CO317 - Graphics (60005)

Lecture 1 - Projections and Transformations

Two Dimensional Graphics

At the lowest level, in every operating system, graphics processing operates on the pixels in a window with primitives, such as;

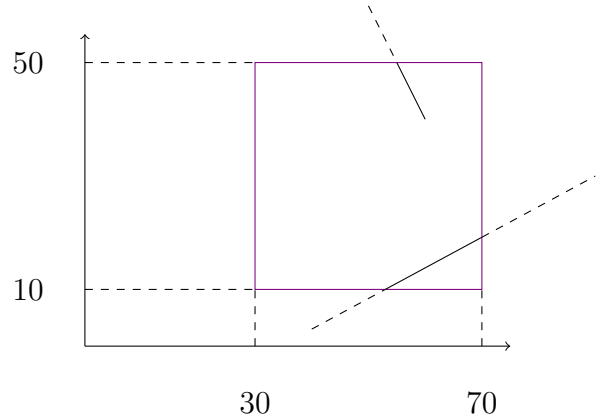
- `SetPixel(int x, int y, int colour);`
- `DrawLine(int xs, int ys, int xf, int yf);`

However, we'd like to be able to draw scenes from a three-dimensional world and have it appear in two-dimensional graphics primitives.

World Coordinate System

In order to achieve independence when drawing objects, we define a world coordinate system. For example, let our world be defined in meters, we can then allow a pixel to represent a millimetre. A viewing area is a window, and is defined as part of our 3D world. **Clipping** occurs when we attempt to draw outside (dashed) of the window;

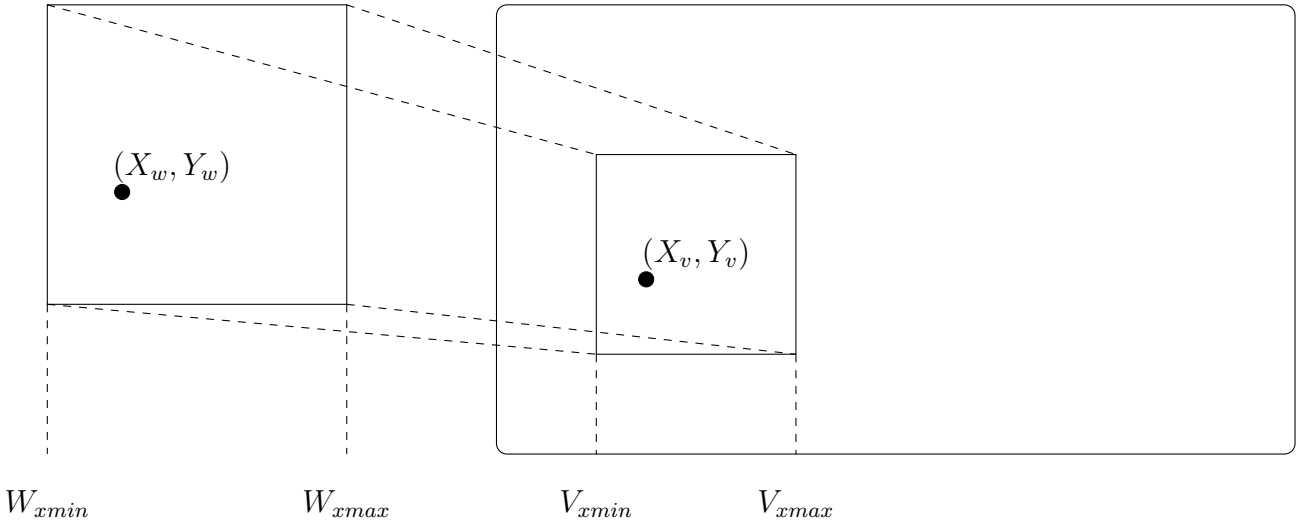
```
SetWindow(30, 10, 70, 50)
DrawLine(40, 3, 90, 30)
DrawLine(50, 60, 60, 40)
```



However, this isn't as trivial to do in 3D, as it cannot simply be left to the operating system. While we can represent 3D objects as a series of 2D commands, it's inefficient and expensive for the OS to perform the clipping (therefore we should do this manually).

