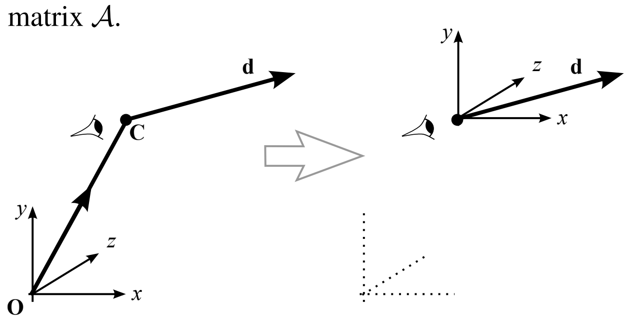
Lecture 2: Scene Transformation and Animation

Flying Sequences

We will now consider an important part of graphics processing: *scene transformation*. In any viewer-centered application, such as a flight simulator or a computer game, we need to view the scene from a moving position. As the viewpoint changes we transform all the coordinates of the scene - such that the viewpoint is the origin and the view direction is the z axis - before projecting and drawing it. Let us suppose that, in the coordinate system in which the scene is defined we wish to view it from the point $\mathbf{C} = (C_x, C_y, C_z)$, looking along the direction $\mathbf{d} = (d_x, d_y, d_z)^T$. The first step is to move the origin to $\mathbf{C}$ for which we use the transformation matrix $\mathcal{A}$.



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

Following this, we wish to rotate about the y-axis so that $\mathbf{d}$ lies in the yz plane ($x = 0$). Define $\mathbf{v}$ to be the projection of $\mathbf{d}$ onto the newly translated xz plane with magnitude v. So $v^2 = d_x^2 + d_z^2$ (See Figure 1 left). Viewing from above, along the negative y-axis, we can derive the matrix $\mathcal{B}$ to carry out the required rotation.

$$\begin{aligned} v &= \sqrt{d_x^2 + d_z^2} \\ \cos\theta &= d_z/v \\ \sin\theta &= d_x/v \end{aligned} \qquad \text{so} \quad \mathcal{B} = \begin{pmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} d_z/v & 0 & -d_x/v & 0 \\ 0 & 1 & 0 & 0 \\ d_x/v & 0 & d_z/v & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

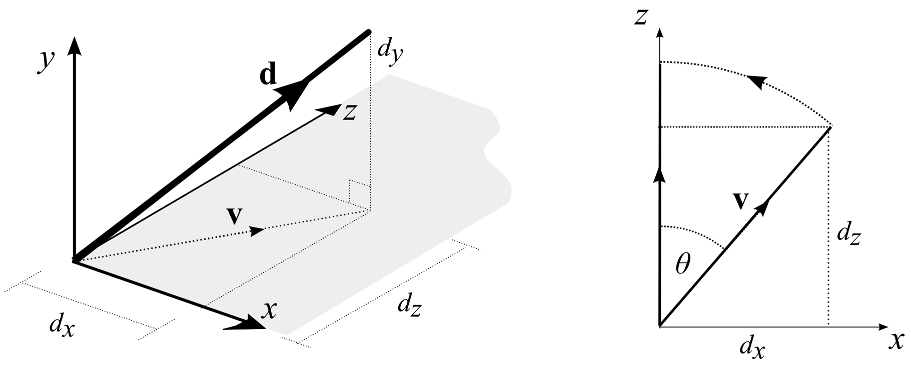


Figure 1: Rotating about the y axis so that $\mathbf{d}$ becomes aligned with the yz plane ($x = 0$). Left: A 3D view. Right: Looking from above at the xz plane.

Notice that we have avoided computing the $\cos$ and $\sin$ functions for this rotation by using the components of $\mathbf{d}$ and the magnitude of $\mathbf{v}$.

