Homogeneous Coordinates for Perspective Projection

Why we need 4D coordinates to represent 3D space on a 2D screen

Nag Varre

1 Introduction

How do we see in 3D? In other words, how do our eyes take the 3D world around us and convert it into a 2D image at the back of our eye? What are we actually seeing?
These are questions I've started to ask myself more and more, having been working on making a FPS (First-Person Shooter) video game with my friends.

Obviously, all the 3D objects in the scene have specific (x, y, z) coordinates, sizes and distances defined in the xyz Cartesian space that we are all familiar with, however, when we hit render and start playing the game, the way our character appears to see things in the game (and the way we perceive objects with our eyes in real life) is different to the way we might think it should. Lines that were defined to be parallel to each other now appear to converge at a point somewhere on the horizon, and small objects that are closer to us appear larger than they really are whilst large objects further away appear smaller.

A very simple but effective example to illustrate this 'perspective effect' is to look at train tracks face-on:

Figure 1: Even though we know these train tracks must always remain parallel to each other, they appear to converge at a point on the horizon, called the **vanishing point**, or the **point at infinity**.

Therfore we can see that perspective plays a key role in how we perceive 3D objects. This is what makes parallel lines appear to converge, and the relative size of objects to vary depending on their distance away from you.

So, we need to apply some sort of **perspective projection** to project these 3D objects onto a 2D viewing plane for us to see. Light entering our eyes naturally does this, as an object closer to us will have more light rays that bounce off it entering our eye. However in a video game, where every vertex of each shape is individually projected onto the screen (a process known as rasterisation - way more efficient than calculating the path of every single light ray!), we need to develop a method to do this.

And so, in this essay, I will take you through how the invention of homogeneous coordinates allows a very beautiful and efficient conversion of objects from the 3D Euclidean geometric space they are defined in, to the 3D Projective geometric space that is more akin to how we perceive them and finally the 2D screen that we experience the video game through. Enjoy!

2 Setup

In computer graphics, the setup is as shown: A camera (usually centred at the origin) looks into the 3D scene through a 2D screen. As a result, what you are able to see is the collection of objects in the volume enclosed by the frustum. Think of it as looking through a window.

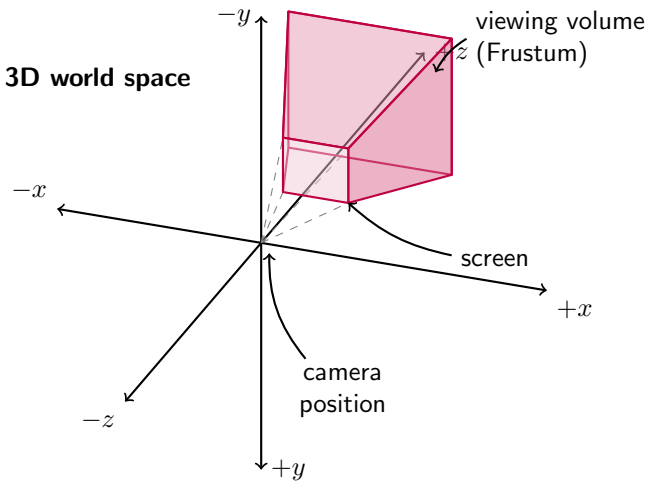


Figure 2: By following the 'light' rays from the camera to our screen, this results in our field of view being depicted by a frustum (that extends indefinitely outwards).

As I mentioned above, the way we render objects within the viewing volume is by projecting the vertices of each object onto the viewing plane/screen that takes into account the relative distances between the vertex and the screen.