Normalisation

A normalisation process is required to convert from device independent commands (where screen resolution isn't taken into account) to drawing commands using pixels. Consider a point in the world coordinate window (X_w, Y_w), and its corresponding result on the viewport (pixel coordinates; (X_v, Y_v));



The expressions are similar for Y;

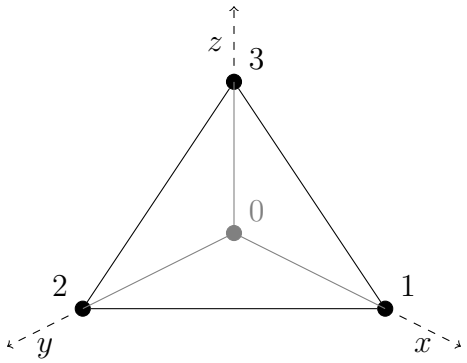
$$\frac{(X_w - W_{xmin})}{(W_{xmax} - W_{xmin})} = \frac{(X_v - V_{xmin})}{(V_{xmax} - V_{xmin})} \Rightarrow X_v = \frac{(X_w - W_{xmin})(V_{xmax} - V_{xmin})}{(W_{xmax} - W_{xmin})} + V_{xmin}$$

This gives us the resulting pair of linear equations (intuitively), where the constants found from the known values W_{xmin}, V_{xmax}, etc. are used to define the normalisation;

$$X_v = AX_w + B$$
$$Y_v = CY_w + D$$

Polygon Rendering

Most graphics applications deal with very simple objects - flat / planar polyhedra, referred to as **faces** or **facets**. These are graphic primitives, and can be used to approximate any shape. Consider the following tetrahedron, consisting of four vertices;



For this, we need a mixture of different data, including numerical data about the actual 3D coordinates of the vertices, as well as topological data regarding what vertices are connected to what. This can be represented in the following tables;

vertex data

index	location
0	$(0, 0, 0)$
1	$(1, 0, 0)$
2	$(0, 1, 0)$
3	$(0, 0, 1)$

face data

index	vertices
0	0 1 3
1	0 2 1
2	0 3 2
3	1 2 3

This separation allows for the vertices to move without affecting the faces.

Projections

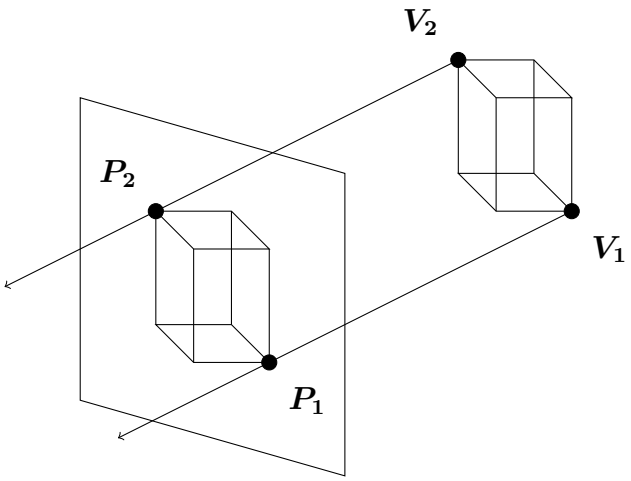
In order to draw a 3D wire frame, the points must first be converted into a 2D representation, via a **projection**, which can then be drawn with simple drawing primitives. Intuitively, we have an observer (a focal point) where all viewing rays converge. The observer is located between a projection surface P and an object V. While it's possible to project onto any surface, we only consider linear projections onto a flat surface.

