An Exploration of 3D Computer Graphics and Rendering Techniques

Henry Díaz Bordón[1]

December 8, 2023

Think about the last time you watched a Pixar movie, or a video on YouTube showing off outstanding computer-generated graphics. These are truly impressive forms of media, and certainly ones which at some point in our lives we all wished we could make. When I was a kid, I always dreamed of creating my own 3D graphics, but I had nobody to teach me how, nor anywhere to learn from, so I had to invent my own ideas and techniques by myself, from scratch. In this essay, my intuition on how rendering can be performed will be explained, as well as some of the mathematical background required. Drawing three-dimensional graphics can become a straightforward process by modeling the objects as mathematical structures, projecting such models onto our bidimensional screen, and finally computing details such as positions and rotations.

Firstly, in order to perform operations on three-dimensional objects, a universal language in which to describe them is needed, and such language is mathematics, since it can be interpreted by both humans and computers. Vertices or dots are considered to be the components of greatest importance in three-dimensional bodies, components which present properties parallel to those of certain mathematical structures named vectors. The standard, high-school definition of a vector is that of an entity which can be represented by a line segment with magnitude—i.e., length—and orientation, however, in mathematics (as this discipline accustoms), the notion of vectors is utterly abstract, being merely reduced to objects that satisfy a certain series of properties. Among these properties, the

[1] email: henrydiazbordon@gmail.com

abilities of being scaled (which can be thought as stretch by a certain factor) and added are possibly the most noteworthy, but they must also satisfy commutativity—meaning that the order in which they are added does not alter the final sum—as well as six other fundamental axioms. It can be proven that three-dimensional points as well as the aforementioned, well known example of oriented line segments satisfy this set of properties (Lankham et al. 2), thus both of them are consequently considered vectors by mathematicians, which implies that the whole set of operations and reasoning developed in the field of linear algebra (the branch of mathematics dedicated to the study of this kind of objects) can be applied to the points of a 3D space, allowing us to deal with them in a mathematical manner.
Three-dimensional entities also require lines, to which I shall give the name of edges, to connect the object's dots and thereby define its shape, since otherwise they would not be but a point cloud with no cohesion whatsoever. These edges are plainly defined as mere line segments connecting any two points in the model, and although they have no mathematical properties attached, serve the role of giving structure to the body. However, a remarkable property of these objects is the fact that, regardless of the position of the two dots the line is connected to, it will always join them, a quality which will be useful later on. Most 3D graphic standards nowadays instead group edges into triangles, which receive the name of "faces." The majority of three-dimensional assets are frequently expressed in a file format—such as a Word document or an image file—ending in the extension .OBJ (shorthand for "object"), which omits entirely the presence of edges and instead joins vertices in faces or triangles (Shen). This new structure can be constructed as a set of three distinct points along with three lines, none of which connects the same two dots, but it comes to light that these edges are, in fact, redundant, since given any three points it is possible to infer the triangle connecting them, therefore we shall consider faces as sets of three unique points which will be pictured connected. Grouping points in triangles instead of any other geometrical figures might seem at first glance an arbitrary choice, but they turn out to be particularly effective due to, for example, the fact that edges become trivial once the points are known and, hence, we can perform a significant reduction of memory usage

by excluding them; edges do not overlap each other—this, in mathematical jargon, is expressed as triangles being “planar”—and therefore simplifying illumination, drawing pictures on them, etc.; and are relatively simple figures, which allows them to approximate more complex geometry with high precision. Then, after reducing 3D entities into mere points and lines we can manipulate employing the universally reliable results of mathematics, we should now ponder how to actually display them on screen for the spectator to perceive.

Our screen, however, does not understand three-dimensional geometry, and therefore we must transform our 3D space into a bidimensional plane, which monitors are indeed more comfortable with. The simplest and most straightforward example of this category of transformations is known as orthogonal (or orthographic) and has a very comprehensible intuition behind. In order to represent the three-dimensional space onto our two-dimensional, flat screen, we will perform what is called a “graphical projection,” which, as its own name suggests, will consist of projecting or transforming the 3D environment into a bidimensional surface, being that transformation any mathematical function that satisfies the previous property. Perhaps the most intuitive idea on how to perform a realistic projection would be that, for each point in our entity, its position has a height, width, and length (x, y, and z coordinates), being only the last one not able to be drawn; then the process of removing this last coordinate and painting the resulting two-dimensional point receives the name of “orthogonal projection.” However, as it is possible to infer from its notorious simplicity, this procedure manifests manifold deficiencies, among which the lack of perspective—meaning that if situated in front of a object, for example, moving to the left or right will not let you see the back of it, but rather you will always see only its front—and scaling—which implies that moving back will not make you to see the object smaller, either—could be said to be the most notable ones, therefore leading to the necessity to seek a new algorithm to perform a realistic projection (“3D Rotations”). In the search for an algorithm to accomplish this task, we can contemplate how nature and physics solves this problem, more specifically, how our eyes are capable of viewing in certain situations. This second approach to