To get the direction vector lying along the z axis, a further rotation is needed. This time it is about the x axis using matrix $\mathcal{C}$. This rotation is illustrated in Figure 2.

$$\begin{aligned} v &= \sqrt{d_x^2 + d_z^2} \\ \cos\phi &= v/|\mathbf{d}| \\ \sin\phi &= d_y/|\mathbf{d}| \end{aligned} \qquad \text{so} \quad \mathcal{C} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi & 0 \\ 0 & \sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & v/|d| & -d_y/|d| & 0 \\ 0 & d_y/|d| & v/|d| & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Finally the transformation matrices are combined into a single transformation matrix by multiplying them together.

$$\mathcal{T} = \mathcal{CBA}$$

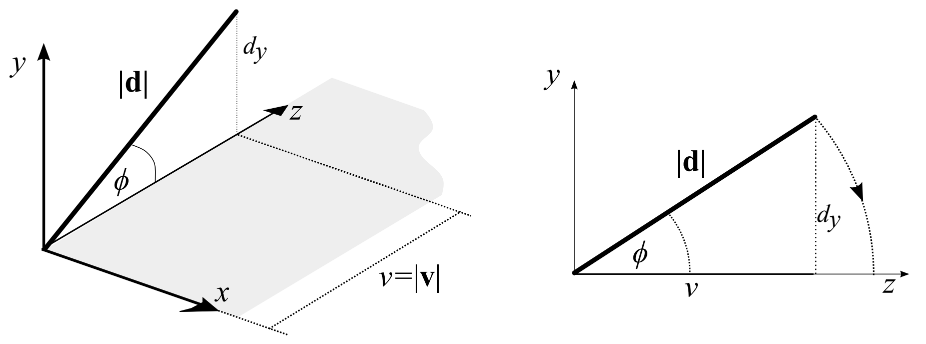


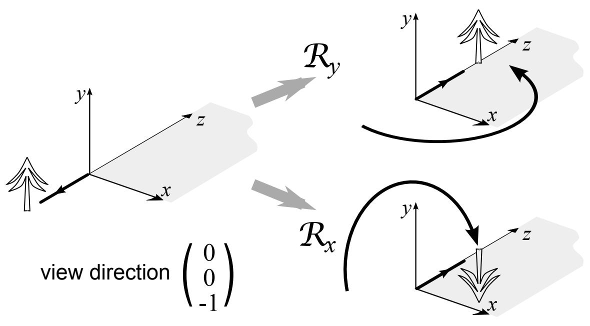
Figure 2: Rotating about the x axis so that $\mathbf{d}$ becomes aligned with the z axis.

Each point $\mathbf{P}$ of the scene is then transformed so that its new coordinates $\mathbf{P}_t$ are given by:

$$\mathbf{P}_t = \mathcal{C}\mathcal{B}\mathcal{A}\mathbf{P} = \mathcal{T}\mathbf{P}$$

Problems with verticals

In the above analysis on how to re-align the view direction, we ignored the concept of the *vertical* direction. This needs attention since the way in which the view direction is re-aligned has an effect on the vertical. For example, it is easy to invert the vertical and end up viewing the scene up side down. In the example on the right, where an object whose base is on the negative z axis is being observed from the origin along $\mathbf{d} = (0, 0, -1)^T$.



If we re-align by rotating about the y axis we get the correct solution. However, a rotation about the x axis (which also correctly aligns the view direction) will invert the image that is observed.

Animation example: Rotation about a general line

A methods used for the viewer centered transformation can also be used to solve problems regarding the animation of objects in a fixed scene. As an example, suppose that we want to rotate an object about a particular line in the Cartesian space where the scene is defined. The line can be written parametrically as $\mathbf{L} + \mu\mathbf{d}$ where μ is the parameter, $\mathbf{L} = (L_x, L_y, L_z)$ is the position vector of any point on the line and $\mathbf{d}$ is a unit direction vector along the line.

The rotation about the line that we need can be carried out in three steps:

1. Transform the scene so that the z-axis is aligned with the direction vector $\mathbf{d}$.
2. Carry out a rotation about the z-axis.
3. Transform the scene back using the inverse of the transformation in step 1.

