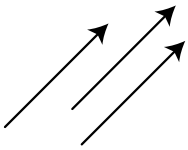
APPENDIX A

Vectors

CONTENTS

To a mathematician, a vector is the fundamental element of what is known as a vector space, supporting the operations of scaling, by elements known as scalars, and also supporting addition between vectors. When using vectors to describe physical quantities, like velocity, acceleration, and force, we can move away from this abstract definition, and stick with a more concrete notion. We can view them as arrows in space, of a particular length and denoting a particular direction, and we can think of the corresponding scalars as simply the real numbers. Practically speaking, a vector is simply a way of simultaneously storing and handling two pieces of information: a direction in space, and a magnitude or length.



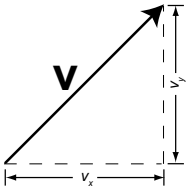
An arrow is a convenient way to draw a vector; since both length and direction are clearly indicated. A real number is a convenient way to represent a scalar, which when multiplied by a vector changes its length. To the left are three visual representations of identical vectors. They are identical, since they are all of the same length and the same direction, i.e. they are parallel to each other. Their location within the space is irrelevant.

In the study of physically based animation, we will initially be interested in vectors in two-dimensional (2D) and in three-dimensional (3D) space, whose elements are real numbers. But, we will see later that vectors can be defined in a space of any number of dimensions, with elements that may themselves be multidimensional.

Notationally, a vector is usually denoted by a lower-case letter, which has a line over it, like $\overline{v}$, or is printed in bold type, like $\mathbf{v}$. For hand written notes, the line is most convenient, but in printed form the bold form is more usual. Throughout these notes the form $\mathbf{v}$ is used.

A vector in 2D Euclidean space is defined by a pair of scalars arranged in a column, like

$$\mathbf{v} = \begin{bmatrix} v_x \\ v_y \end{bmatrix}.$$



Examining the diagram to the right, we see that v_x denotes the horizontal extent or *component* of the vector, and v_y its vertical component. Note, that in a computer program this structure can be easily represented as a two-element array of floating point numbers, or a struct containing two floats. When working in 2D, the direction of the vector can be given by the slope $m = v_y/v_x$. Its magnitude, also called its *norm*, is written $\|\mathbf{v}\|$. By the Pythagorean Theorem, $\|\mathbf{v}\| = \sqrt{v_x^2 + v_y^2}$.

A vector in 3D space is defined by three scalars arranged in a column,

$$\mathbf{v} = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix},$$

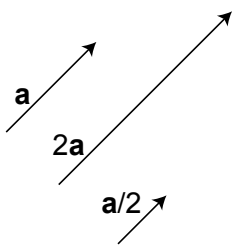
where v_x is the horizontal component, v_y the vertical component, and v_z the depth

component. The norm of a 3D vector $\mathbf{v}$ is

$$\|\mathbf{v}\| = \sqrt{v_x^2 + v_y^2 + v_z^2}.$$

In 3D there is no simple equivalent to the slope. The direction of a 3D vector is often given in terms of its azimuth and elevation. But, for our purposes it will be best understood by its corresponding unit vector, which we will describe after first defining some key algebraic vector operations.

A.1 SCALING A VECTOR



Multiplication of a vector by a real number scalar leaves the vector's direction unchanged, but multiplies its magnitude by the scalar. Algebraically, we multiply each term of the vector by the scalar. For example

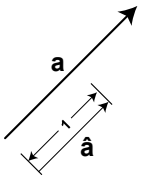
$$2\mathbf{a} = 2\begin{bmatrix} a_x \\ a_y \end{bmatrix} = \begin{bmatrix} 2a_x \\ 2a_y \end{bmatrix}.$$

Division by a scalar is the same as multiplication by the reciprocal of the scalar:

$$\mathbf{a}/2 = \begin{bmatrix} a_x/2 \\ a_y/2 \end{bmatrix}.$$

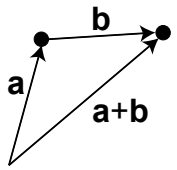
A.2 UNIT OR DIRECTION VECTORS

The direction of a vector is most easily described by a *unit vector*, also called a *direction vector*. A unit vector, for a particular vector, is parallel to that vector but of unit length. Therefore, it retains the direction, but not the norm of the parent vector. Throughout these notes the notation $\hat{\mathbf{v}}$ will be used to indicate a unit vector in the direction of parent vector $\mathbf{v}$. For example, the unit or direction vector corresponding with the 2D vector $\mathbf{a}$ would be



$$\hat{\mathbf{a}} = \begin{bmatrix} a_x/\|\mathbf{a}\| \\ a_y/\|\mathbf{a}\| \end{bmatrix} = \begin{bmatrix} \hat{a}_x \\ \hat{a}_y \end{bmatrix}.$$

A.3 VECTOR ADDITION

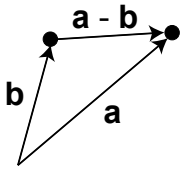


Addition of vectors can be expressed by a diagram. Placing the vectors end to end, the vector from the start of the first vector to the end of the second vector is the sum of the vectors. One way to think of this is that we start at the beginning of the first vector, travel along that vector to its end, and then travel from the start of the second vector to its end. An arrow constructed between the starting and ending points defines a new vector, which is the sum of the original vectors. Algebraically, this is equivalent to adding corresponding terms of the two vectors:

$$\mathbf{a} + \mathbf{b} = \begin{bmatrix} a_x \\ a_y \end{bmatrix} + \begin{bmatrix} b_x \\ b_y \end{bmatrix} = \begin{bmatrix} a_x + b_x \\ a_y + b_y \end{bmatrix}.$$

We can think of this as again making a trip from the start of the first vector to the end of the second vector, but this time traveling first horizontally the distance $a_x + b_x$ and then vertically the distance $a_y + b_y$.