For the sake of simplification, let's focus on the perspective projection of a single vertex:

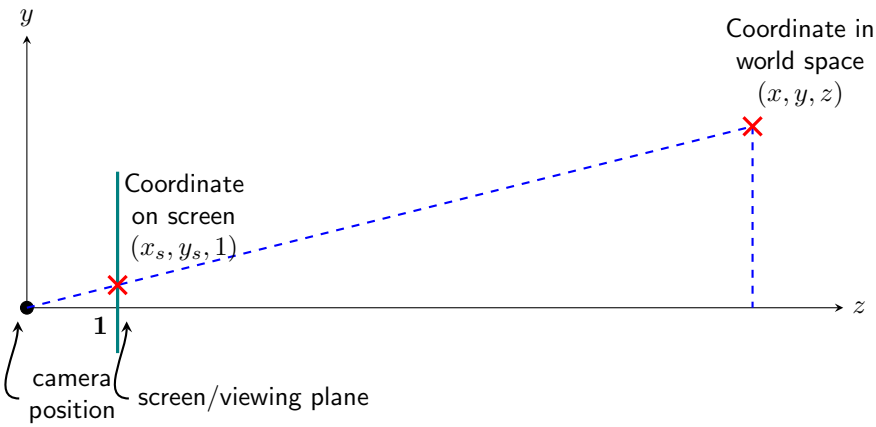


Figure 3: A side view showing how we project an arbitrary vertex (x, y, z) in the world space to a point $(x_s, y_s, 1)$ on the screen at $z = 1$

Hopefully this makes it easier to identify how we might project vertex (x,y,z) onto the screen, whilst preserving its distance to height ratio: we use similar triangles as shown

$$\frac{y_s}{1} = \frac{y}{z} \implies y_s = \frac{y}{z}$$

Similarly, we can derive for x_s,

$$\frac{x_s}{1} = \frac{x}{z} \implies x_s = \frac{x}{z}$$

And so, this begins to give us a **projection matrix** to take a point (x, y, z) in the 3D world space, to a point $(x_s, y_s, 1)$ on the 2D viewing plane at $z = 1$.

Why matrices? Computer graphics involves large amounts of transformations (enlargements, rotations, translations, etc.) - just think about how often and quickly things in the environment move as you play a game. By

encoding each transformation as a matrix, we can multiply all of these matrices together, with the projection matrix applied at the very end to transform it to the viewing plane, giving one total transformation matrix, that is then directly applied to each vertex on an object. With hundreds of objects each with hundreds of vertices, applying a single matrix to each vertex is much more computationally efficient than performing each transformation on each vertex individually!

$$\underset{\text{Projection matrix}}{\begin{pmatrix} ? & ? & ? \\ ? & ? & ? \\ ? & ? & ? \end{pmatrix}} \times \underset{\text{Point in world space}}{\begin{pmatrix} x \\ y \\ z \end{pmatrix}} = \underset{\text{Point on screen}}{\begin{pmatrix} x_s \\ y_s \\ z_s \end{pmatrix}}$$

$$\implies \begin{pmatrix} ? & ? & ? \\ ? & ? & ? \\ ? & ? & ? \end{pmatrix} \times \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x/z \\ y/z \\ 1 \end{pmatrix}$$

Recall that matrix-vector multiplication involves multiplying each row of the matrix with the vector's components (Note that throughout this essay, the $\times$ symbol refers solely to matrix multiplication). Therefore, a matrix that can take z and move it to the x and y components is impossible!

How do we solve this? Here's where projective geometry and homogeneous coordinates come in...

3 Projective Geometry and Homogeneous Coordinates

This concept of perspective is not a new one. Artists have been using it for centuries in order to make their scenes very realistic:

