IS71021E Assignment 2

An Interactive Exploration of Camera Projection

Quince Gore-Rodney[1], Damian Lau[1], and Oliver J. Martin[2]

Department of Computing
Goldsmiths, University of London

February 9, 2026

[1]MSc Virtual and Augmented Reality
[2]MSc Computer Games

Abstract

This is a technical report detailing and analysing the development of a computer graphics program created in Unity which explains camera projection. The techniques used to create the application, and the underlying mathematical concepts will be cross referenced from academic sources and informational books. Additionally, the effectiveness of the interactive applications ability to intuitively educate users on the topic will be examined and constructively critiqued. Consequently, further improvements which to improve the experience are explored.

1 Objective

The aim of this project was to explain the principle of perspective through showcasing different types of cameras used in computer graphics. The variables of the camera's projection type and transformations can be adjusted, such that the relationship between the camera and the finished rasterized image result can be understood. The purpose of this is to visually demonstrate the relationship between the two visualisations, as to nurture long-term memorisation of the conceptual model so that it can be recalled in the future (Benyon, p.469-71, 2014). This report will elaborate on the methodology for how it was be executed.

2 Introduction

2.1 History

The earliest perspective projection cameras in history are known as pinhole cameras, which take advantage of the *camera obscura* (or pinhole image) phenomenon - an innovation which the rest of perspective geometry builds on. Despite the phenomenon being recorded across many civilizations dating back to antiquity, the 11th century Arabic mathematician Ibn al-Haytham is credited with being the first to provide a clear scientific and geometric description of the effect and of the practical applications of the pinhole camera – leading to him being widely considered its 'inventor' (Camera Obscura & World Illusions, 2026).Importantly, al-Haytham showed that the inverted the image of the viewing screen proved that light travelled in straights lines, as well as noting the relationship between the focal point and the size of the pinhole (Camera Obscura & World of Illusions Edinburgh, 2026).

During the European renaissance, pinhole images were used to sketch views of natural landscapes and cities. It is suggestible that the accuracy of the pinhole image allowed for accurate drawing of nature, not directly, but by observations. Artists like Samuel van Hoogstraten to Leonardo DaVinci used the pinhole camera to understand perspective projection (Delsaute, p.113, 1998).

By approximately the end of the 16th Century, a portable camera obscura was built equipped with converging lenses. It could correct the inverted image to produce a direct representation of the perspective from the view of the user. Mathematician Johannes

Figure 1: Visualisation of Ibn al-Haytham's camera obscura, comprising of a small hole in a dark room with a screen. (Camera Obscura & World of Illusions Edinburgh, 2026)

Kepler(1571-1630) insisted he only used it "as a mathematician, not as a painter." Similarly, through this style of visual communication, this project plans to teach users how to understand the principles of perspective with computer graphics.

2.2 Computer Graphics

The foundation of 3D computer graphics began with wireframe CAD simulations, Boeing flight simulations, and the construction of the perspective -generating algorithm by Larry G. Roberts (Manovich, L, 1993). Roberts introduced a 4x4 matrix, known as the perspective projection matrix, which serves to map points in 3D eye space to the 2D surface of the computer monitor. To do this, the viewing volume of the camera (in eye space) is transformed into the canonical view volume - a normalised 3D cube ranging from -1 to 1. For a given point in eye space $(x_{eye}, y_{eye}, z_{eye})$ and projection matrix P, this can be calculated as:

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ w_c \end{bmatrix} = P \begin{bmatrix} x_{eye} \\ y_{eye} \\ z_{eye} \\ w_{eye} = 1 \end{bmatrix} \tag{1}$$

Where (x_c, y_c, z_c, w_c) is the representation of the point in the homogenous 'clip' coordinate system. It is worth noting that the eye space point is implicitly converted into a homogenous point by assigning the w component to 1. In clip coordinates, it is efficient to determine whether a point is within the viewing volume - as a point is visible if $-w_c \leq x_c, y_c, z_c \leq w_c$. This point can then be transformed into normalised canonical view volume coordinates through a process known as perspective division:

$$\begin{bmatrix} x_n \\ y_n \\ z_n \end{bmatrix} = \begin{bmatrix} \frac{x_c}{w_c} \\ \frac{y_c}{w_c} \\ \frac{z_c}{w_c} \end{bmatrix} \tag{2}$$

Figure 2: Samuel van Hoogstraten (Dordrecht, 1627-1678), Perspective of an Open Gallery ('The Tuscan Gallery').)