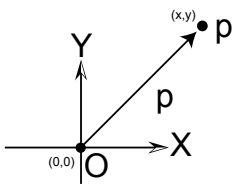
A.4 VECTOR SUBTRACTION



Subtraction of vectors can be shown in diagram form by placing the starting points of the two vectors together, and then constructing an arrow from the head of the second vector in the subtraction to the head of the first vector. Algebraically, we subtract corresponding terms:

$$\mathbf{a} - \mathbf{b} = \begin{bmatrix} a_x \\ a_y \end{bmatrix} - \begin{bmatrix} b_x \\ b_y \end{bmatrix} = \begin{bmatrix} a_x - b_x \\ a_y - b_y \end{bmatrix}.$$

A.5 POINTS AND VECTORS



This leads us to the idea that points and vectors can be interchanged — almost. While vectors can exist anywhere in space, a point is always defined relative to the origin, O. Thus, we can say that a point, $p = (x, y)$, is defined by the origin, $O = (0, 0)$ and a vector, $\mathbf{p} = \begin{bmatrix} x \\ y \end{bmatrix}$, i.e.

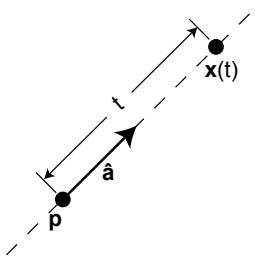
$$p = O + \mathbf{p}.$$

Because the origin is assumed to be the point $(0, 0)$, points and vectors can be represented the same way, e.g. the point $(2, 3)$ can be represented as the vector $\begin{bmatrix} 2 \\ 3 \end{bmatrix}$. This interchangeability can be very convenient in many cases, but can also lead

to confusion. It is a good idea to make sure that when storing data, you clearly indicate which values are points, and which are vectors. As will be seen below, the homogeneous coordinates used to define transformations can help with this.

Equivalent to the above, we can write, $\mathbf{p} = p - O$, i.e. a vector defines the measure from the origin to a particular point in space. More generally, a vector can always be defined by the difference between any two points, p and q. The vector $\mathbf{v} = p - q$ represents the direction and distance from point q to point p. Conversely, the point q and the vector $\mathbf{v}$ define the point, $p = q + \mathbf{v}$, which is translated from q by the components of $\mathbf{v}$.

A.6 PARAMETRIC DEFINITION OF LINES AND RAYS



This leads us to a compact definition of a line in space, written in terms of a unit vector and a point. Let $\mathbf{p}$ be a known point (expressed in vector form) on the line being defined, and let $\hat{\mathbf{a}}$ be a unit vector whose direction is parallel to the desired line. Then, the locus of points on the line is the set of all points $\mathbf{x}$, satisfying

$$\mathbf{x}(t) = \mathbf{p} + t\hat{\mathbf{a}}.$$

The variable t is a real number, and is known as the line parameter. It measures the distance from the point $\mathbf{p}$ to the point $\mathbf{x}(t)$. If t is positive, the point $\mathbf{x}$ lies in the direction of the unit vector from point $\mathbf{p}$, and if t is negative, the point lies in the direction opposite to the unit vector.

The definition of a ray is identical to the definition of a line, except that the parameter t of a ray is limited to the positive real numbers. Thus, a ray can be interpreted as starting from the point $\mathbf{p}$, and traveling in the direction of $\hat{\mathbf{a}}$ a distance corresponding to t, as t goes from 0 to increasingly large positive values. On a ray, the point $\mathbf{p}$ is called the ray origin, $\hat{\mathbf{a}}$ the ray direction, and t the distance along the ray.

A.7 DOT OR INNER PRODUCT

Vector-vector multiplication is not as easily defined as addition, subtraction and scalar multiplication. There are actually several vector products that can be defined. First, we will look at the *dot product* of two vectors, which is often called their *inner product*.

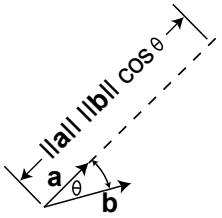
Defined algebraically, the dot product of two vectors is given by

$$\mathbf{a} \cdot \mathbf{b} = \begin{bmatrix} a_x \\ a_y \end{bmatrix} \cdot \begin{bmatrix} b_x \\ b_y \end{bmatrix} = a_x b_x + a_y b_y.$$

We multiply corresponding terms and add the result. The result *is not a vector*, but is

in fact a scalar. This turns out to have many ramifications. The dot product is a *mighty* operation and has many uses in graphics!

A.7.1 Trigonometric interpretation of dot product



The dot product can be written in trigonometric form as

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\|\|\mathbf{b}\| \cos\theta,$$

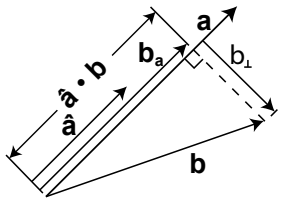
where θ is the smallest angle between the two vectors. Note, that this definition of θ applies in both 2D and 3D. Two nonparallel vectors always define a plane, and the angle θ is the angle between the vectors measured in that plane. Note that if both **a** and **b** are unit vectors, then $\|\mathbf{a}\|\|\mathbf{b}\| = 1$, and $\mathbf{a} \cdot \mathbf{b} = \cos\theta$. So, in general if you want to find the cosine of the angle between two vectors **a** and **b**, first compute the unit vectors $\hat{\mathbf{a}}$ and $\hat{\mathbf{b}}$ in the directions of **a** and **b** then

$$\cos\theta = \hat{\mathbf{a}} \cdot \hat{\mathbf{b}}.$$

Other things to note about the trigonometric representation of dot product that follow directly from the cosine relationship are that

1. the dot product of *orthogonal* (perpendicular) vectors is zero, so if $\mathbf{a} \cdot \mathbf{b} = 0$, for vectors **a** and **b** with non-zero norms, we know that the vectors must be orthogonal,
2. the dot product of two vectors is positive if the magnitude of the smallest angle between the vectors is less than 90°, and negative if the magnitude of this angle exceeds 90°.

