APPENDIX D

# Coordinate Systems

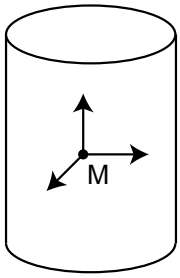
CONTENTS



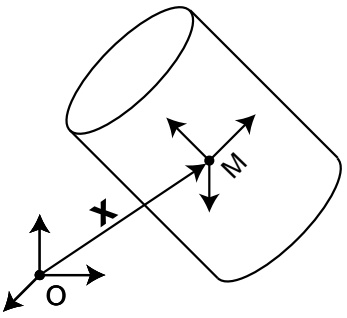
The idea of a *coordinate system*, or *coordinate frame* is pervasive in computer graphics. For example, it is usual to build a model in its own *modeling frame*, and later place this model into a scene in the *world coordinate frame*. We often refer to the modeling frame as the *object frame*, and the world coordinate frame as the *scene frame*. The figure below shows a cylinder that has been built in modeling frame **M**. To place the cylinder into the world frame **O**, the modeling frame has first been rotated about its own center, then translated to the position **x** in the world frame. Such a transformation can be encoded in a transformation matrix from the model frame to world frame,

$$M_{mw} = T_{mw}R_{mw},$$

where $T_{mw}$ encodes the translation and $R_{mw}$ encodes the rotation.



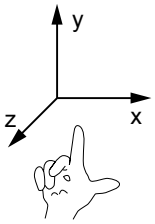

Cylinder in model frame M



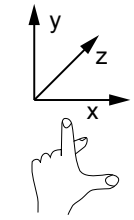

Cylinder in world frame O. The cylinder's frame has been rotated and then translated by **x**

## D.1 LEFT AND RIGHT HANDED COORDINATE FRAMES

Let us develop the idea of a coordinate frame and how we can construct them for use in computer graphics. A 3D coordinate frame might be drawn as shown in the diagram to the right. The three axes are understood to be at right angles (orthogonal) to each other. In the figure, $x$ denotes the horizontal axis, $y$ the vertical axis, and $z$ the depth axis (coming out of the page). This is the usual *right-handed* coordinate system seen in Computer Graphics.



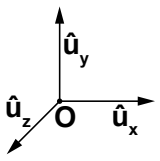

*Right Hand System*




*Left Hand System*

The coordinate system shown above is called right handed, since if you place your thumb, index finger and the middle finger of the right hand at right angles to each other, as demonstrated in the figure, they look like coordinate axes. The thumb represents the $x$ axis, the index finger represents the $y$ axis, and the middle finger represents the $z$ axis. A left handed coordinate system is shown in the figure to the left. In a left handed system, the $z$ axis is reversed, measuring depth into the page, if we keep the $x$ axis going to the right. Some older Computer Graphics texts used this convention so it is good to be aware that it exists, and what the difference is between the two conventions.

## D.2 COORDINATE FRAME EXPRESSED AS A POINT AND ORTHOGONAL UNIT VECTORS




In any coordinate system, the position where the coordinate axes cross is called the *origin*, and by definition has the coordinates $\mathbf{O} = (0, 0, 0)$ in that coordinate system. In order to work with coordinate frames in the algebraic language of vectors and matrices, we can re-label the axes of the coordinate system with unit vectors directed along the coordinate directions, as shown in the diagram to the right. We use the notation $\hat{\mathbf{u}}_x$ to represent a unit vector in the $x$ direction, $\hat{\mathbf{u}}_y$ in the $y$ direction, and $\hat{\mathbf{u}}_z$ in the $z$ direction. With this notation, a 3D point $\mathbf{p} = (p_x, p_y, p_z)$ in this coordinate frame can be rewritten

$$\mathbf{p} = \mathbf{O} + p_x\hat{\mathbf{u}}_x + p_y\hat{\mathbf{u}}_y + p_z\hat{\mathbf{u}}_z.$$

Now, if we wish to rotate our coordinate frame we can apply a rotation matrix to the vectors $\hat{\mathbf{u}}_x$, $\hat{\mathbf{u}}_y$, and $\hat{\mathbf{u}}_z$, and if we wish to translate the frame we can apply a translation matrix to $\mathbf{O}$.