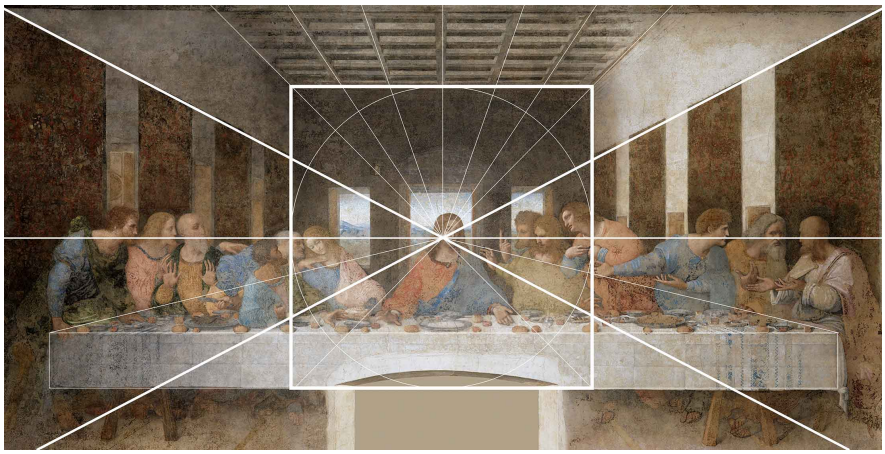


Figure 4: One of my favourite pieces of art that effectively uses perspective is the ingenius Leonardo Da Vinci's *The Last Supper*. As you can see, all the parallel lines defining the room converge at a single vanishing point at the centre of Jesus's right eye!

Mathematically, this manifests itself in **projective geometry**, a more general form of Euclidean geometry that only takes into account the relationships between points and lines (unlike Euclidean geometry which also takes into account angles and distances). It is defined as *the study of geometric properties that are invariant under projective transformations.*

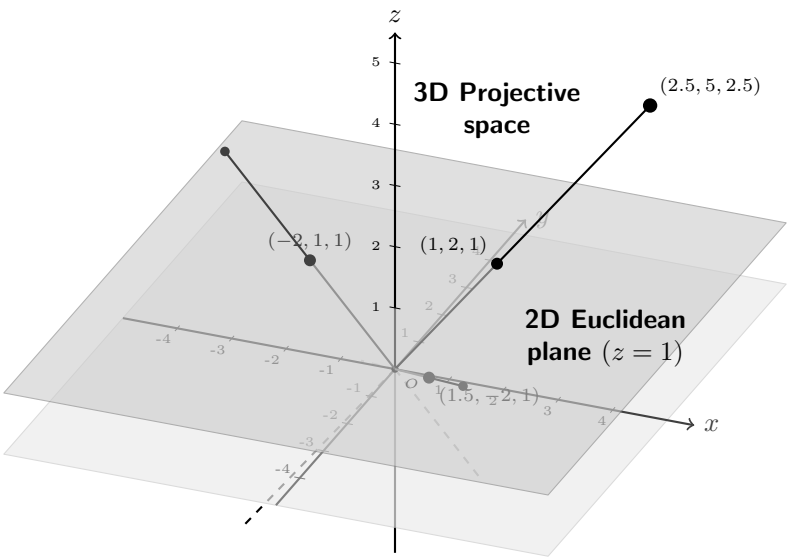


Figure 5: 3D Projective Space. This contains the projection plane at $z = 1$, which is a 2D Euclidean plane with 2D Cartesian coordinates

We can see in the diagram above that points defined in the 3D projective space can be projected onto the 2D Euclidean plane. Think of projection here as trying to capture the 2D shadow of our 3D scene.

However, we have no way to convert these coordinates from 3D to 2D...until in 1827, August Ferdinand Möbius in his work *Der barycentrische Calcul* introduced the concept of **homogeneous coordinates**. These are coordinates expressed as a ratio, such that a coordinate in the form $(x : y : 1)$ has Cartesian equivalent (x, y), which makes sense as the Euclidean plane is defined at $z = 1$. This therefore allows for points to be easily projected to various planes, e.g. the Cartesian coordinate $(1, 2)$ can be represented as homogeneous coordinates $(3 : 2 : 1)$ on the Euclidean plane $z = 1$, or as the point $(2.5 : 5 : 2.5)$ on the plane $z = 2.5$ etc.

Therefore, to project a point $(x : y : z)$ onto the Euclidean plane, all we do is divide by the z coordinates, i.e. $(x/z : y/z : 1)$, giving us 2D Cartesian coordinate $(x/z, y/z)$. In the context of computer graphics, we call this step **perspective division**.

This is already looking very promising!