A.7.2 Geometric interpretation of dot product



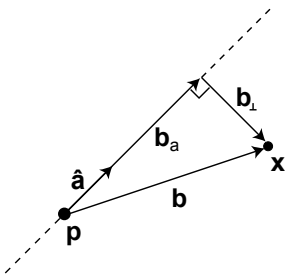
Another very useful interpretation of the dot product is that it can be used to compute the component of one vector in the direction parallel to another vector. For example, let $\hat{\mathbf{a}}$ be a unit vector in the direction of vector **a**. Then the length of the projection of another vector **b** in the direction of vector **a** is $\hat{\mathbf{a}} \cdot \mathbf{b}$. You can think of this as the length of the shadow of vector **b** on vector **a**. Therefore, the vector component of **b** in the direction of **a** is

$$\mathbf{b}_a = (\hat{\mathbf{a}} \cdot \mathbf{b})\hat{\mathbf{a}}.$$

So, $\mathbf{b}_a$ is parallel to **a** and has length equal to the projection of **b** onto **a**. Note also that $\mathbf{b}_\perp = \mathbf{b} - \mathbf{b}_a$ will be the component of **b** perpendicular to vector **a**.

The dot product has many uses in graphics that the following two examples will serve to illustrate.

A.7.3 Dot product example: The distance from a point to a line



Let us look at how dot product can be used to compute an important geometric quantity: the distance from a point to a line. We will use the parametric definition of a line, described above, specified by point $\mathbf{p}$ and a direction vector $\hat{\mathbf{a}}$. To compute the distance of an arbitrary point $\mathbf{x}$ from this line, first compute the vector $\mathbf{b} = \mathbf{x} - \mathbf{p}$, from the point $\mathbf{p}$ on the line to the point $\mathbf{x}$. The component of $\mathbf{b}$ in the direction of vector $\hat{\mathbf{a}}$ is

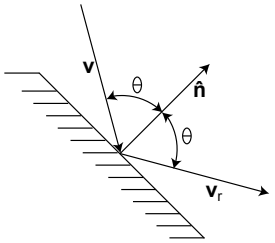
$$\mathbf{b}_a = (\hat{\mathbf{a}} \cdot \mathbf{b})\hat{\mathbf{a}}.$$

The component of $\mathbf{b}$ perpendicular to $\mathbf{a}$ is

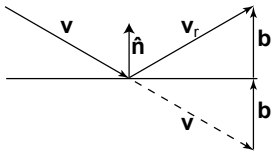
$$\mathbf{b}_\perp = \mathbf{b} - \mathbf{b}_a,$$

and the distance of point $\mathbf{x}$ from the line is simply $\|\mathbf{b}_\perp\|$.

A.7.4 Dot product example: Mirror reflection



Another very useful example of the use of dot product in geometric calculations is the computation of the mirror reflection from a surface. Assume that we have a flat mirror surface, whose *surface normal* is the unit vector $\hat{\mathbf{n}}$. The surface normal is defined to be a direction vector perpendicular to the surface. Since there are two such vectors at any point on a surface, the convention is to take the direction of the surface normal to be pointing in the "up" direction of the surface. For example, on a sphere it would point out of the sphere, and on a plane it would point in the direction considered to be the top of the plane. Now, we shine a light ray with direction $\mathbf{v}$ at the surface. The direction of the reflected ray will be given by $\mathbf{v}_r$. What must be true is that the angle θ between the normal $\hat{\mathbf{n}}$ and the light ray $\mathbf{v}$ should be the same as the angle between the reflected ray and the normal, and all three vectors $\mathbf{v}$, $\hat{\mathbf{n}}$, and $\mathbf{v}_r$ must lie in the same plane. Given these constraints, below is one way to calculate the light reflection ray $\mathbf{v}_r$.



To make the figure to the left, we first rotated the scene so everything is in a convenient orientation, with the surface normal $\hat{\mathbf{n}}$ pointing vertically, and the surface horizontal. Now, move vector $\mathbf{v}$ so that its tail is at the reflection point, as shown by the vector drawn with a dashed line in the figure. If $\mathbf{b}$ is the vector parallel to $\hat{\mathbf{n}}$ from the head of $\mathbf{v}$ to the surface, then by vector addition we have

$$\mathbf{v}_r = \mathbf{v} + 2\mathbf{b}.$$

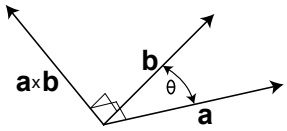
Now the vector $\mathbf{b}$ is just the negative of the component of $\mathbf{v}$ in the direction of $\hat{\mathbf{n}}$. So,

$$\mathbf{b} = -(\hat{\mathbf{n}} \cdot \mathbf{v})\hat{\mathbf{n}}.$$

Thus,

$$\mathbf{v}_r = \mathbf{v} - 2(\hat{\mathbf{n}} \cdot \mathbf{v})\hat{\mathbf{n}}.$$

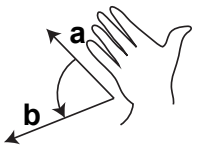
A.8 CROSS PRODUCT



The cross product $\mathbf{a} \times \mathbf{b}$ between two vectors $\mathbf{a}$ and $\mathbf{b}$ is a new vector perpendicular to the plane defined by the original two vectors. In other words, the cross product of two vectors is a vector that is perpendicular to both of the original vectors. The figure to the left illustrates the construction.

This notion of cross product does not make sense in 2D space, since it is not possible for a third 2D vector to be perpendicular to two (non parallel) 2D vectors. Thus, in graphics, the notion of cross product is reserved for working in 3D space.

Since there are two directions perpendicular to the plane formed by two vectors, we must have a convention to determine which of these two directions to use. In graphics, it is most common to use the *right hand rule*, and we use this convention throughout this text. The right-hand rule works as follows. Hold your right hand out flat, with the thumb out, aligning the fingers so they point in the direction of $\mathbf{a}$.



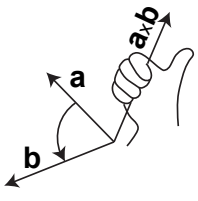
Now, rotate your hand so you can curl your fingers in the direction from vector $\mathbf{a}$ to vector $\mathbf{b}$. Your thumb will point in the direction of $\mathbf{a} \times \mathbf{b}$. If you reverse this, and first align your fingers with $\mathbf{b}$ and then curl them towards $\mathbf{a}$ you will see that you have to turn your hand upside down, reversing the direction in which your thumb is pointing. From this it should be apparent that $\mathbf{b} \times \mathbf{a} = -(\mathbf{a} \times \mathbf{b})$. In other words, the order of the operands in the cross product changes the polarity of the resulting cross product vector. The result is still perpendicular to both of the original vectors, but the direction is flipped.