## D.3 A NOTATIONAL SCHEME FOR POINTS AND VECTORS

There is a convenient notational trick that can be used to discriminate between vectors and points represented in homogeneous coordinates. We represent 3D vectors on the 4D hyperplane $w = 0$ and points on the hyperplane $w = 1$. For example, a surface normal vector might be written $\hat{\mathbf{n}} = \begin{bmatrix} \hat{n}_x \\ \hat{n}_y \\ \hat{n}_z \\ 0 \end{bmatrix}$, while a point might be written $\mathbf{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix}$. If we are consistent with this notation, then a rotation matrix will affect both points and vectors, but a translation matrix will affect only points. For example

$$\begin{bmatrix} 1 & 0 & 0 & \triangle x \\ 0 & 1 & 0 & \triangle y \\ 0 & 0 & 1 & \triangle z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{n}_x \\ \hat{n}_y \\ \hat{n}_z \\ 0 \end{bmatrix} = \begin{bmatrix} \hat{n}_x \\ \hat{n}_y \\ \hat{n}_z \\ 0 \end{bmatrix},$$

but

$$\begin{bmatrix} 1 & 0 & 0 & \triangle x \\ 0 & 1 & 0 & \triangle y \\ 0 & 0 & 1 & \triangle z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} p_x + \triangle x \\ p_y + \triangle y \\ p_z + \triangle z \\ 1 \end{bmatrix}.$$

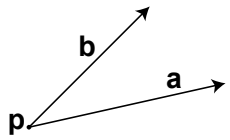
A transformation matrix that first rotates and then translates will then work perfectly for transforming both vectors and points in one frame to another frame.

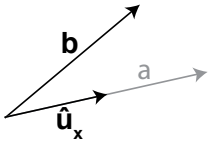
A final note is that to transform geometry from one frame to another frame we have a

choice. We can either transform the coordinate frame itself, representing this transformation by a matrix, and leave all of the points and normals in the original coordinate frame. Or, we can transform all the points and normals from the original frame to the new frame. The latter approach is referred to as *baking* the transformation. Baking is usually reserved for cases in which we have applied a set of transformations and wish to preserve the transformed shape as the new base geometry. This technique is frequently used during the modeling process, but rarely used during animation. In an animation it is preferable to keep the geometry in its original frame, and simply update the transformation matrix as the animation proceeds. If, instead, we iteratively bake the transformed geometry, we stand the risk that over many iterations numerical errors will build up causing our geometry to deform from its original shape.

## D.4 CREATING COORDINATE FRAMES

If we have two non-parallel 3D vectors and a 3D point we can use them to conveniently construct a unique 3D coordinate frame (i.e. three orthogonal directions in space, together with an origin). Let us say that we have the vectors **a** and **b**, and an origin point **p**. To describe our new coordinate frame, we would like to create three mutually perpendicular unit vectors $\hat{\mathbf{u}}_z$, $\hat{\mathbf{u}}_y$, and $\hat{\mathbf{u}}_z$, aligned with the three coordinate axes of the space.



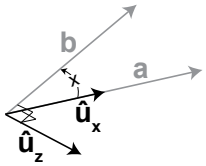



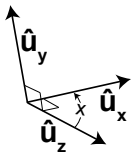

We can arbitrarily pick the $x$ axis direction to be aligned with **a**, so

$$\hat{\mathbf{u}}_x = \mathbf{a}/\|\mathbf{a}\|.$$

We know that the cross product between **a** and **b** will be perpendicular to both vectors, so let us say that this aligns with the $z$ axis, giving

$$\hat{\mathbf{u}}_z = (\mathbf{a} \times \mathbf{b})/\|\mathbf{a} \times \mathbf{b}\|.$$



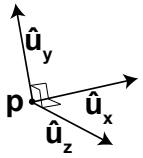




The third, or $y$ axis must be perpendicular to both $\hat{\mathbf{u}}_x$ and $\hat{\mathbf{u}}_z$, so

$$\hat{\mathbf{u}}_y = \hat{\mathbf{u}}_z \times \hat{\mathbf{u}}_x.$$

Note that $\hat{\mathbf{u}}_y$ is guaranteed to be a unit vector since both $\hat{\mathbf{u}}_x$ and $\hat{\mathbf{u}}_z$ are unit vectors, and the angle between them is $90°$ (i.e. $\sin 90° = 1$).

Providing the origin point **p** completes the construction of the coordinate frame. In this new coordinate frame, by definition the origin of the frame has coordinates $(0, 0, 0)$, and the directions of the vectors $\hat{\mathbf{u}}_x$, $\hat{\mathbf{u}}_y$, and $\hat{\mathbf{u}}_z$ are the directions of the frame's $x$, $y$, and $z$ coordinate axes.



