapped to columns of wall texture, in order to reduce the number of potential ray collision points (Permadi, 1996). No better method for reducing cost was found, and so ray-casting was limited to preprocess rendering; namely CGI movies (Christensen, Fong, Laur and Batali, 2006). Only in recent years, with improvements in hardware, has the method become viable without limitations for real-time rendering (Burnes, 2019).

So Carmack looked elsewhere for ways to improve the speed of the rendering process. When rendering a scene, the number of objects rendered is limited to those visible to the camera, so time isn’t wasted rendering those which are hidden behind other objects. The brute force way to achieve this is to enumerate through each object from back to front, and check if it’s in front of any other object with respect to the camera. This was an area that could be improved, so Carmack searched for a new approach. He came across an idea from Seth Teller (1992): create a binary tree structure of objects during preprocessing, from which the relative depths of objects could be queried at runtime more efficiently. Objects in this tree have a front and back surface (which can be chosen arbitrarily). Each object can then be inserted into the tree on one side of an object if they’re in front of it, and the other if they’re behind it. Now at runtime, the tree can be traversed starting at the root, recursively traversing the branch of objects behind it, rendering the root node, then recursively traversing the branch of objects in front of it. This will result in the objects being rendered from back to front, now in linear time (Abrash, 1997). This method was used for Carmack’s company’s own game engine id Tech, as well as the source engine, which were used for some of the most popular and influential games of the 90’s and 2000’s. But this approach was eventually superseded by more effective solutions, namely static meshes.

Next, Carmack had a strong impact on the process of creating shadows for 3D games.

When processing a 3D scene, we want to figure out which areas are and aren’t in shadow. One way of doing this is to create ‘shadow volumes’: 3D objects whose contents are all in shadow. To figure out which areas should contain shadow volumes, virtual rays of light are projected from each light source through each point along the edge of each object. The area created by these rays will be in shadow. This is too costly for real-time applications, so a faster method was found by Tim Heidmann. He discovered that for the surfaces between an object and a light source, if the number facing away from the light source is greater than the number facing towards it, the visible surface will be in shadow (Heidmann, 1991). This information was added to a stencil buffer, which says whether or not each pixel on the screen is in shadow. This method was fast enough to be performed in real-time, but there was still an issue here: if an object is between a light source and the camera - i.e. the camera is inside a shadow volume - the back face of the shadow volume will be included in the surface count, and the shadows will be inverted. Carmack (2000) solved this problem when he discovered that you could instead count the number of surfaces behind the object. This removed the error case, as shadow volumes between the camera and the object were no longer counted. The credit for this discovery shouldn’t lie solely with Carmack however. While he came up with this solution independently, others researchers had found a similar solution around the same time (Bilodeau and Songy, 1999). Regardless, this process became commonly used for games throughout the 2000’s, and served as a starting point for modern shadow rendering (Evanson, 2020).

Along with rendering shadows, Carmack was influential in the area of lighting in 3D games. The standard method for lighting an object at the time was the Gouraud method, where light values were calculated for each vertex of a polygon and then linearly interpolated to the lighting for the edges and spans (Abrash, 1997). This method had a number of issues Carmack wanted to overcome. Firstly, the effectiveness of this method was highly dependent on the vertex count. With a low number, interpolation errors will become more frequent and apparent, and increasing the number of vertices to account for this would take up too much memory. Secondly, the interpolation was linear, meaning that the difference in light value between subsequent pixels along the edges and spans were equal. This results in incorrect lighting when the camera isn’t parallel to the face of the polygon. Thirdly, there was no rotational consistency with this method (Abrash, 1997). Consider a four-sided face with light values of one at the right and left vertices, and zero at the top and bottom vertices. When interpolating the light values, this face is split into two triangles. If the face is split horizontally, there will be two light values of one and one of zero, resulting in the interpolated light across the face weighted towards one. Now if this face is rotated ninety degrees, there will be two light values of zero and one of one. The light across the face will now have changed to be weighted towards zero. Because of this, rotating a face will result in shifting lighting. Carmack discovered a method which would solve all of these problems (Abrash, 1997). A grid of light values would be created in preprocessing for each face of the polygon. This removed the need for high vertex counts, and allowed for consistent, perspective-correct lighting. But it was less efficient.

So Carmack looked elsewhere to find areas that could be sped up. He noticed that each pixel on the screen was updated every frame with the respective light and material information (Abrash, 1997). An object which has more recently been viewed by the camera is more likely to be on the screen in the next frame, so this could be improved with a cache. This would contain the lighting and material information (‘surface textures’) as well as their respective ages, and would be faster to access than the lighting and material textures themselves (Abrash, 1997). Each frame, the elements of this cache are processed. If the element is still on the screen, its age is updated to the current frame, otherwise the age is unchanged. Elements aren’t removed from the if they aren’t on the screen, as we know that it’s more likely that they’ll become visible again than some random surface texture. Only if an element has been invisible for some given period of time is discarded. If a new surface is visible, it is added to the cache. If there is space in the cache, the data is added at that location, otherwise the data replaces the oldest element present.

This fixed the speed issue, but now the cache, made up of large surface textures, took up too much space in memory. To solve this, the area of a single surface was increased for objects further away, with the light value for this surface calculated as the average light values of the grid at that surface (Abrash, 1997). With this method, fewer surface textures needed to be stored with a loss only in the accuracy of the shadows, which was less relevant the further away the surface is from the camera. This method of shading was used in Carmack’s own id Tech engine, and has since become the standard in the industry, used by engines such as Unity and Unreal.

From these examples we can understand Carmack’s ingenuity, and his impact on the games industry and beyond. But we can also see the influence of others throughout the course of his researching, discovering, and implementing these processes. Therefore we can see how the history of the field of software engineering can be understood not just as a series of great individuals, but as an ongoing collaborative effort.

References

Abrash, M. (1997) *Graphics Programming Black Book*. Arizona: Coriolis Group.

Bilodeau, B., & Songy, M. (1999) ‘Real Time Shadows’, *Creativity*. Los Angeles, California, May. Los Angeles: Creative Labs Inc.

Burnes, A. (2019) *Ray Tracing: Your Questions Answered: Types of Ray Tracing, Performance on GeForce GPUs, and More*. Available at: <https://www.nvidia.com/en-us/geforce/news/geforce-gtx-dxr-ray-tracing-available-now/> (Accessed 13 February 2020).

Carmack, J. (2000) Email to Mark Kilgard, 23rd May.

Christensen, P., Fong, J., Laur, D., & Batali, D. (2006) ‘Ray Tracing for the Movie ‘Cars’’, *2006 IEEE Symposium on Interactive Ray Tracing*, pp. 1-6. Available at <https://graphics.pixar.com/library/RayTracingCars/paper.pdf> (Accessed 10 February 2020).

Evanson. N (2020) *How 3D Game Rendering Works: Lighting and Shadows*. Available at: <https://www.techspot.com/article/1998-how-to-3d-rendering-lighting-shadows/> (Accessed 11 February 2020).

Heidmann, T. (1991) ‘Real Shadows Real Time’, *IRIS Universe*, 18, pp. 28-31.

Permadi, F. (1997) *Ray-Casting Tutorial for Game Development and Other Purposes*. Available at: <https://permadi.com/1996/05/ray-casting-tutorial-table-of-contents/> (Accessed 10 February 2020).

Teller, S. (1992) *Visibility Computations in Densely Occluded Polyhedral Environments*. PhD thesis in Computer Science. University of California at Berkeley.