Where (x_n, y_n, z_n) is the point in the normalised canonical view volume. x_n and y_n are the normalised screen coordinates, and can be converted to pixel coordinates by remapping them from the range (-1, 1) to the pixel resolution of the target screen (with width W and height H):

$$\begin{aligned} x_{screen} &= (x_n + 1) \cdot \frac{W}{2} \\ y_{screen} &= (y_n + 1) \cdot \frac{H}{2} \end{aligned} \tag{3}$$

2.2.1 Perspective Projection

Perspective projection aims to mimic human vision by making distant objects appear smaller and converge towards a vanishing point - much like we see in photography, where parallel lines converge towards the origin and disappear (Fig. 3). To achieve this, the camera view volume is defined as a truncated pyramid frustrum as shown in figure 4. The near and far clipping planes of the frustrum are defined as $-n$ and $-f$, respectively, and the projection plane (the near plane) of the frustrum is rectangular, with a width ranging from l to r in the x direction and from b to t in the y direction. The projection

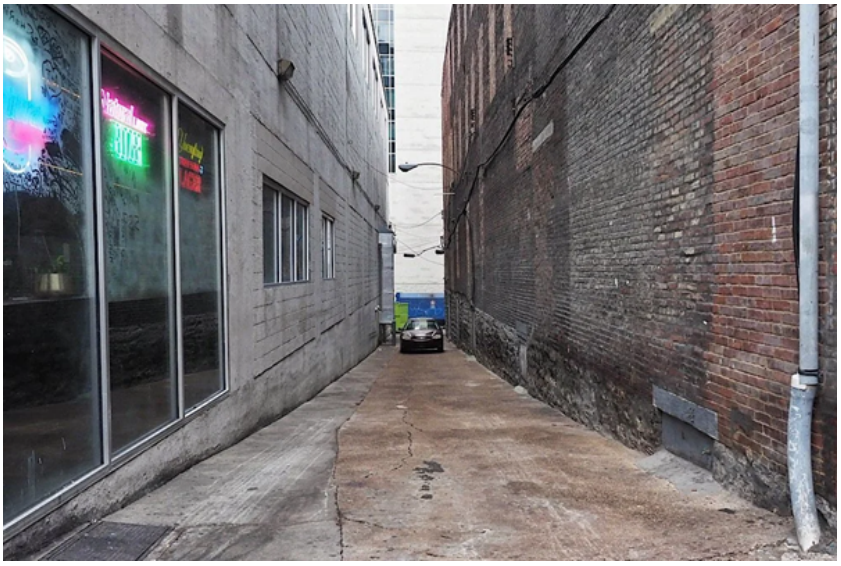

Figure 3: Single Perspective Photography by Diane Wehr.

matrix for perspective projection can then be derived as:

$$P_p = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{4}$$

The use of homogenous coordinates is crucial for obtaining the perspective effect as after a point has been transformed into clip coordinates, the w component is equivalent to the z coordinate in eye coordinates. Therefore, when this point is transformed into canonical view volume coordinates (equation 2), the x and y values are divided by their distance from the camera in world space - leading to points further away converging towards (0, 0), the centre of the camera's view (Susta, 2024).

2.2.2 Orthographic Projection

Orthographic projection is a form of projection which preserves parallel lines and relative distances by keeping all the projection lines orthogonal to the projection plane. This makes orthographic projection particularly useful for technical applications, such as CAD software.
In contrast to the frustrum used as the view volume for perspective projection, orthographic projection instead remaps a cuboidal bounding box to the canonical view volume

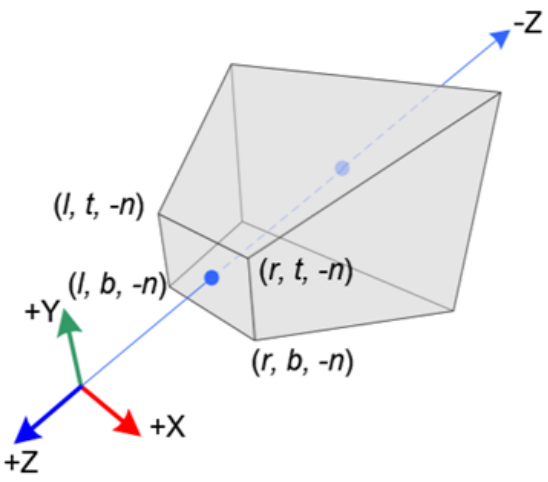

Figure 4: The camera frustrum used for perspective projection in the OpenGL rendering pipeline.

(Figure 5)- which allows for a simpler linear transformation of contained points.
The projection matrix for orthographic projection is then defined as:

$$P_o = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & \frac{-(f+n)}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5}$$

The fourth row of the projection matrix remaining as $(0, 0, 0, 1)$ ensures that the w component has no effect on the projected positions, as it will remain as 1 after transformation.

3 Methodology

Consisting of a 3D Unity project scene with a multi-camera setup, where the primary camera is displaying the rendering of a simple mesh `GameObject`, the second camera observes the primary cameras inner workings. A sample of the camera's rays are visualised using the `LineRenderer` and custom geometry to display where on the objects surface the ray hits, is where the pixel will be displayed. The pixel-to-ray ratio is not an exact translation of the camera's resolution but made for demonstration purposes.