3.1 Vanishing point theorem

Homogeneous coordinates also have the advantage of being able to represent the vanishing point, or the point at infinity. Remember that this is the point where we see parallel lines appear to converge at. Let's demonstrate this property in a proof to show that parallel lines meet at this point at infinity:

Consider the parallel Cartesian lines

$$ax + by + c = 0 \text{ and } ax + by + d = 0 \text{ where } c \neq d \text{ (or else they would be the same line!)}$$

Converting to homogeneous coordinates, we get

$$a\tfrac{x}{z} + b\tfrac{y}{z} + c = 0 \text{ and } a\tfrac{x}{z} + b\tfrac{y}{z} + d = 0$$
$$\implies ax + by + cz = 0 \text{ and } ax + by + dz = 0$$

Subtracting one from another,

$$cz - dz = 0 \implies z = 0 \qquad \text{(since } c \neq d\text{)}$$

Thus, the lines meet at the point $(x : y : 0)$, which is our point at infinity (since we are dividing by 0)

As a side note, the point $(x : y : 0)$ actually specifies a line of vanishing points. In fact, every family of parallel lines on the 2D Euclidean plane meet along their own line of infinite points on the $z = 0$ plane. This is evident as all points in the 3D projective space are projected from this plane - we're just using the origin as our sole vanishing point from where our camera is defined to keep things simple.

I hope this has convinced you that projective geometry is indeed the geometry of perspective.

3.2 Translation matrix with homogeneous coordinates

Along with perspective projection, another key use of homogeneous coordinates in computer graphics is to derive a translation matrix. In general (i.e. Euclidean geometry), we know translations are done by adding a translation vector, e.g.

$$\overset{\text{Point in world space}}{\begin{pmatrix} x \\ y \\ z \end{pmatrix}} + \overset{\text{Translation vector}}{\begin{pmatrix} T_x \\ T_y \\ T_z \end{pmatrix}} = \overset{\text{Translated point vector}}{\begin{pmatrix} x + T_x \\ y + T_y \\ z + T_z \end{pmatrix}}$$

However, as I discussed above, it's way more computationally efficient if we are able to represent the translation vector as a matrix that we multiply the point by instead of manually step-by-step multiplying then adding.

And homogeneous coordinates allows us to do this!

To derive how this might work, let's consider the 2x2 matrix $\begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}$.

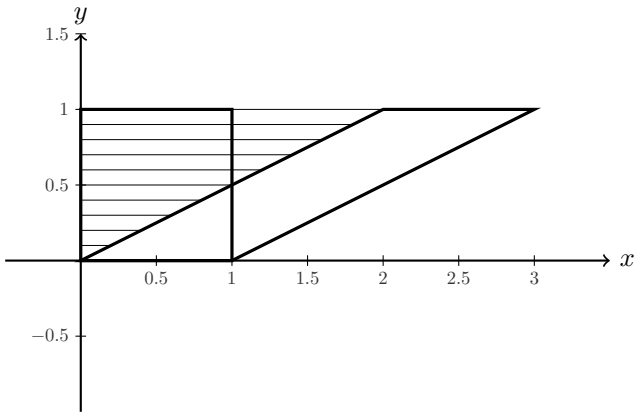


Figure 6: The effect of the shear matrix on the unit square

This is a shear transformation parallel to the x-axis that takes each point (x, y) and transforms it to the point $(x + 2y, y)$. You can see what happens to the unit square in the above diagram.

What does this mean? Essentially, each horizontal line of points $y = a$ is getting shifted (or translated!) by $2a$ units in the x direction. I've tried to show this by demonstrating what happens to the points on the left side of the unit square with the horizontal lines.

Thus, a 2D shear is a 1D translation!