A.8.1 Trigonometric interpretation of cross product

The magnitude of the cross product is given by

$$||\mathbf{a} \times \mathbf{b}|| = ||\mathbf{a}||||\mathbf{b}||| \sin \theta|,$$

where θ is the small angle between vectors $\mathbf{a}$ and $\mathbf{b}$. Thus, if $\mathbf{a}$ and $\mathbf{b}$ are unit vectors, the magnitude of the cross product is the magnitude of $\sin \theta$.



Note, that the cross product of two parallel vectors will be the

zero vector **0**. This is consistent with the geometric notion that the cross product produces a vector orthogonal to the original two vectors. If the original vectors are parallel, then there is no unique direction perpendicular to both vectors (i.e. there are infinitely many orthogonal vectors, all parallel to any plane perpendicular to either vector).

Algebraically, the cross product is defined as follows. If two vectors are defined

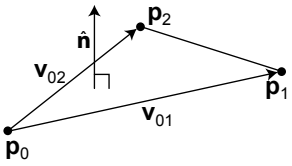
$$\mathbf{a} = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}, \text{ and } \mathbf{b} = \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix},$$

then

$$\mathbf{a} \times \mathbf{b} = \begin{bmatrix} a_y b_z - a_z b_y \\ a_z b_x - a_x b_z \\ a_x b_y - a_y b_x \end{bmatrix}.$$

The cross product has many uses in graphics, which the following two examples will serve to illustrate.

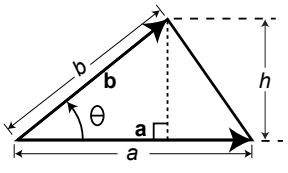
A.8.2 Cross product example: Finding surface normals



Suppose we have triangle $(\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2)$, and we want to find the triangle's surface normal. We can do this easily by use of a cross product operation. First, define vectors along two of the triangle edges: $\mathbf{v}_{01} = \mathbf{p}_1 - \mathbf{p}_0$, and $\mathbf{v}_{02} = \mathbf{p}_2 - \mathbf{p}_0$. Then the cross product $\mathbf{v}_{01} \times \mathbf{v}_{02}$ is a vector perpendicular to both $\mathbf{v}_{01}$ and $\mathbf{v}_{02}$, and therefore perpendicular to the plane of the triangle. Scaling this vector to a unit vector yields the surface normal vector

$$\hat{\mathbf{n}} = (\mathbf{v}_{01} \times \mathbf{v}_{02}) / \|\mathbf{v}_{01} \times \mathbf{v}_{02}\|.$$

A.8.3 Cross product example: Computing the area of a triangle



Another application of cross product to triangles uses the trigonometric definition of the magnitude of the cross product. Suppose we have a triangle, like the one shown to the right. If we know the lengths of sides a and b, and we know the angle θ between these sides, the area computation is straightforward. Relative to side a, the height of the triangle is given by $h = b \sin\theta$, and we know that the area of the triangle is $A = 1/2ah$, so we have $A = 1/2ab \sin\theta$. If we represent the sides of the triangle by vectors **a** and **b**, $a = \|\mathbf{a}\|$ and $b = \|\mathbf{b}\|$. Since the magnitude of the cross product $\|\mathbf{a} \times \mathbf{b}\| = \|\mathbf{a}\|\|\mathbf{b}\| \sin\theta|$, it follows that

$$A = 1/2\|\mathbf{a} \times \mathbf{b}\|.$$

APPENDIX B

Matrix Algebra

CONTENTS

Although it is the intent of this book to be reasonably self contained, the subject of matrices and matrix algebra is a complex topic, subsumed under the field of Linear Algebra. What we are attempting in this section is to give a simple, and practical overview of some of the basic principles of matrix algebra that will be essential to the introductory study of physically based animation. The student who wishes to go on in computer graphics is strongly encouraged to make a thorough study of Linear Algebra, since it furnishes many of the key mathematical tools necessary to understand advanced texts and research papers in the field.

B.1 MATRIX DEFINITIONS

A single real number is called a *scalar*. If we have a column of scalars, we have a *vector*. A set of vectors, each with the same number of entries and arranged in a rectangular array is called a *matrix*. [1] This construction can be continued to higher dimensions, collecting a set of matrices together to form a *tensor*. Abstractly, all of these objects are considered to be tensors of various orders: a scalar is an order 0 tensor, a vector an order 1 tensor, and a matrix an order 2 tensor.

The individual scalars making up a matrix are called its *elements*. An arrangement of n horizontal rows, and m vertical columns is called an $n \times m$ matrix. An example would be

[1] Please note the construction of the plural: one matrix, two matrices.

the matrix

$$M = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}.$$

M's elements are the scalars a through i, whose values are real numbers. M is a 3×3 matrix, consisting of the three rows:

$$\begin{bmatrix} a & b & c \end{bmatrix}, \begin{bmatrix} d & e & f \end{bmatrix}, \begin{bmatrix} g & h & i \end{bmatrix};$$

and the three columns:

$$\begin{bmatrix} a \\ d \\ g \end{bmatrix}, \begin{bmatrix} b \\ e \\ h \end{bmatrix}, \begin{bmatrix} c \\ f \\ i \end{bmatrix}.$$

Because M has the same number of rows as columns, it is called a *square matrix*. Since the columns of a matrix, taken individually, are really vectors, they are called *column vectors*. Similarly, the rows of a matrix, taken individually, are called *row vectors*. The sequence of elements of a square matrix forming the diagonal from the upper left to the lower right corner is called the *diagonal* of the matrix. All other elements of the matrix are called the *off diagonal* elements. Our example matrix M has the diagonal $\begin{bmatrix} a & e & i \end{bmatrix}$.