Orthographic Projections

The simplest form of a projection is an **orthographic projection**. The assumptions made are that the viewpoint is located at $z = -\infty$, and the plane of projection is $z = 0$. With the viewing point being infinitely far away, the rays become parallel. This gives all projectors the same direction;

$$\boldsymbol{d} = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}$$

This gives the following, with each projection line having the equation $\boldsymbol{P} = \boldsymbol{V} + \mu\boldsymbol{d}$;



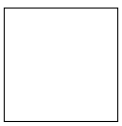
By substituting in the direction $\boldsymbol{d}$ we have determined, it gives the following Cartesian equations for each component;

$$P_x = V_x + 0$$
$$P_y = V_y + 0$$
$$P_z = V_z - \mu$$

However, since we have the projection plane $z = 0$, we also know that $P_z = 0$, therefore we don't need to solve for μ. From this, we can determine the projected location is the 3D x and y components of the vertex;

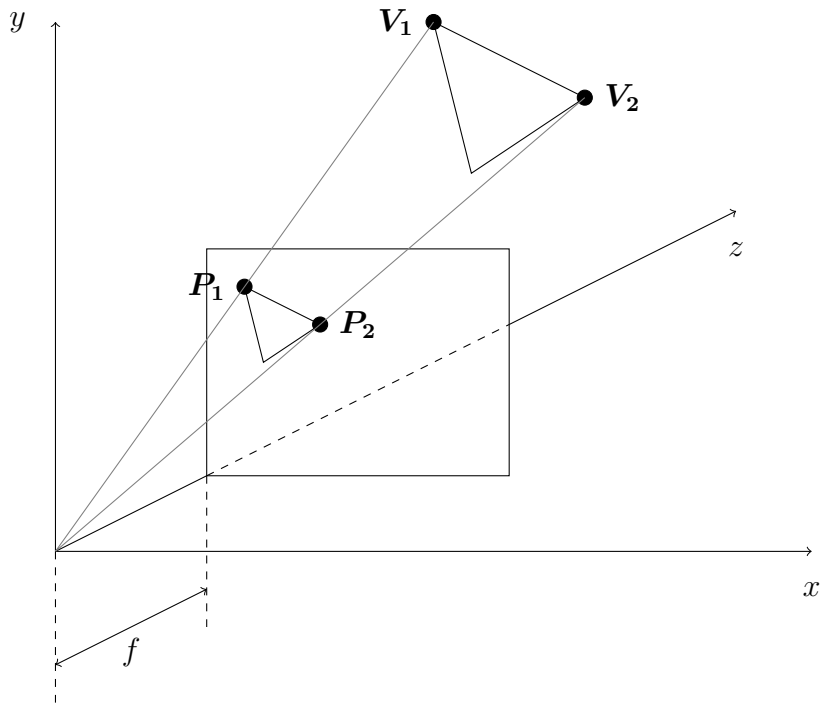
$$\boldsymbol{P} = \begin{bmatrix} V_x \\ V_y \\ 0 \end{bmatrix}$$

Viewing the wireframe for a cube directly from a face would look like the following;



Perspective Projection

While orthographic projections are fine when depth isn't a consideration (such as objects mostly being at the same distance from the viewer), it's insufficient for close work, where we want details to be realistic. The difference here is that we are no longer at an infinite distance (instead being at the origin), and the projection plane is $z = f$ (where f stands for focal length);



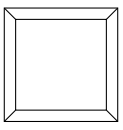
This gives us the following equation (since all projectors must go through the origin);

$$\boldsymbol{P} = \mu \boldsymbol{V}$$

We can work out the value of μ, let it be μ_p as follows;