As in the earlier case for the view transformation, we can use a translation to move the origin so that it is on the line (matrix $\mathcal{A}$ above, but transforming the origin to point $\mathbf{L}$ rather than $\mathbf{C}$). This is followed by two rotations to make the z axis coincident with the direction vector $\mathbf{d}$ (matrices $\mathcal{B}$ and $\mathcal{C}$). Now we can perform a rotation of the object about the z-axis using the standard rotation matrix $\mathcal{R}_z$ defined previously. Finally we need to restore the coordinate system as it was, such that the viewpoint is the same as before. To do this we simply invert the transformation matrices $\mathcal{A}$ $\mathcal{B}$ and $\mathcal{C}$. As before we multiply all the individual transformation matrices together to make one matrix

$$\mathcal{T} = \mathcal{A}^{-1}\,\mathcal{B}^{-1}\,\mathcal{C}^{-1}\,\mathcal{R}_z\,\mathcal{C}\,\mathcal{B}\,\mathcal{A}$$

and for all the points, calculating $\mathbf{P}_t = \mathcal{T}\,\mathbf{P}$ will achieve the required rotation.

Other object transformations for graphical animation are performed similarly. For example to make an object shrink towards a particular centre point, we move the origin to centre, perform a scaling and then restore the origin to its original position.

Projection by Matrix Multiplication

If we use homogeneous co-ordinates, it is also possible to compute projections via multiplication by a projection matrix. For a perspective projection, placing the centre of projection at the origin and using $z = f$ as before, we can use matrix $\mathcal{M}_p$ where

$$\mathcal{M}_p = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/f & 0 \end{pmatrix}$$

It is not immediately obvious that matrix $\mathcal{M}_p$ produces the correct perspective projection. Let us transform an arbitrary point $\mathbf{V}$ with homogeneous co-ordinates $(x, y, z, 1)^T$ by using matrix multiplication. For the projected point $\mathbf{P}$ we get:

$$\mathbf{P} = \mathcal{M}_p \mathbf{V} = \begin{pmatrix} x \\ y \\ z \\ z/f \end{pmatrix}$$

This point must be normalised into a Cartesian coordinate. To do this we divide the first three ordinates by the value of the fourth to get

$$\mathbf{P}_c = \begin{pmatrix} xf/z \\ yf/z \\ f \\ 1 \end{pmatrix}$$

which is the required point after perspective projection. By contrast, for an orthographic projection, with the projection plane at $z = 0$, we can use matrix $\mathcal{M}_o$

$$\mathcal{M}_o = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Check the effect this has when multiplied by a general point $\mathbf{V}$. It is interesting to note that the projection matrices are obviously singular (each has a column of zeros) and have no inverse. This must be so because it is impossible to reconstruct a 3D object from its 2D projection without other information. Projection matrices can of course be combined with the other matrices. Orthographic projection is popular as it needs fewer calculations, the z row and column fall to zero. Although a simplification also applies for the perspective projection, there is also the need to normalise the resulting homogenous coordinates which adds to the computation time.

Homogenous coordinates

Position vectors and direction vectors

We now take a second look at homogeneous coordinates, and their relation to vectors. Previously, we described the fourth ordinate as a scale factor, and ensured that, with the exception of the projection transformation, it was always normalised to 1. Alternatively, we can view the fourth ordinate as indicating the object type represented.

Informally we can distinguish between two types of vectors: *position vectors* and *direction vectors*. Postion vectors (which we often call 'Cartesian coordinates') represent a fixed point in space and direction vectors indicate a general direction but are not associated with a particular point.

If the fourth ordinate of a homogeneous coordinate is non-zero, we can divide the other ordinates by it to get a Cartesian coordinate for a fixed point in space. By contrast, if the last ordinate is zero, i.e. we have $(x, y, z, 0)$, we cannot normalise it because it is impossible to divide by zero.

So clearly a homogenous coordinate of the form $(x, y, z, 0)$ cannot be directly associated with a point in Cartesian space. However, it still has a *magnitude* and a *direction* so we can consider it to be a direction vector. Therefore homogenous coordinates fall into two classes:

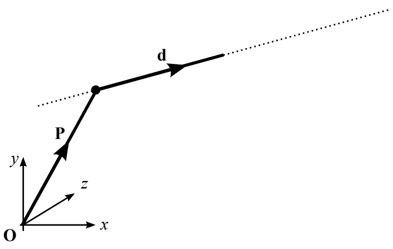
- Those with a non-zero final ordinate which can be normalised into position vectors of points.
- Those with zero final ordinate which are direction vectors, or vectors in the pure sense of the word.