And using a 1D homogeneous ordinate allows us to algebraically show this:

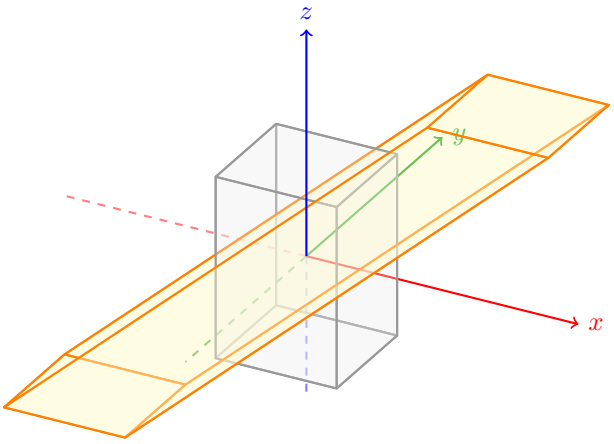
$$\begin{pmatrix} 1 & T_x \\ 0 & 1 \end{pmatrix} \times \begin{pmatrix} x \\ 1 \end{pmatrix} = \begin{pmatrix} x + T_x \\ 1 \end{pmatrix}$$

This is fantastic, let's follow the same setup and see what happens in 3D:

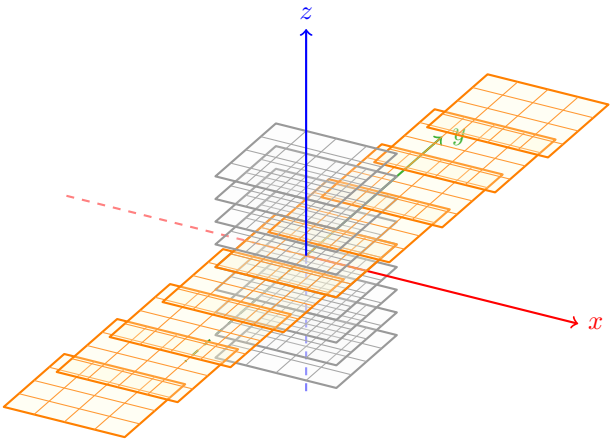
$$\overset{\text{3D shear matrix}}{\begin{pmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{pmatrix}} \times \overset{\text{2D homogeneous coordinate}}{\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}} = \overset{\text{Translated point vector}}{\begin{pmatrix} x + T_x \\ y + T_y \\ 1 \end{pmatrix}}$$

We can see how using the homogeneous coordinate system makes it very easy to derive the correct transformation matrix by manually going through the matrix multiplication row-by-row and 'reverse multiplying'.

We can visualise why this is the case by splitting the 3D cube into multiple horizontal planes, and considering only the top one. We see that this plane gets translated the most, as dictated by T_x and T_y:



The effect of the 3D shear matrix $\begin{pmatrix} 1 & 0 & T_x = 2 \\ 0 & 1 & T_y = 3 \\ 0 & 0 & 1 \end{pmatrix}$ on the unit cube



Visualising what happens in 2D by splitting cube into a stack of 2D planes.

Figure 7: Visualising the effect of the shear matrix on the unit cube

Going back to projective geometry, we can kind of see why this makes sense, as we are always concerned only with the 2D Euclidean plane at $z = 1$, which is the top plane of the 3D cube we are shearing. And, by 'taking the shadow' of only this plane, we observe that this 3D shear matrix produces a 2D translation of this plane.

As you can probably guess, the same logic applies to translating objects in 3D, where we use a 4x4 shear matrix as shown (derived the same way as the 3x3 one):

$$\overset{\text{4D shear matrix}}{\begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix}} \times \overset{\text{2D homogeneous coordinate}}{\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}} = \overset{\text{Translated point vector}}{\begin{pmatrix} x + T_x \\ y + T_y \\ z + T_z \\ 1 \end{pmatrix}}$$

Obviously, it is hard to neatly visualise the geometric interpretation of this 4-dimensional transformation, but hopefullly having seen it in 2 and 3 dimensions as well as algebraically has given you some intuition as to why this is true (an ND shear is equivalent to an (N-1)D translation!).

4 Putting the pieces together: Deriving our projection matrix

Hopefully, our discussion of homogeneous coordinates and matrices has convinced you of the beauty and usefulness of adding an extra dimension to our coordinates!