The *transpose* of a matrix is constructed by interchanging its rows and columns. Thus, the transpose of an $m \times n$ matrix will be an $n \times m$ matrix. Returning to our example, the transpose of matrix M is

$$M^T = \begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix}.$$

Note, that the row vectors of the original matrix are now the column vectors of the transpose. Likewise, the column vectors are now the row vectors of the transpose.

We can unify the notions of vector and matrix if we consider a column vector of n elements to be an $n \times 1$ matrix. Similarly, a row vector of n elements can be considered to be a $1 \times n$ matrix. If we think of a vector in this way, we can take its transpose, turning a column vector into a row vector. If $\mathbf{v} = \begin{bmatrix} v_x \\ v_y \end{bmatrix}$, then $\mathbf{v}^T = \begin{bmatrix} v_x & v_y \end{bmatrix}$.

The *determinant* of a matrix is a scalar value, written $|M|$. The determinant is defined only for square matrices. For small matrices it is defined as follows:

$$2 \times 2\text{:}\quad M = \begin{bmatrix} a & b \\ c & d \end{bmatrix},\quad |M| = ad - bc,$$

$$3 \times 3\text{:}\quad M = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix},\quad |M| = aei + bfg + cdh - (ceg + bdi + afh).$$

For larger matrices, the definition of the determinant becomes more complex, and the reader is referred to a more advanced text.

Matrix multiplication is defined between matrices of compatible dimensions. An $a \times b$

matrix multiplied by a $b \times c$ matrix yields an $a \times c$ matrix. For example, multiplying a 3×3 matrix by another 3×3 matrix gives you another 3×3 matrix, and multiplying a 2×2 matrix by a 2×1 vector yields another 2×1 vector.

To understand how matrix multiplication works, let us first consider the multiplication of a matrix by a vector, $M\mathbf{v}$. One way to understand this operation is to treat the rows of the matrix as row vectors. The elements of the result vector are formed by taking the dot product of each row by the vector. Using, as our example, the matrix $M = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ and the vector $\mathbf{v} = \begin{bmatrix} v_x \\ v_y \end{bmatrix}$, the resulting vector is given by

$$M\mathbf{v} = \begin{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} \cdot \begin{bmatrix} v_x \\ v_y \end{bmatrix} \\ \begin{bmatrix} c \\ d \end{bmatrix} \cdot \begin{bmatrix} v_x \\ v_y \end{bmatrix} \end{bmatrix} = \begin{bmatrix} av_x + bv_y \\ cv_x + dv_y \end{bmatrix}.$$

Another way to understand this operation is to think of the elements of the vector as scaling each of the column vectors of the matrix, which then are added together:

$$M\mathbf{v} = v_x \begin{bmatrix} a \\ c \end{bmatrix} + v_y \begin{bmatrix} b \\ d \end{bmatrix} = \begin{bmatrix} av_x + bv_y \\ cv_x + dv_y \end{bmatrix}.$$

Matrix–matrix multiplication works like matrix-vector multiplication, treating each column of the second matrix as a vector being multiplied by the first matrix, and returning a new column. For example

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} \cdot \begin{bmatrix} e \\ g \end{bmatrix} & \begin{bmatrix} a \\ b \end{bmatrix} \cdot \begin{bmatrix} f \\ h \end{bmatrix} \\ \begin{bmatrix} c \\ d \end{bmatrix} \cdot \begin{bmatrix} e \\ g \end{bmatrix} & \begin{bmatrix} c \\ d \end{bmatrix} \cdot \begin{bmatrix} f \\ h \end{bmatrix} \end{bmatrix} = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}.$$

The multiplication operator requires an identity element and an inverse to complete its definition. The identity element for matrix–matrix or matrix–vector multiplication is the identity matrix I, whose diagonal elements are all 1's and whose off diagonal elements are all 0's. The 2D and 3D identity matrices are

$$I_{2x2} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \text{ and } I_{3x3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Now, we can define the inverse of a matrix M^{-1}, as that matrix, which when multiplied by the original matrix M, yields the identity matrix I, or

$$MM^{-1} = M^{-1}M = I.$$

For a 2×2 matrix, the inverse is given by

$$M = \begin{bmatrix} a & b \\ c & d \end{bmatrix},$$

$$M^{-1} = \frac{1}{|M|} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}.$$

For a 3×3 matrix, the inverse is given by

$$M = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix},$$

$$M^{-1} = \frac{1}{|M|} \begin{bmatrix} ei - fh & ch - bi & bf - ce \\ fg - di & ai - cg & cd - af \\ dh - eg & bg - ah & ae - bd \end{bmatrix}.$$

For larger matrices, the reader is again referred to a more advanced text.

A word of caution is necessary with regard to matrix inverse. First, a matrix that is non-square has no inverse. Second, as we can see from the equations above, computing the inverse of a matrix involves dividing by the determinant of the matrix. If this determinant is 0, the matrix inverse is indeterminate.

B.2 SYSTEMS OF LINEAR EQUATIONS

Matrices arose in an attempt to develop a compact algebra to describe the solution of sets of simultaneous linear equations. Suppose we have two variables x and y, and we want to find the solutions for x and y satisfying the equations

$$ax + by = u$$
$$cx + dy = v$$

One way to solve these equations would be to begin by solving the first equation for $x = (u - by)/a$, and then substitute this expression for x into the second equation, leaving a single equation with only the variable y, which has the solution $y = (av - cu)/(ad - cb)$. Inserting the solution for y into the equation for x gives us $x = (du - bv)/(ad - cb)$.

In Linear Algebra, the original pair of equations would be written in the form of a matrix and two vectors:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} u \\ v \end{bmatrix},$$

or more abstractly

$$M\mathbf{x} = \mathbf{u},$$

where M, $\mathbf{x}$, and $\mathbf{u}$ have the obvious definitions from the original expanded form of the equation. Notationally, we have gone from the more cumbersome original pair of linear equations to a more compact algebraic expression.