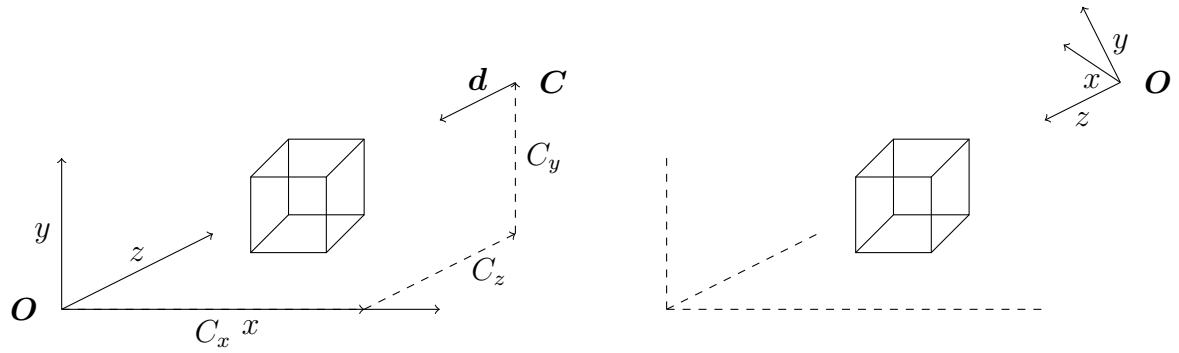
$$\begin{aligned}
P_z &= f && \text{by projection plane} \\
\mu_p &= \frac{P_z}{V_z} \\
&= \frac{f}{V_z} \\
P_x &= \mu_p V_x \\
&= \frac{f V_x}{V_z} \\
P_y &= \mu_p V_y \\
&= \frac{f V_y}{V_z}
\end{aligned}$$

Viewing the wireframe for a cube directly from a face would look like the following (note the difference to the orthographic projection);



Transformations

Scenes are defined in a particular coordinate system, but we want to be able to draw a scene from any angle. To do so, it's easier to have the viewpoint at the origin, and the z-axis as the direction of view. As such, we need to be able to **transform** the coordinates of a scene.



These are done by the application of transformation matrices. For example, a standard transformation to make an object twice as big from the origin;

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Translation

However, being restricted to matrix operations with $\mathbb{R}^{3\times 3}$ means that we cannot represent translations (for example, a shift of two units on the x-axis, such that $x' = x + 2$). The solution to this is to use 4D **homogenous coordinates**, where we assume the fourth dimension is fixed to 1.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Frequently the last ordinate is 1, however in general it is a scale factor;

$$\underbrace{(p_x, p_y, p_z, s)}_{\text{homogenous}} \Leftrightarrow \underbrace{\left(\frac{p_x}{s}, \frac{p_y}{s}, \frac{p_z}{s}\right)}_{\text{Cartesian}}$$

Affine Transformations

Affine transformations preserve parallel lines. Most of the transformations we require are affine, with the most important being scaling, rotation, and translation;

- **scaling** by (s_x, s_y, s_z)

$$\begin{bmatrix} s_x & 0 & 0 & 1 \\ 0 & s_y & 0 & 1 \\ 0 & 0 & s_z & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} s_x p_x \\ s_y p_y \\ s_z p_z \\ 1 \end{bmatrix}$$

- **rotation**

  In order to define a rotation, we need both an axis and an angle, with the simplest rotations being about the Cartesian axes. The following matrices are used for rotations of θ about each of the axes;

$$\boldsymbol{\mathcal{R}_x} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{\mathcal{R}_y} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{\mathcal{R}_z} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

It's important to note that **rotations have a direction**. In this course, we use a left-handed coordinate system, where the rotation is anti-clockwise when looking along the axis of rotation (think about the origin being closer to you, and the axis going off to ∞ away from you).

- **translation** by (t_x, t_y, t_z)

$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} p_x + t_x \\ p_y + t_y \\ p_z + t_z \\ 1 \end{bmatrix}$$

However, perspective projections are an example of a non-affine transformation, as it doesn't preserve parallels. Intuitively, it's not invertible (singular), as we cannot convert from a photograph to a 3D model.

Note that we should be careful when we combine transformation. As matrix multiplication isn't commutative, we should read a sequence of matrices multiplied together from right to left, with the right-most matrix, before the vector, being the **first** transformation to be applied.