projection is founded on a branch of physics named optics, and more concretely, on the idea of how we perceive the world when looking through a window (which in this analogy would be the screen), and employing straightforward geometry and trigonometry we reach the following formulas for the x and y position of the projected point in such surface (here denoted x' and y' for the sake of simplicity, whereas x, y, and z refer to the true coordinates of the point): $x' = x\frac{f}{z};\ y' = y\frac{f}{z}$ (Ramamoorthi). In the previous equations, however, the constant f refers to a measure named "focal length," a concept which, in the analogy above, can be thought as the distance between the spectator and the window through which he or she is looking, and which gives the name to this projection procedure: "Focal-length-based projection." This is, however, a simplified version of the focal-length-based projection formula, since the correct equation will also transform z into $z\frac{f}{z} = f$, but since projections shall discard the z coordinate, we can apply the earlier defined orthogonal projection thereafter to accomplish such purpose, producing the truncated expressions listed in the previous sentences. Although the former procedures fulfill their purpose (from my perspective, in a relatively accurate manner) a detail such as the viewer's position is yet to be considered, and a certainly important one which definitely cannot be ignored. This quantity truly matters since, depending on the spectator's situation, a section of space or another will be visible on screen—that is, I will not see the same if I am located in front of a object, than if I move a few steps and face another one—and so we warrant an algorithm through which we can determine the visible slice of space given the viewer's position. In order to comprehend this procedure, firstly consider that if the viewer's eyes must be fixed in the origin (that is, in the coordinates $x = 0,\ y = 0,\ z = 0$) then every entity's points will be described in terms of such location, and so for this to happen given any circumstances, one should displace the whole space by the opposite of the observer's coordinates (e.g., given that the watcher is at the position $x = 1,\ y = 2,\ z = 3$, we shall shift every single point by one unit to the left, two units down and three units back), being that algebraically equivalent to subtracting the viewer's position vector from each one of the vertices in space. However, a problem arises when vertices reside behind the observer's

point of view, since commanding a computer to project a point with negative coordinates could result in inopportune consequences, and so this condition should be taken into account when implementing the proceedings described above. Thus, we have now a procedure for displaying the elements of our environment on screen, however, we have only contemplated the edge cases when the watcher's perspective is static and points in a constant direction.

Most common scenarios, however, involve the observer rotating their field of vision, so this is therefore a case we shall address properly.The said instances result substantially frequent and, in order to describe them properly, plain three-dimensional rotations have to be understood first. As it has already been discussed above, the 3D space consists of three axes, namely x, y and z, which implies that any given trajectory in the three dimensions can be described as a combination of movements along each of these axes, and that same property holds for rotations, since any possible rotation of a three-dimensional object can be expressed as the merge of revolutions around every axis. Rotations are—according to our previously proposed definition—space transformations, and moreover, they belong to a certain category named "linear transformations", which albeit have a somewhat complex, rigorous mathematical definition, can be visualized as transformations that keep distances even, meaning that after the alteration is performed, the distance between one and two will remain the same as such of three and four, and so forth. These special types of functions can be described with structures named matrices, which take the shape of rectangular grids of numbers whose entries specify the orientation and stretching of the axes, and can be composed through their multiplication, which is represented in mathematical notation by their juxtaposition, hence giving us a method to describe complex rotations as products of simple terms. We then say that matrices form their own algebra, and as such, properties we take for granted in the case of numbers may or may not hold for these newly constructed objects. It turns out that the operation of matrix multiplication does not satisfy commutativity ("Matrix Multiplication is not Commutative"), meaning that the order in which our rotation transformations are applied will matter, and that we have to be careful and figure out their