We now have the machinery to solve a linear system of equations written in matrix form:

$$M\mathbf{x} = \mathbf{u},$$
$$M^{-1}M\mathbf{x} = M^{-1}\mathbf{u},$$
$$I\mathbf{x} = M^{-1}\mathbf{u},$$
$$\mathbf{x} = M^{-1}\mathbf{u}.$$

Applying this logic to our two equation example from the start of this section, we have

$$\mathbf{x} = M^{-1}\mathbf{u} = \frac{1}{ad-bc}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} (du-bv)/(ad-bc) \\ (av-cu)/(ad-bc) \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix},$$

which matches the solutions for x and y obtained by the original substitution process.

Note that there are cases where a system of linear equations will not yield a unique solution. These correspond exactly with cases where the determinant of the system's matrix is 0. For example, suppose we had the following two word-problems regarding a collection of apples and bananas. In both problems, let a be the number of apples, and b be the number of bananas.

1. There are twice as many apples as bananas. The total number of pieces of fruit is 30.

$$a - 2b = 0, \quad a + b = 30$$
$$M\mathbf{x} = \mathbf{u}, \text{ with } M = \begin{bmatrix} 1 & -2 \\ 1 & 1 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} a \\ b \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} 0 \\ 30 \end{bmatrix}$$
$$\mathbf{x} = M^{-1}\mathbf{u}, \text{ with } |M| = 1/3, \quad M^{-1} = \begin{bmatrix} 1 & 2/3 \\ -1/3 & 1/3 \end{bmatrix}$$
$$\mathbf{x} = \begin{bmatrix} 20 \\ 10 \end{bmatrix}$$

2. There are twice as many apples as bananas. There are half as many bananas as apples.

$$a - 2b = 0, \quad -1/2a + b = 0$$
$$M\mathbf{x} = \mathbf{u}, \text{ with } M = \begin{bmatrix} 1 & -2 \\ -1/2 & 1 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} a \\ b \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$
$$\mathbf{x} = M^{-1}\mathbf{u}, \text{ with } |M| = 0, \text{ so } M^{-1} \text{is indeterminate}$$

One can easily see why there is no unique solution to the second problem. It is because the second condition, "There are half as many bananas as apples," adds no new information to the problem. We already knew that there are twice as many apples as bananas. The consequence, in the formation of matrix M, is that the second row of the matrix is just a scalar multiple (-1/2) of the first row.

In general, a matrix that has a row that is a linear combination (i.e. a weighted sum) of one or more of the other rows will have a determinant of 0, and thus no inverse. Such a matrix is called degenerate, and indicates that the original problem does not have enough information to yield a unique solution.

APPENDIX C

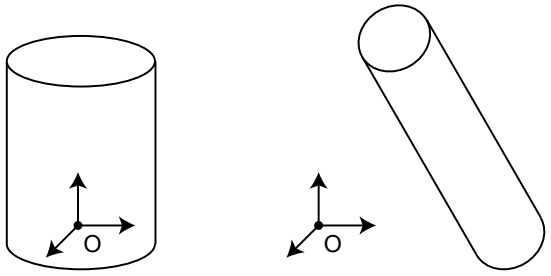
Affine Transformations

CONTENTS

C.1 THE NEED FOR GEOMETRIC TRANSFORMATIONS

One could imagine a computer graphics system that requires the user to construct everything directly into a single scene. But, one can also immediately see that this would be an extremely limiting approach. In the real world, things come from various places and are arranged together to create a scene. Further, many of these things are themselves collections of smaller parts that are assembled together. We may wish to define one object relative to another – for example we may want to place a hand at the end of an arm. Also, it is often the case that parts of an object are similar, like the tires on a car. And, even things that are built on scene, like a house for example, are designed elsewhere, at a scale that is usually many times smaller than the house as it is built. Even more to the point, we will often want to animate the objects in a scene, requiring the ability to move them around relative to each other. For animation we will want to be able to move not only the objects, but also the camera, as we render a sequence of images as time advances to create an illusion of motion. We need good mechanisms within a computer graphics system to provide the flexibility implied by all of the issues raised above.

The figure below shows an example of what we mean. On the left, a cylinder has been built in a convenient place, and to a convenient size. Because of the requirements of a scene, it is first scaled to be longer and thinner than its original design, rotated to a desired orientation in space, and then moved to a desired position (i.e. translated). The set of operations providing for all such transformations, are known as the *affine transforms*. The affines include translations and all linear transformations, like scale, rotate, and shear.



Original cylinder model

Transformed cylinder. It has been scaled, rotated, and translated

C.2 AFFINE TRANSFORMATIONS

Let us first examine the affine transforms in 2D space, where it is easy to illustrate them with diagrams, then later we will look at the affines in 3D.

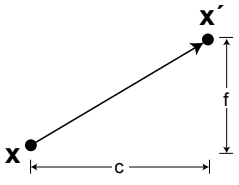
Consider a point $\mathbf{x} = (x, y)$. Affine transformations of $\mathbf{x}$ are all transforms that can be written

$$\mathbf{x}' = \begin{bmatrix} ax + by + c \\ dx + ey + f \end{bmatrix},$$

where a through f are scalars.

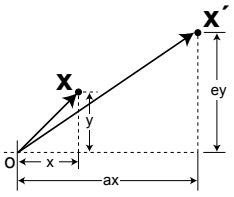
For example, if $a, e = 1$, and $b, d = 0$, then we have a pure translation

$$\mathbf{x}' = \begin{bmatrix} x + c \\ y + f \end{bmatrix}.$$



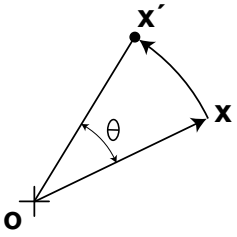
If $b, d = 0$ and $c, f = 0$ then we have a pure scale.

$$\mathbf{x}' = \begin{bmatrix} ax \\ ey \end{bmatrix}$$



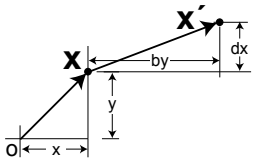
And, if $a, e = \cos\theta$, $b = -\sin\theta$, $d = \sin\theta$, and $c, f = 0$, then we have a pure rotation about the origin

$$\mathbf{x}' = \begin{bmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \end{bmatrix}.$$