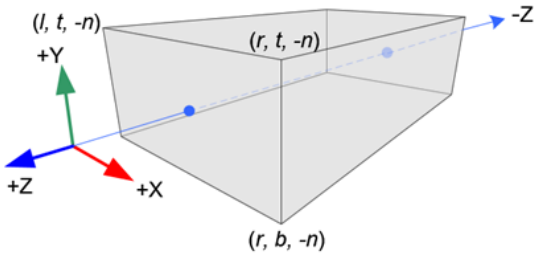

Figure 5: The cuboidal view volume used for orthographic projection in the OpenGL rendering pipeline.

To show how the different projections differ, the primary camera can smoothly interpolate from a perspective to an orthographic view. This was achieved by overriding the camera's default settings by assigning a custom matrix to the `projectionMatrix` property in Unity's `Camera` class. The new matrix is constructed by linearly interpolating each value within the matrix between the perspective projection matrix (equation 4) and the orthographic projection matrix (equation 5). A new variable, O, is introduced indicating the 'orthography' of the camera (i.e. the point at which the interpolation is performed) with a range of 0 to 1 - with $O = 0$ producing the perspective projection matrix and $O = 1$ producing the orthographic matrix. The projection matrix for a given value of O is therefore:

$$P(O) = \begin{bmatrix} \frac{2((1-O)n+O)}{r-l} & 0 & (1-O)\frac{r+l}{r-l} & -O\frac{r+l}{r-l} \\ 0 & \frac{2((1-O)n+O)}{t-b} & (1-O)\frac{t+b}{t-b} & -O\frac{t+b}{t-b} \\ 0 & 0 & \frac{(O-1)(f+n)-2O}{f-n} & \frac{2(O-1)fn-O(f+n)}{f-n} \\ 0 & 0 & -O & O \end{bmatrix}$$

As the orthographic projection requires a larger screen plane than the perspective projection to have similarly sized view volumes, the size of the near plane is also increased with respect to O - specifically O^5 to create an 'ease-in' style effect. The world-space coordinates of the vertices of the far plane are calculated by multiplying the inverse of $P(O)$ with the coordinates of the far corners of the canonical view volume ($(-1,1,1),(1,1,1),(1,-1,1)$ and $(-1,-1,1)$) which are then used in conjunction with a `MeshRenderer` to visualise the view volume.

The visualisation of the camera's rays were implemented using an array of custom ray-casts. These are emitted from the far plane of the view volume, in the $+z$ direction of

the camera in eye-space (i.e. back towards the camera) - ensuring that all objects will be detected, regardless of the current projection. If a raycast intersects with a target mesh, a second raycast is emitted from the camera to the point of intersection to obtain the visible point (as otherwise all the detected points would be on the rear of the target mesh). This point is then converted into screen coordinates by mutliplying it by $P(O)$, and a `lineRenderer` constructs a line between the point of collision and the projected point on the screen. This visually demonstrates the key difference between perspective and orthographic projection, as it shows the projection rays converging towards the centre of the camera at $O = 0$ and becoming parallel as O approaches 1.
Within the application, the user can interact with the UI to change key parameters of the camera. This is intended to give the user an understanding of the relationship between the two models of camera projection. The aim of this demonstration is to explain how the two models of camera projection are 2D projections of 3D space, but with different ways of handling depth. The following variables in the applications are customisable:

- Multiple Camera Perspectives.
- The 'orthography' of the camera.
- Projection Raycast orientation.
- Visiblity of Raycast Lines.

This ensures that the users with different viewpoints can understand what the relationship is between the two mapping techniques.

4 Conclusion

The skewing/diverging effect of perspective projection and how the matrix multiplication relates to the conversion of the 3D space onto a 2D plane can be aided by computer simulations. Techniques of comparing it with different types of parallel projections proved to be the most visually clear example to explain how the matrix is calculated. Secondly, the visualised camera rays proved useful to understand how the pixels relate to individual rays, but there is greater potential for them to explain how they factor into the depth calculations.
To further emphasise the importance of the depth factor in creating the perspective projection, it would be informative to provide an indication of how much distance each ray travelled. The most informative way of display this information would be a billboard annotation above each arrow, from the distance of the primary camera to the surface it hits.
Alternatively, a more visually intuitive way could be a colour indication, using a curve of different RGB values per Line Renderer to show the distance it had travelled in comparison. Perhaps, it would be useful to A/B test the two outcomes with student participants to deduce which is more effective than the other (UX Book).
Finally, the current implementation of the application is such that it could easily be

expanded to allow exploration of other aspects of camera projection. The visualisation aspects are generic with respect to the specific projection matrix used, which allows for future expansion into other types of projection - such as oblique projection. The dimensions of the view volume are also fully customisable, and no assumptions are made about the volume being symmetrical - opening up the possibility of exploration into asymmetrical projections such as those used for mixed-reality displays.

5 Contributions

This project was split by task, with Oliver handling projection matrix mathematics and implementation, Damian constructing the UI, and Quince writing the technical report. The Structural Planning and ideation stages involved all group members.