What this means for our computer graphics setup is we now use 3D homogeneous coordinates $(x : y : z : w)$ to represent the vertices of each 3D object. This means that our objects are in 3D Euclidean space within the 4D projective space, which (similar to the 2D setup discussed above), is where $w = 1$.

So, all this does to our 3D vertices is adds an extra w-coordinate of 1, which allows for all our usual matrix transformations (which now includes translations!) to be applied as normal...until the very last step when we apply our perpective projection matrix.

Going back to the first section where we were deriving our projection matrix, we have to update this slightly to take into account our new coordinate system:

$$\overset{\text{Projection matrix}}{\begin{pmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{pmatrix}} \times \overset{\text{Point in world space}}{\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}} = \overset{\text{Point on screen}}{\begin{pmatrix} x/z \\ y/z \\ 1 \end{pmatrix}}$$

From here, a couple more changes need to be made. Firstly, to go from a 4D vector to a 3D one, we have to use a 3x4 matrix.

And, using what we know about projecting points onto the 2D Cartesian using homogeneous coordinates, we can just leave the dividing each coordinate by z step at the very end, meaning we end up with:

$$\underbrace{\begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\end{pmatrix}}_{\text{Projection matrix}} \times \underbrace{\begin{pmatrix}x\\ y\\ z\\ 1\end{pmatrix}}_{\substack{\text{Point in 3D Euclidean space}\\ \text{within 4D Projective space}}} = \underbrace{\begin{pmatrix}x\\ y\\ z\end{pmatrix}}_{\text{Point in 3D Projective space}}$$

$$\xrightarrow[\text{to project onto 2D Euclidean plane}]{\text{perspective division}} \underbrace{\begin{pmatrix}x/z\\ y/z\\ 1\end{pmatrix}}_{\substack{\text{Point on the 2D Euclidean plane}\\ \text{within 3D Projective space}}} \rightarrow \underbrace{\begin{pmatrix}x/z\\ y/z\\ 1\end{pmatrix}}_{\text{Cartesian coordinate on 2D screen}}$$

Once again, I used the reverse-multiplication trick in order to figure out the values within the projection matrix.

And so, we finally have a way of projecting our 3D objects onto a 2D screen that accounts for perspective!

Although it doesn't look like much, the perspective projection matrix is doing something very important: it transforms our objects from 3D Euclidean space to 3D Projective space, where they can be easily projected onto the 2D Euclidean plane via our homogeneous coordinates and perspective division.

And most importantly, this matrix can be combined with all the other transformation matrices to give a single resultant matrix, thus resulting in a computationally efficient process!

5 Conclusion and final thoughts

In summary, we have looked into the idea of perspective and how that is encapsulated beautifully in the field of projective geometry. We have then explored how homogeneous coordinates allow all possible transformations to be encoded as a matrix. Finally, we've shown how together, matrices and homogeneous coordinates allow a very clean transformation of points in 3D Euclidean space to the 3D Projective space, where projecting them onto the 2D viewing plane is a simple matter of division.

It's truly amazing how maths developed on the basis of a concept as fundamental as perspective has been repurposed two centuries later to power the hyper-realistic 3D video games and animations of today.

Thank you for reading.

6 Sources

https://www.youtube.com/watch?v=U0_ONQQ5ZNM

https://www.youtube.com/watch?v=x1F4eFN_cos

https://g5m.cs.washington.edu/

https://en.wikipedia.org/wiki/Projective_space

https://en.wikipedia.org/wiki/Projective_geometry

https://en.wikipedia.org/wiki/Homogeneous_coordinates

https://www.scratchapixel.com/lessons/3d-basic-rendering/rendering-3d-scene-overview/perspective-projec
html

https://www.euclideanspace.com/maths/geometry/space/nonEuclid/projective/index.htm

https://www.songho.ca/math/homogeneous/homogeneous.html

https://harry7557558.github.io/tools/matrixv.html#