Consider now how vector addition works with these definitions. If we add two direction vectors we obtain a direction vector because the fourth ordinate remains zero.

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

This is the normal vector addition rule which operates independently of Cartesian space. However, if we add a direction vector to a position vector we obtain a *position* vector:

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



This is a nice result, because it ties in with our definition of a straight line in Cartesian space, which is the sum of a point and a scaled direction.

Now, consider a general transformation matrix of the type described above. We will ignore perspective projections or shears so that the last row will always be $(0, 0, 0, 1)$. We define three direction vectors $\mathbf{q}$, $\mathbf{r}$ and $\mathbf{s}$ and a position vector $\mathbf{C}$:

$$\mathbf{q} = \begin{pmatrix} q_x \\ q_y \\ q_z \end{pmatrix} \qquad \mathbf{r} = \begin{pmatrix} r_x \\ r_y \\ r_z \end{pmatrix} \qquad \mathbf{s} = \begin{pmatrix} s_x \\ s_y \\ s_z \end{pmatrix} \qquad \mathbf{C} = \begin{pmatrix} C_x \\ C_y \\ C_z \end{pmatrix}$$

The transformation matrix will be of the form

$$\begin{pmatrix} q_x & r_x & s_x & C_x \\ q_y & r_y & s_y & C_y \\ q_z & r_z & s_z & C_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

To see what the individual columns mean we consider the effect of the transformation in simple cases. For example take the unit direction vectors along the Cartesian axes, e.g. for $\boldsymbol{i} = (1, 0, 0, 0)^T$

$$\begin{pmatrix} q_x & r_x & s_x & C_x \\ q_y & r_y & s_y & C_y \\ q_z & r_z & s_z & C_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} q_x \\ q_y \\ q_z \\ 0 \end{pmatrix}$$

In other words the direction vector in the first column represents the direction in which the x axis points after transformation. Similarly, we find that $\boldsymbol{j} = (0, 1, 0, 0)^T$ will be transformed to direction $(r_x, r_y, r_z, 0)^T$ and $\boldsymbol{k} = (0, 0, 1, 0)^T$ will be transformed to $(s_x, s_y, s_z, 0)^T$.

We can see the effect of the last column by applying the transformation to the position vector at the origin. The origin has homogeneous coordinate $(0, 0, 0, 1)^T$ and is transformed to the position vector $(C_x, C_y, C_z, 1)^T$

$$\begin{pmatrix} q_x & r_x & s_x & C_x \\ q_y & r_y & s_y & C_y \\ q_z & r_z & s_z & C_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} C_x \\ C_y \\ C_z \\ 1 \end{pmatrix}$$

Notice also that the zero in the last ordinate of direction vectors ensures that they are not affected by the translation. On the other hand, all position vectors will be moved by the same translation. If we denote arbitrary values that are known or can be calculated as a star, a direction vector can be written as $(*, *, *, 0)^T$ and an arbitrary position vector as $(*, *, *, 1)^T$, then we have

$$\begin{pmatrix} q_x & r_x & s_x & C_x \\ q_y & r_y & s_y & C_y \\ q_z & r_z & s_z & C_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} * \\ * \\ * \\ 0 \end{pmatrix} = \begin{pmatrix} * \\ * \\ * \\ 0 \end{pmatrix} \quad \begin{pmatrix} q_x & r_x & s_x & C_x \\ q_y & r_y & s_y & C_y \\ q_z & r_z & s_z & C_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} * \\ * \\ * \\ 1 \end{pmatrix} = \begin{pmatrix} * + C_x \\ * + C_y \\ * + C_z \\ 1 \end{pmatrix}$$

Notice also that if we do not apply a shear, the three vectors $\mathbf{q}$ $\mathbf{r}$ and $\mathbf{s}$ will remain orthogonal so that $\mathbf{q} \cdot \mathbf{r} = \mathbf{r} \cdot \mathbf{s} = \mathbf{q} \cdot \mathbf{s} = \mathbf{0}$.