Lecture 2 - Transformations for Animation

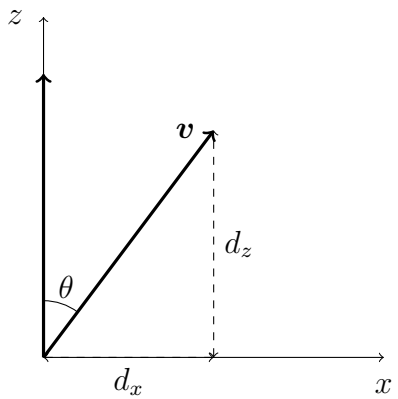
Recall the transformation in the previous lecture, which took way too long to draw, moving the origin to the view point. It consists of three steps, with the latter two being used to align the z-axis with the view direction;

1. translation of the origin fairly trivial to do

$$\boldsymbol{\mathcal{A}} = \begin{bmatrix} 1 & 0 & 0 & -C_x \\ 0 & 1 & 0 & -C_y \\ 0 & 0 & 1 & -C_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2. rotation about y-axis

   Consider the following, looking at the $x - z$ plane;

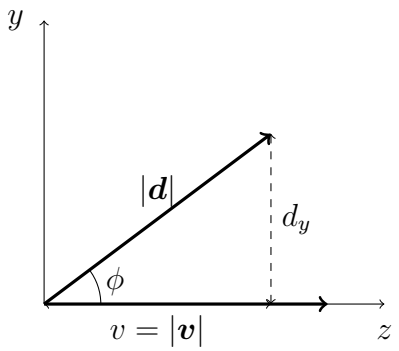


This can be used to calculate the following (note that here we are using a right-handed system);

$$
\begin{aligned}
||\boldsymbol{v}|| &= v \\
&= \sqrt{d_x^2 + d_z^2} \\
\cos\theta &= \frac{d_z}{v} \\
\sin\theta &= \frac{d_x}{v} \\
\boldsymbol{\mathcal{B}} &= \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} \frac{d_z}{v} & 0 & -\frac{d_x}{v} & 0 \\ 0 & 1 & 0 & 0 \\ \frac{d_x}{v} & 0 & \frac{d_z}{v} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned}
$$

3. rotation about x-axis

   This follows a similar process, note that we are now aligned along the $y - z$ plane (for clarity, the horizontal distance is v);



   Similarly, the matrix can be obtained as follows;

$$
\begin{aligned}
\cos\phi &= \frac{v}{|\boldsymbol{d}|} \\
\sin\phi &= \frac{d_y}{|\boldsymbol{d}|} \\
\boldsymbol{\mathcal{C}} &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi & 0 \\ 0 & \sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{v}{|\boldsymbol{d}|} & -\frac{d_y}{|\boldsymbol{d}|} & 0 \\ 0 & \frac{d_y}{|\boldsymbol{d}|} & \frac{v}{|\boldsymbol{d}|} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned}
$$

From this, we are able to combine the matrices into the following;

$$\boldsymbol{\mathcal{T}} = \boldsymbol{\mathcal{C}}\boldsymbol{\mathcal{B}}\boldsymbol{\mathcal{A}}$$

For every point $\boldsymbol{P}$ in the scene, we can obtain $\boldsymbol{P_t} = \boldsymbol{\mathcal{T}}\boldsymbol{P}$, with the view in **canonical** form, allowing us to apply the standard perspective or orthographic projection.

Rotation About a General Line

Rotation of a scene around a general line can be done as a combination of transformations. The idea is similar, with the following three steps;

1. making the line of rotation one of the Cartesian axes

   This uses the matrices derived before, rotating the general line to be aligned with the z-axis.

2. perform the rotation

   Standard rotation around the z-axis defined previously.

3. restore line to original place

   Inversion of the initial matrices to revert rotation.