Finally if $a, e = 1$, and $c, f = 0$ we have the shear transforms

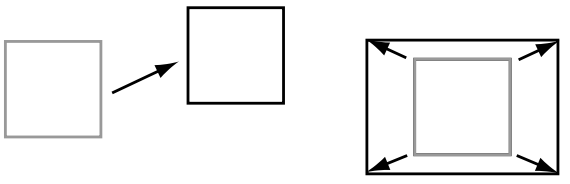
$$\mathbf{x}' = \begin{bmatrix} x + by \\ y + dx \end{bmatrix}.$$

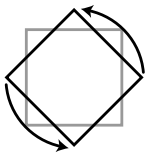


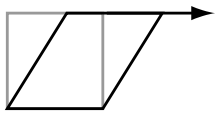
In summary, we have the four basic affine transformations shown in the figure below:

- Translate moves a set of points a fixed distance in x and y,
- Scale scales a set of points up or down in the x and y directions,
- Rotate rotates a set of points about the origin,
- Shear offsets a set of points a distance proportional to their x and y coordinates.

Note that only shear and scale change the shape determined by a set of points.







C.3 MATRIX REPRESENTATION OF THE LINEAR TRANSFORMATIONS

The affine transforms scale, rotate and shear are actually linear transforms and can be represented by a matrix multiplication of a point represented as a vector,

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} ax + by \\ dx + ey \end{bmatrix} = \begin{bmatrix} a & b \\ d & e \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix},$$

or $\mathbf{x}' = M\mathbf{x}$, where M is the matrix.

One very nice feature of the matrix representation is that we can use it to factor a complex transform into a set of simpler transforms. For example, suppose we want to scale an object up to a new size, shear the object to a new shape, and finally rotate the object. Let S be the scale matrix, H be the shear matrix and R be the rotation matrix. Then

$$\mathbf{x}' = R(H(S\mathbf{x}))$$

defines a sequence of three transforms: 1st-scale, 2nd-shear, 3rd-rotate. Because matrix multiplication is associative, we can remove the parentheses and multiply the three matrices together, giving a new matrix $M = RHS$. Now we can rewrite our transform

$$\mathbf{x}' = (RHS)\mathbf{x} = M\mathbf{x}$$

If we have to transform thousands of points on a complex model, it is clearly easier to do one matrix multiplication, rather than three, each time we want to transform a point. Thus, matrices are a very powerful way to encapsulate a complex transform and to store it in a compact and convenient form.

In matrix form, we can catalog the linear transforms as

$$\text{Scale: } \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}, \text{ Rotate: } \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}, \text{ Shear: } \begin{bmatrix} 1 & h_x \\ h_y & 1 \end{bmatrix},$$

where s_x and s_y scale the x and y coordinates of a point, θ is an angle of counterclockwise rotation around the origin, h_x is a horizontal shear factor, and h_y is a vertical shear factor.

C.4 HOMOGENEOUS COORDINATES

Since the matrix form is so handy for building up complex transforms from simpler ones, it would be very useful to be able to represent all of the affine transforms by matrices. The problem is that translation is not a linear transform. The way out of this dilemma is to turn the 2D problem into a 3D problem, but in *homogeneous coordinates*.

We first take all of our points $\mathbf{x} = (x, y)$, express them as 2D vectors $\begin{bmatrix} x \\ y \end{bmatrix}$ and make these

into 3D vectors with identical (thus the term homogeneous) 3rd coordinates set to 1:

$$\begin{bmatrix} x \\ y \end{bmatrix} \Longrightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.$$

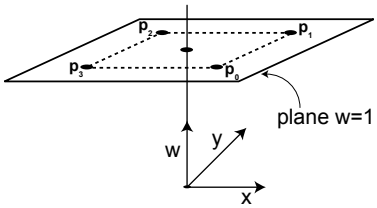
By convention, we call this third coordinate the w coordinate, to distinguish it from the usual 3D z coordinate. We also extend our 2D matrices to 3D homogeneous form by appending an extra row and column, giving

$$\text{Scale:} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}, \text{Rotate:} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}, \text{Shear:} \begin{bmatrix} 1 & h_x & 0 \\ h_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Note what happens when we multiply our 3D homogeneous matrices by 3D homogeneous vectors:

$$\begin{bmatrix} a & b & 0 \\ d & e & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax + by \\ dx + ey \\ 1 \end{bmatrix}.$$

This is the same result as in 2D, with the exception of the extra w coordinate, which remains 1. All we have really done is to place all of our 2D points on the plane $w = 1$ in 3D space, and now we do all the operations on this plane. Really, the operations are still 2D operations.



But, the magic happens when we place the translation parameters c and f in the matrix in the 3rd column:

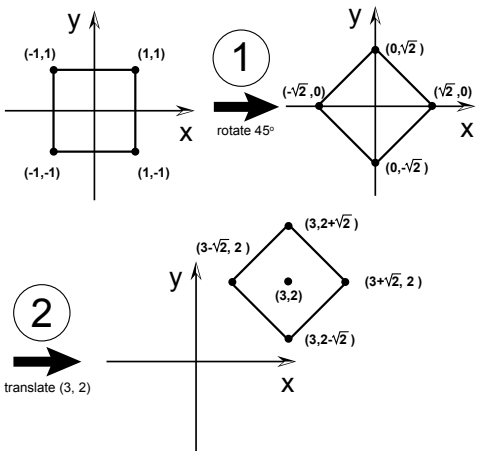
$$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax + by + c \\ dx + ey + f \\ 1 \end{bmatrix}$$

We can now do translations as linear operations in homogeneous coordinates! So, we can add a final matrix to our catalog:

$$\text{Translate:} \begin{bmatrix} 1 & 0 & \triangle x \\ 0 & 1 & \triangle y \\ 0 & 0 & 1 \end{bmatrix},$$

where $\triangle x$ is the translation in the x direction and $\triangle y$ is the translation in the y direction. The astute reader will see the trick behind the magic – 2D translation is now being expressed as a shear in 3D space.