Unfortunately however, this analysis only tells us, for a given matrix, what the old axes and origin are when transforming coordinates. It does not help us to find an unknown transformation matrix when we are given a viewpoint and view direction. However, we will still need this analysis later in the course when we look at animation.

The dot product

Let us assume that we already know the unit vectors $\boldsymbol{u}$,$\boldsymbol{v}$, and $\boldsymbol{w}$ which form the view centered coordinate system. To see how to write down a transformation matrix from these vectors we need to introduce the notion of the *dot product* (or scalar product) as a projection onto a line.

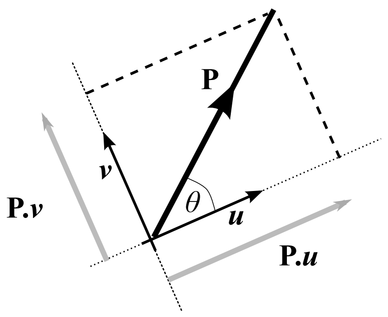


Figure 3: A 2D illustration: The dot product is used to calculate the projection of $\mathbf{P}$ along the directions of unit vectors $\boldsymbol{u}$ and $\boldsymbol{v}$.

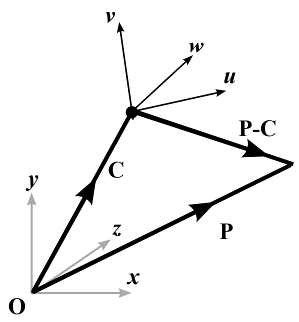


Figure 4: In 3D, the dot product can be used to get the components of $\mathbf{P}$ in the view centred coordinate system (uvw).

An illustration of the dot product in two dimensions is shown in Figure 3. By dropping perpendiculars from the point $\mathbf{P}$ to the lines defined by the unit vectors $\boldsymbol{u}$ and $\boldsymbol{v}$, the distance from the origin along each line is $\mathbf{P} \cdot \boldsymbol{u}$ and $\mathbf{P} \cdot \boldsymbol{v}$, we have

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

since $\boldsymbol{u}$ is a unit vector. Now, suppose that we wish to rotate the scene so that the x and y axes become aligned with the $\boldsymbol{u}$ and $\boldsymbol{v}$ vectors, then the x ordinate would be defined by $\mathbf{P} \cdot \boldsymbol{u}$ and the y by $\mathbf{P} \cdot \boldsymbol{v}$.

More generally, we often need to work three dimensions and the vector $\mathbf{P}$ need not start at the origin of the new axes. This situation is shown in figure 4, in which a point $\mathbf{P}$ is transformed into the (uvw) axis system translated by vector $\mathbf{C}$ from $\mathbf{O}$. The components of the transformed point $\mathbf{P}^t$ in the new system are:

$$\begin{aligned} P^t_x &= (\mathbf{P} - \mathbf{C}) \cdot \boldsymbol{u} = \mathbf{P} \cdot \boldsymbol{u} - \mathbf{C} \cdot \boldsymbol{u} \\ P^t_y &= (\mathbf{P} - \mathbf{C}) \cdot \boldsymbol{v} = \mathbf{P} \cdot \boldsymbol{v} - \mathbf{C} \cdot \boldsymbol{v} \\ P^t_z &= (\mathbf{P} - \mathbf{C}) \cdot \boldsymbol{w} = \mathbf{P} \cdot \boldsymbol{w} - \mathbf{C} \cdot \boldsymbol{w} \end{aligned}$$

This can be expressed as matrix multiplication as follows

$$\begin{pmatrix} P^t_x \\ P^t_y \\ P^t_z \\ 1 \end{pmatrix} = \begin{pmatrix} u_x & u_y & u_z & -\mathbf{C} \cdot \boldsymbol{u} \\ v_x & v_y & v_z & -\mathbf{C} \cdot \boldsymbol{v} \\ w_x & w_y & w_z & -\mathbf{C} \cdot \boldsymbol{w} \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ P_z \\ 1 \end{pmatrix}$$

So now by maintaining values of $\mathbf{C}$, $\boldsymbol{u}$, $\boldsymbol{v}$ and $\boldsymbol{w}$ throughout an animation sequence, for example by adjusting their values in response to mouse or joystick commands, we can simply write down the correct scene transformation matrix for viewing the virtual world from the correct position.