This gives us the following full matrix;

$$\mathcal{T} = \underbrace{\mathcal{A}^{-1}\mathcal{B}^{-1}\mathcal{C}^{-1}}_{3}\underbrace{\mathcal{R}_z}_{2}\underbrace{\mathcal{C}\mathcal{B}\mathcal{A}}_{1}$$

Projection Matrices

For the canonical / orthographic projection, the matrix simply drops the z component;

$$\mathcal{M}_O = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathcal{M}_O \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix}$$

This is clearly non-invertible, as we are losing information about one of the axes. An effect of this is that we must do any effects in 3D **before** applying the projection matrix.

The perspective projection matrix can also be done in a similar way;

$$\mathcal{M}_p = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{f} & 0 \end{bmatrix}$$

$$\mathcal{M}_p \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ \frac{z}{f} \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} \frac{fx}{z} \\ \frac{fy}{z} \\ f \\ 1 \end{bmatrix}$$ using the fourth ordinate as a scale factor

Note that homogenous coordinates and vectors fall into one of two types;

1. **position vectors**

These have a non-zero final ordinate ($s > 0$) and can be normalised to Cartesian form. If two position vectors are added, we instead obtain the mid-point;

$$\begin{bmatrix} X_a \\ Y_a \\ Z_a \\ 1 \end{bmatrix} + \begin{bmatrix} X_b \\ Y_b \\ Z_b \\ 1 \end{bmatrix} = \begin{bmatrix} X_a + X_b \\ Y_a + Y_b \\ Z_a + Z_b \\ 2 \end{bmatrix} = \begin{bmatrix} \frac{X_a+X_b}{2} \\ \frac{Y_a+Y_b}{2} \\ \frac{Z_a+Z_b}{2} \\ 1 \end{bmatrix}$$

This has no real meaning in geometry, but is a useful observation.

2. **direction vectors**

These have a zero in the final ordinate, and have a direction and magnitude. If two direction vectors are added, we obtain a direction vector (following the normal addition rule);

$$\begin{bmatrix} x_i \\ y_i \\ z_i \\ 0 \end{bmatrix} + \begin{bmatrix} x_j \\ y_j \\ z_j \\ 0 \end{bmatrix} = \begin{bmatrix} x_i + x_j \\ y_i + y_j \\ z_i + z_j \\ 0 \end{bmatrix}$$

However, if a direction vector is added to a position vector, we obtain another position vector;

$$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} + \begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix} = \begin{bmatrix} X + x \\ Y + y \\ Z + z \\ 1 \end{bmatrix}$$

Structure of a Transformation Matrix

Note that the bottom row of such a matrix is **always** 0 0 0 1. We can decompose the columns of a transformation matrix into three direction vectors and one position vector. The three direction vectors are the new axes and the position vector is the new origin.

$$\underbrace{\begin{bmatrix} q_x & r_x & s_x & C_x \\ q_y & r_y & s_y & C_y \\ q_z & r_z & s_z & C_z \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{matrix}} \rightarrow \underbrace{\begin{bmatrix} q_x \\ q_y \\ q_z \\ 0 \end{bmatrix} \begin{bmatrix} r_x \\ r_y \\ r_z \\ 0 \end{bmatrix} \begin{bmatrix} s_x \\ s_y \\ s_z \\ 0 \end{bmatrix}}_{\text{direction vectors}} \underbrace{\begin{bmatrix} C_x \\ C_y \\ C_z \\ 1 \end{bmatrix}}_{\text{position vector}}$$

When applying this transformation to a direction vector, where the last ordinate is zero, no **translation** is applied. On the other hand, if it is a position vector, where the last ordinate is 1, all vectors will have the same displacement.