## D.5 TRANSFORMING BETWEEN COORDINATE FRAMES

Once we have the three unit vectors and the origin point describing the new coordinate frame, it is easy to turn these into a matrix that transforms from the current frame to this new frame. Treating the current frame as the modeling frame $m$, and the new frame as the world frame $w$, we construct the rotation matrix

$$R_{mw} = \begin{bmatrix} \hat{\mathbf{u}}_x & \hat{\mathbf{u}}_y & \hat{\mathbf{u}}_z \end{bmatrix},$$

that rotates from the model to the world frame. The columns of this matrix are the three direction vectors of the world frame, expressed in model frame coordinates. You can demonstrate to yourself that this matrix works, since it rotates the three model frame coordinate axes into these new vectors:

$$R_{mw}\begin{bmatrix}1\\0\\0\end{bmatrix} = \hat{\mathbf{u}}_x, \quad R_{mw}\begin{bmatrix}0\\1\\0\end{bmatrix} = \hat{\mathbf{u}}_y, \quad \text{and} \quad R_{mw}\begin{bmatrix}0\\0\\1\end{bmatrix} = \hat{\mathbf{u}}_z.$$

We know that this matrix can do only a pure rotation since it is an *orthogonal matrix*, i.e. its column vectors are mutually orthogonal unit vectors. This is exactly the condition that must hold for a matrix to describe a pure rotation, so an orthogonal matrix is often called a *rotation matrix*. Note that unlike rotations around the three coordinate axes, this form of rotation matrix rotates around an arbitrary rotation axis.

To complete the description of the transform from the current frame to the new frame, we also need to provide a translation from the old origin to the new origin. In going from the model frame to the world frame, the origin of the model frame must be moved to the new origin at point $\mathbf{p}$. The translation matrix in homogeneous form

$$T_{mw} = \begin{bmatrix} 1 & 0 & 0 & p_x \\ 0 & 1 & 0 & p_y \\ 0 & 0 & 1 & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

will do this. Converting the rotation matrix $R_{mw}$ into homogeneous form, and multiplying the translation matrix into the rotation matrix on the left forms the complete transform from the model to world frame,

$$M_{mw} = T_{mw}R_{mw}.$$

Note that this first rotates the coordinate axes into the new frame and then translates to the new origin.

The matrix to convert back from the world frame to the model frame is simply the inverse of $M_{mw}$, and may be denoted $M_{wm}$. By application of the principles of matrix algebra, we can compute this without having to take a matrix inverse. The inverse of the product of two matrices is the product of the inverses of these matrices, but taken in the reverse order, so

$$M_{wm} = M_{mw}^{-1} = (T_{mw}R_{mw})^{-1} = R_{mw}^{-1}T_{mw}^{-1}.$$

Since $R_{mw}$ is a rotation matrix, $R_{mw}^{-1} = R_{mw}^{T}$. You can demonstrate this to yourself by multiplying $R_{mw}^{T}$ by $R_{mw}$. Because the column vectors of $R_{mw}$ are mutually orthogonal unit vectors, the row-column dot products done in computing the elements of the product matrix will yield 0 except on the diagonal where they will yield 1. Therefore,

$$R_{wm} = R_{mw}^{T} = \begin{bmatrix} \hat{\mathbf{u}}_x^T \\ \hat{\mathbf{u}}_y^T \\ \hat{\mathbf{u}}_z^T \end{bmatrix},$$

i.e. the matrix whose rows are the three direction vectors transposed. The inverse of a translation is simply an equivalent translation but in the opposite direction, so

$$T_{wm} = T_{mw}^{-1} = \begin{bmatrix} 1 & 0 & 0 & -p_x \\ 0 & 1 & 0 & -p_y \\ 0 & 0 & 1 & -p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Finally, we have the combined matrix from world space back to model space

$$M_{wm} = R_{wm} T_{wm}.$$

Note that unlike the transform from model to world space, we first translate everything back relative to the origin of the model space and then rotate back to the model space orientation.

## D.6 MATRIX-FREE TRANSFORMATIONS

Sometimes we know the origin $\mathbf{p}$ of the new coordinate frame, along with an axis of rotation $\hat{\mathbf{u}}$ and a rotation angle $\theta$. In this case, it may be advantageous to be able to directly rotate points about this axis and then translate, rather than constructing a new coordinate frame and associated transformation matrix. To do this we can first rotate a point $\mathbf{r}$ using Rodrigues' rotation formula [Murray et al., 1994], to obtain

$$\mathbf{r}' = \mathbf{r}\cos\theta + (\hat{\mathbf{u}} \times \mathbf{r})\sin\theta + (\hat{\mathbf{u}} \cdot \mathbf{r})\hat{\mathbf{u}}(1 - \cos\theta),$$

and then translate by $\mathbf{p}$ to obtain the transformed point in the new coordinate frame

$$\mathbf{r}'' = \mathbf{r}' + \mathbf{p}.$$