Deriving the general viewing matrix transformation

Finally, we will derive the matrix used for a view centered transformation with the viewer at point $\mathbf{C}$ and looking in direction $\mathbf{d}$. We do this by first finding the new axis system $\boldsymbol{u}$, $\boldsymbol{v}$, $\boldsymbol{w}$, adn then we can then deduce the transformation matrix as shown above.

The direction vector for viewing is $\mathbf{d} = (d_x, d_y, d_z)^T$ and this must lie along the $\boldsymbol{w}$ direction, so

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

We next find any two vectors, $\mathbf{p}$ and $\mathbf{q}$,which have the same directions as $\boldsymbol{u}$ and $\boldsymbol{v}$, but do not necessarily have unit length. We can constrain the system to preserve horizontals and verticals with two assumptions:

- Vector $\mathbf{p}$ will be in the direction of the new x-axis, so to preserve the horizontal, we choose: $p_y = 0$.
- Vector $\mathbf{q}$ is along the new y axis (vertical) and should have a positive y component (so that the picture is not upside down) so we choose (arbitrarily): $q_y = 1$.

Substituting our constraints in the Cartesian components we get:

$$\mathbf{p} = (p_x, 0, p_z)^T \qquad \text{and} \qquad \mathbf{q} = (q_x, 1, q_z)^T$$

We need to write the components of $\mathbf{p}$ and $\mathbf{q}$ in terms of d_x, d_y and d_z. Now we know that, in an axis system, $\boldsymbol{k} = \boldsymbol{i} \times \boldsymbol{j}$ so therefore we can write $\mathbf{d} = \mathbf{p} \times \mathbf{q}$ since we have not as yet constrained the magnitude of $\mathbf{p}$ or $\mathbf{q}$. Evaluating the cross product we get three equations:

$$\begin{aligned} d_x &= p_y q_z - p_z q_y \\ d_y &= p_z q_x - p_x q_z \\ d_z &= p_x q_y - p_y q_x \end{aligned}$$

Substituting the values $p_y = 0$ and $q_y = 1$ these equations simplify to

$$\begin{aligned} d_x &= -p_z \\ d_y &= p_z q_x - p_x q_z \\ d_z &= p_x \end{aligned}$$

The first and last equations give us a solution for vector $\mathbf{p} = (d_z, 0, -d_x)^T$.

To solve for $\mathbf{q}$ we use the condition that $\mathbf{p}$ and $\mathbf{q}$ are orthogonal and so have zero dot product:

$$\begin{aligned} \mathbf{p} \cdot \mathbf{q} &= d_z q_x + 0 - d_x q_z = 0 \\ \Rightarrow q_z &= d_z q_x / d_x \end{aligned}$$

and from the cross product already evaluated:

$$\begin{aligned} d_y &= p_z q_x - p_x q_z \\ &= -d_x q_x - d_z q_z \\ &= -d_x q_x - d_z^2 q_x / d_x \\ \Rightarrow q_x &= -d_y d_x / (d_x^2 + d_z^2) \end{aligned}$$

So putting the results for q_x and q_z together we get

$$\mathbf{q} = \left(\frac{-d_y d_x}{d_x^2 + d_z^2}, 1, \frac{d_y d_z}{d_x^2 + d_z^2} \right)^T$$

Finally we can obtain $\boldsymbol{u}$ and $\boldsymbol{v}$ by normalising our results for $\mathbf{p}$ and $\mathbf{q}$:

$$\boldsymbol{u} = \frac{\mathbf{p}}{|\mathbf{p}|} \qquad \text{and} \qquad \boldsymbol{v} = \frac{\mathbf{q}}{|\mathbf{q}|}$$

Thus we can deduce the whole of the transformation matrix as described in the previous section.