The following results can be proved by observing changes to the standard basis vectors, and the origin after applying the matrix;

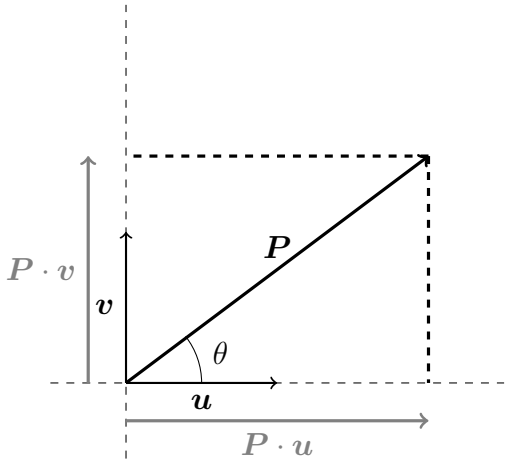
- $\boldsymbol{q}$ transformed x-axis
- $\boldsymbol{r}$ transformed y-axis
- $\boldsymbol{s}$ transformed z-axis
- $\boldsymbol{C}$ transformed origin

Dot Product

We can consider the dot product as a projection;

$$\boldsymbol{P} \cdot \boldsymbol{u} = |\boldsymbol{P}||\boldsymbol{u}| \cos\theta$$

Visually, we can see the dot product as the following;



If $\boldsymbol{u}$ is along a co-ordinate axis, then $\boldsymbol{P} \cdot \boldsymbol{u}$ is the ordinate of $\boldsymbol{P}$ in the direction of $\boldsymbol{u}$.

Consider changing to the new axes $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}$, and origin $\boldsymbol{C}$. We can call the first co-ordinate of $\boldsymbol{P}$ in the new system $\boldsymbol{P}_x^t$;

$$\begin{aligned}\boldsymbol{P}_x^t &= (\boldsymbol{P} - \boldsymbol{C}) \cdot \boldsymbol{u} \\ &= \boldsymbol{P} \cdot \boldsymbol{u} - \boldsymbol{C} \cdot \boldsymbol{u}\end{aligned}$$

However, this can also be represented in matrix notation;

$$\begin{bmatrix} P_x^t \\ P_y^t \\ P_z^t \\ 1 \end{bmatrix} = \begin{bmatrix} u_x & u_y & u_z & -\boldsymbol{C} \cdot \boldsymbol{u} \\ v_x & v_y & v_z & -\boldsymbol{C} \cdot \boldsymbol{v} \\ w_x & w_y & w_z & -\boldsymbol{C} \cdot \boldsymbol{w} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix}$$

Returning to the original problem, where we need to find a transformation matrix with a viewpoint $\boldsymbol{C}$ and direction $\boldsymbol{d}$. This can be done by first finding the vectors $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}$. Since $\boldsymbol{d}$ is the direction of the new axes, we can write (to get a unit vector);

$$\boldsymbol{w} = \frac{\boldsymbol{d}}{|\boldsymbol{d}|}$$

We also want to maintain an orthogonal basis, as well as all unit vectors. For the horizontal direction, we can write $\boldsymbol{u}$ in terms of some horizontal vector $\boldsymbol{p}$;

$$\boldsymbol{u} = \frac{\boldsymbol{p}}{|\boldsymbol{p}|}$$

$p_y = 0$ ensure horizontal, no vertical component

Similarly, we want some vertical vector $\boldsymbol{q}$ to write $\boldsymbol{v}$;

$$\boldsymbol{v} = \frac{\boldsymbol{q}}{|\boldsymbol{q}|}$$

$q_y = 1$ vertical, positive component

This gives us the following to solve;

$$\boldsymbol{p} = \begin{bmatrix} p_x \\ 0 \\ p_z \end{bmatrix}$$ new horizontal

$$\boldsymbol{q} = \begin{bmatrix} q_x \\ 1 \\ q_z \end{bmatrix}$$ new vertical

However, the view direction can also be written as the following (since we have the view direction, which happens to be the new z-axis as perpendicular to the remaining two vectors);

$$\boldsymbol{d} = \boldsymbol{p} \times \boldsymbol{q}$$

Using this, we can write $\boldsymbol{p}$ in terms of $\boldsymbol{q}$;