correct order or, otherwise, they will not generate the desired, expected result. The vast majority of resources I have consulted suggest rotating first along the x-axis and thereafter, over y and z (Plein; "3D Rotations"), whilst other recommended computing the y, x and z revolutions in that specific order ("How to Make a Three-Dimensional Project"); and although both candidates appear plausible at first sight, they both fail to produce the expected outcomes, and the intuition for their misbehavior is that, once the x and y spins are performed, z will not produce a "vertical rotation," but rather a revolution around the line perpendicular to the front of the viewer, which again, is not what was meant to occur. The solution I propose for this issue is to plainly get rid of the z rotation at first, that is, to perform it firstly, and subsequently compute y and x, a method which indeed guarantees the expected outcome. Most three-dimensional digital environments, however, provide four keys to interact with rotation: two for increasing and decreasing the horizontal spin and two others for the vertical one, which essentially results more intuitive than managing three different axes of rotation at the same time. Since there is only one single axis pointing upwards, being that the y-axis, that will be the sole one able to produce horizontal rotation when spun, whereas the two, x and z axes, since they are perpendicular to each other and have no height whatsoever, will occasion a vertical movement instead. As it has already been discussed in the previous sentence, the x and z rotations will interfere with each other, and consequently, the amount of spin along the x and z axes shall be determined by the y-axis rotation; knowing that when our y rotation angle is 0º our z rotation has to be zero whereas that of x must be one, and vice versa for a horizontal rotation angle of 90º, we can therefore infer that the formulas for the x and z rotations have to be $\theta_x = \lambda \cos \theta_y$ and $\theta_z = \lambda \sin \theta_y$, where the value λ represents our vertical rotation angle, that is, for given vertical and horizontal rotation angles, we can compute employing the formulae above their respective, true rotation angles with respect to each of the 3D axes. Ultimately, in order to avoid possible graphical issues related to the unexpected behavior of larger angles, we shall keep the horizontal and vertical rotation angles in the closed interval $[-\pi, \pi]$, i.e., being greater than or equal to negative pi and less than or equal to pi. Thus, we have addressed the

last, final basic characteristic of three-dimensional visualization, being now able to observe any 3D scenario with any given properties regarding its objects and watcher.

This is, as far as I am concerned, the most straightforward and intuitive way to produce three-dimensional scenes, and by far the one I employ the most often. This method has made it possible for me to accomplish my dream of creating my own 3D computer graphics. With it, it is achievable to create three-dimensional illustrations while only knowing high-school mathematics. And at the same time produces high-quality, photorealistic results with a relatively small effort. So the next time you, for example, play your favorite 3D video game on your computer, and want to create your own, you will know how to proceed.

Works Cited

"How to Make a Three-Dimensional Project." *Scratch Wiki*, 30 October 2023, https://en.scratch-wiki.info/wiki/How_to_Make_a_Three-Dimensional_Project#Rotation. Accessed 7 December 2023.

Lankham, Isaiah, et al. "Vector Spaces." *University of California, Davis*, 2007, https://www.math.ucdavis.edu/~anne/WQ2007/mat67-Le-Vector_Spaces.pdf. Accessed 4 December 2023.

"Matrix Multiplication is not Commutative." *ProofWiki*, 4 November 2021, https://proofwiki.org/wiki/Matrix_Multiplication_is_not_Commutative. Accessed 7 December 2023.

Plein, Dominic. "Extrinsic & intrinsic rotation: Do I multiply from right or left?" *DEV Community*, 2 May 2022, https://dominicplein.medium.com/extrinsic-intrinsic-rotation-do-i-multiply-from-right-or-left-357c38c1abfd. Accessed 7 December 2023.

Ramamoorthi, Ravi. "Projection and viewing." University of California San Diego, 22 January 2018, https://cseweb.ucsd.edu/classes/wi18/cse167-a/lec4.pdf. Accessed 7 December 2023.

Shen, Han-Wei. *Guidance to write a parser for .Obj and mtl file*, The Ohio State University, 23 February 2011, http://web.cse.ohio-state.edu/~shen.94/581/Site/Lab3_files/Labhelp_Obj_parser.htm. Accessed 7 December 2023.

"3D Rotations." The Intelligent Motion Lab, University of Illinois, http://motion.pratt.duke.edu/RoboticSystems/3DRotations.html. Accessed 7 December 2023.