Now, suppose we have a 2×2 square centered at the origin and we want to first rotate the square by $45°$ about its center and then move the square so its center is at $(3, 2)$. We can do this in two steps, as shown in the diagram to the right.



In matrix form:

$$M = T_{(3,2)}R_{45°} = \begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos 45° & -\sin 45° & 0 \\ \sin 45° & \cos 45° & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos 45° & -\sin 45° & 3 \\ \sin 45° & \cos 45° & 2 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \sqrt{2}/2 & -\sqrt{2}/2 & 3 \\ \sqrt{2}/2 & \sqrt{2}/2 & 2 \\ 0 & 0 & 1 \end{bmatrix}.$$

Note that

$$M \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 + \sqrt{2} \\ 1 \end{bmatrix}, \text{and } M \begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 - \sqrt{2} \\ 2 \\ 1 \end{bmatrix},$$

verifying that we get the same result shown in the figure.

C.5 3D FORM OF THE AFFINE TRANSFORMATIONS

Now, we can extend all of these ideas to 3D in the following way:

1. Convert all 3D points to homogeneous coordinates

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \Longrightarrow \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

The extra (4th) coordinate is again called the w coordinate.

2. Use matrices to represent the 3D affine transforms in homogeneous form.

The following matrices constitute the basic affine transforms in 3D, expressed in homogeneous form:

$$\text{Translate:}\begin{bmatrix}1 & 0 & 0 & \triangle x\\ 0 & 1 & 0 & \triangle y\\ 0 & 0 & 1 & \triangle z\\ 0 & 0 & 0 & 1\end{bmatrix},\ \text{Scale:}\begin{bmatrix}s_x & 0 & 0 & 0\\ 0 & s_y & 0 & 0\\ 0 & 0 & s_z & 0\\ 0 & 0 & 0 & 1\end{bmatrix},$$

and

$$\text{Shear:}\begin{bmatrix}1 & h_{xy} & h_{xz} & 0\\ h_{yx} & 1 & h_{yz} & 0\\ h_{zx} & h_{zy} & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}.$$

In addition, there are three basic rotations in 3D,

$$\text{Rotation about the } x \text{ axis:}\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & \cos\theta_x & -\sin\theta_x & 0\\ 0 & \sin\theta_x & \cos\theta_x & 0\\ 0 & 0 & 0 & 1\end{bmatrix},$$

$$\text{Rotation about the } y \text{ axis:}\begin{bmatrix}\cos\theta_y & 0 & \sin\theta_y & 0\\ 0 & 1 & 0 & 0\\ -\sin\theta_y & 0 & \cos\theta_y & 0\\ 0 & 0 & 0 & 1\end{bmatrix},$$

and

$$\text{Rotation about the } z \text{ axis:}\begin{bmatrix}\cos\theta_z & -\sin\theta_z & 0 & 0\\ \sin\theta_z & \cos\theta_z & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}.$$

The rotations, specified in this way, determine an amount of rotation about each of the individual axes of the coordinate system. The angles θ_x, θ_y, and θ_z of rotation about the three axes are called the Euler angles. They can be used to describe an off-axis rotation, by combining Euler angle rotations via matrix multiplication. Note, however, that the order of rotation affects the end result, so besides specifying Euler angles, an order of rotation must be specified. In general, affine transformations are associative but are not commutative, so the order in which operations are done is highly important. One can see this for rotations by computing the product $R_{\theta_x}R_{\theta_y}R_{\theta_z}$, and comparing with the result obtained by the product $R_{\theta_z}R_{\theta_y}R_{\theta_x}$. Please see Appendix D for a more powerful and general look at rotation.

APPENDIX D

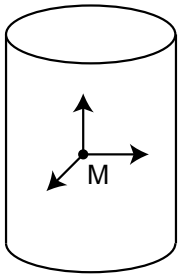
Coordinate Systems

CONTENTS

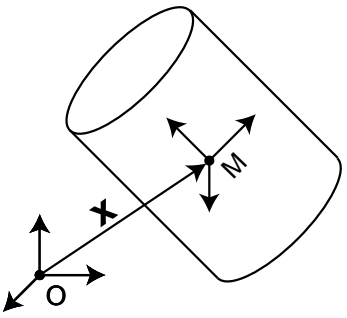
The idea of a *coordinate system*, or *coordinate frame* is pervasive in computer graphics. For example, it is usual to build a model in its own *modeling frame*, and later place this model into a scene in the *world coordinate frame*. We often refer to the modeling frame as the *object frame*, and the 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.



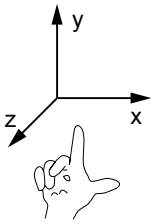
Cylinder in model frame M



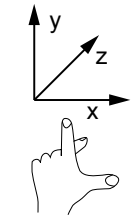
Cylinder in world frame O. The cylinder's frame has been rotated and then translated by **x**

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.



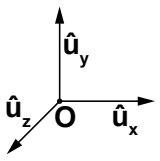
Right Hand System



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

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

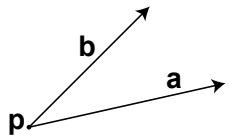
A transformation matrix that first rotates and then translates will then work perfectly for transforming both vectors and points in one frame to another frame.

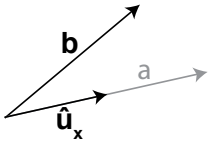
A final note is that to transform geometry from one frame to another frame we have a

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.



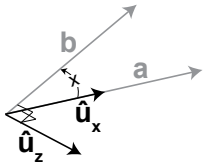


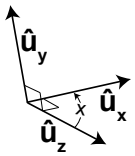
We can arbitrarily pick the x axis direction to be aligned with **a**, so

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

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

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



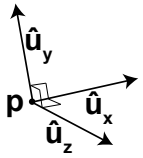


The third, or y axis must be perpendicular to both $\hat{\mathbf{u}}_x$ and $\hat{\mathbf{u}}_z$, so

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

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 θ. 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

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