$$\begin{aligned}
\boldsymbol{d} &= \begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix} \\
&= \boldsymbol{p} \times \boldsymbol{q} \\
&= \begin{vmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} \\ p_x & 0 & p_z \\ q_x & 1 & q_z \end{vmatrix} \\
&= -p_z\boldsymbol{i} + (p_z q_x - p_x q_z)\boldsymbol{j} + p_x\boldsymbol{k} \\
&= \begin{bmatrix} -p_z \\ p_z q_x - p_x q_z \\ p_x \end{bmatrix} && \Rightarrow \\
d_x &= -p_z \\
d_y &= p_z q_x - p_x q_z \\
d_z &= p_x && \Rightarrow \\
\boldsymbol{p} &= \begin{bmatrix} d_z \\ 0 \\ -d_x \end{bmatrix}
\end{aligned}$$

However, since we know that the vectors $\boldsymbol{p}$ and $\boldsymbol{q}$ are orthogonal, we can say the following;

$$\begin{aligned}
\boldsymbol{p} \cdot \boldsymbol{q} &= 0 && \Rightarrow \\
p_x q_x + p_z q_z &= 0 \\
d_y &= p_z q_x - p_x q_z && \text{from above}
\end{aligned}$$

Both of these equations can be fully written in terms of $\boldsymbol{d}$.

Lecture 3 - Clipping and 3D Geometry

Clipping is the process of eliminating portions of objects outside the **viewing frustum** (bounadries of the image plane projected in 3D, consistening of a near and far clipping plane). This is useful to avoid degeneracy (by not drawing objects behind the camera), as well as improving effiency by not processing objects which won't be visible.

There are three points when we could clip;

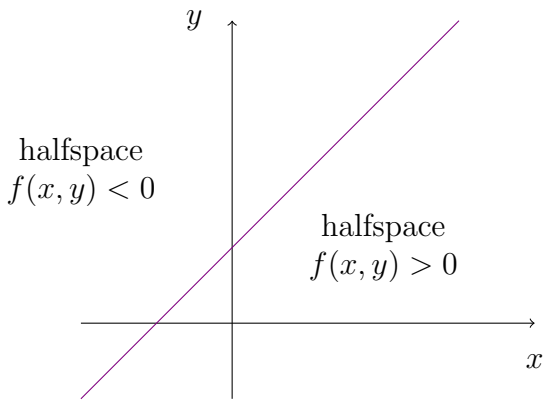
- before perspective transform (3D space) — world co-ordinates
- after perspective transform — homogeneous co-ordinates
- after perspective division — screen space

The second option, in camera co-ordinates, is ideal. This is because the clipping planes are axis aligned, as we can discard anything further than the far plane or closer than the near plane.

Halfspace

We can define any infinite line (for simplicity, in 2D) as a test;

$$f(x, y) = 0 \text{ such as } x - y + 1 = 0$$



In 3D, the plane equation is $f(x, y, z) = Ax + By + Cz + D = 0$, which also divides space into two spaces, one where the test is positive, and one where it is negative.

Note that we can define $\boldsymbol{H}$ as the normal vector of our plane, and also normalise it to avoid infinite solutions by scaling;

$$\boldsymbol{H} = \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix}$$

$$A^2 + B^2 + C^2 = 1$$

The distance is quite easily calculated as follows;

$$d = \boldsymbol{H} \cdot \boldsymbol{P} = \boldsymbol{H}^\top \boldsymbol{P}$$

This is a **signed** distance, where a positive value would denote inside, and a negative value would denote outside.

Consider the **view frustum**, where we have 6 planes, and their normals oriented towards the interior of the frustum (where each plane has its own $\boldsymbol{H}$). If $\boldsymbol{H} \cdot \boldsymbol{P} < 0$ for **any** of the planes, then it is clipped, as it would be outside of the frustum.