GEOMETRIC MODELING: A First Course

2. Motions and Projections

2.1 Points and Vectors

Imagine a small solid object and let its dimensions decrease indefinitely. The result of this conceptual experiment is modeled by a mathematical abstraction called a *point.* Modern mathematics defines rigorously *Euclidean spaces* as sets whose elements, called *points,* satisfy certain axioms. In everyday language we talk of "being at a point in space", and in geometric modeling we use Euclidean points to define mathematically the locations of objects. In addition, sets of points serve to model more complicated objects, from trajectories to physical solids.

Consider now a solid object in straight-line motion. The object's velocity has a direction and a magnitude, or speed, measured *e.g.* in meters per second. Velocities and other physical entities such as forces that can be characterized by a direction and a magnitude are modeled mathematically by *vectors.* Vectors may be added by using the familiar parallelogram rule of analytical geometry and elementary mechanics. They may also be multiplied by scalar numbers. Scalar multiplication changes the magnitude of the vector but not its direction. In modern mathematics a *vector space* is a set of elements, called vectors, with two operations defined on them—vector addition and multiplication by a scalar—that have certain algebraic properties defined axiomatically. The vector-space axioms ensure that the usual Cartesian vectors of analytic geometry are a special case of abstract vectors. Interestingly, there are many other useful entities that are abstract vectors as well. Examples include polynomials of degree $\leq n$, the spline functions we will discuss later in this course, periodic functions with period T, continuous functions in a closed interval $[a,b]$, and so on. The theory of vector spaces applies equally well to all of these entities. This is a good example of the power and elegance of abstraction in modern mathematics.

Points and vectors are intimately connected. In principle there are no privileged points or directions in space, i.e., space is homogeneous and isotropic. But let us pick some arbitrary point $\mathbf{o}$ and call it the *origin*. (Typically, the origin is selected for convenience in solving a specific problem.) Now each point $\mathbf{p} \neq \mathbf{o}$ plus $\mathbf{o}$ define a direction and a length. Therefore, for a fixed origin $\mathbf{o}$, each point $\mathbf{p}$ corresponds to a vector $\boldsymbol{x}$, and conversely. That is, there is a one-to-one correspondence between points and vectors. We are arguing intuitively, but the argument can be rigorized by defining *point difference* as an operation that produces a vector from two point arguments such that $\mathbf{p} - \mathbf{o} = \boldsymbol{x}$. We use the notation

$$\mathbf{p} \overset{\mathbf{o}}{\leftrightarrow} \boldsymbol{x}$$

to signify that point $\mathbf{p}$ corresponds to vector $\boldsymbol{x}$ in the fixed origin $\mathbf{o}$. (We are using lower-case boldface for points and italic boldface for vectors.)

The correspondence between points and vectors depends on the choice of origin, as shown in Figure 2.1.1. Here point **p** corresponds to vector $\boldsymbol{x}$ when the origin is **q**. However, **p** corresponds to $\boldsymbol{y}$ when the origin is **o**. There is a simple relationship between $\boldsymbol{x}$ and $\boldsymbol{y}$. Specifically, $\boldsymbol{y} = \boldsymbol{x} + \delta$, where $\delta = \mathbf{q} - \mathbf{o}$ is the difference of the two origins. Therefore the effects of a change of origin are easy to evaluate.

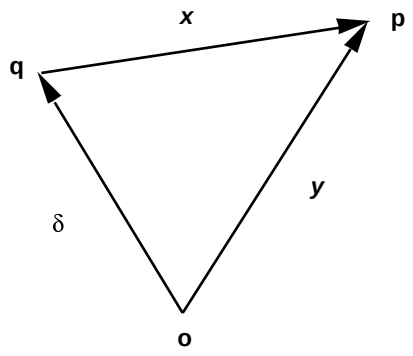

Figure 2.1.1 Change of origin

In the geometric modeling field it is customary not to distinguish between points and vectors, because a fixed "lab" or "master" origin is assumed to exist. By abuse of language one talks of operations such as point addition, although addition is defined only for vectors, and extending the vector operation to points produces results that depend on the choice of origin.

The vector-space elements introduced above are called in this course simply *vectors* or *ordinary vectors*, to distinguish them from free and applied vectors, which will be discussed further on.

In a vector space it is always possible to select a minimal set of vectors $\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_n$ such that any vector of the space can be expressed as a linear combination

$$\boldsymbol{x} = x_1\boldsymbol{e}_1 + x_2\boldsymbol{e}_2 + \ldots + x_n\boldsymbol{e}_n.$$

The $\boldsymbol{e}_i$ are called a *basis*, and the x_i are the *components* of the vector $\boldsymbol{x}$ in the given basis. There are infinitely many bases in a vector space, but they all have the same number n of vectors, and n is called the *dimension* of the space. In this course we are primarily interested in dimensions 2 and 3. The components of a vector in a fixed basis are unique, and, conversely, a set of components determines a unique vector.

Therefore, for a fixed basis, there is a one-to-one correspondence between vectors and arrays of components. We denote this correspondence as

$$\boldsymbol{x} \overset{E}{\longleftrightarrow} X.$$

It is convenient to arrange the components of a vector in a column matrix, and the vectors of a basis in a row matrix:

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad E = \begin{bmatrix} \boldsymbol{e}_1 & \boldsymbol{e}_2 & \dots & \boldsymbol{e}_n \end{bmatrix}.$$

Using matrix notation a vector can be written as $\boldsymbol{x} = EX$. The correspondence between vectors and matrices preserves addition and multiplication by a scalar. The matrix Z that corresponds to the sum of two vectors $\boldsymbol{z} = \boldsymbol{x} + \boldsymbol{y}$ is the sum

$$Z = X + Y = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} x_1 + y_1 \\ x_2 + y_2 \\ \vdots \\ x_n + y_n \end{bmatrix}.$$

For multiplication by a scalar, if $\boldsymbol{z} = a\,\boldsymbol{x}$, then $Z = a\,X$, or

$$Z = \begin{bmatrix} ax_1 \\ ax_2 \\ \vdots \\ ax_n \end{bmatrix}.$$

The *inner* or *dot product*, denoted $\boldsymbol{x} \cdot \boldsymbol{y}$, is another useful operation defined on vectors. It produces a scalar given two vector arguments. It is defined formally by a set of axioms. The square root of the inner product of a vector with itself is the *norm* or *length* of the vector, denoted

$$|\boldsymbol{x}| = \sqrt{\boldsymbol{x}.\boldsymbol{x}}.$$

A *unit vector* is a vector whose length equals unity. Two vectors are *orthogonal* if their dot product is zero. The cosine of the angle between two vectors is given by

$$\cos\theta = \frac{\boldsymbol{x}.\boldsymbol{y}}{\|\boldsymbol{x}\|.\|\boldsymbol{y}\|}.$$

The most convenient bases are the *orthonormal* bases, composed of unit vectors that are pairwise orthogonal. In an orthonormal basis the inner product of two vectors is

$$\boldsymbol{x}.\boldsymbol{y} = X^t Y = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n,$$

where the superscript denotes matrix transposition, obtained by interchanging rows with columns. This is the familiar formula from analytic geometry. (Note that this formula is not valid in non-orthonormal bases.) The length in an orthonormal basis becomes

$$|\boldsymbol{x}| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2},$$

which is also a well-known formula. In a Euclidean space we define the distance between two points **p** and **q** as the norm of the vector **p** − **q**.

Because points correspond to vectors, for a fixed origin, and vectors correspond to column matrices, for a fixed basis, there is also a one-to-one correspondence between points and column matrices. A pair (origin, basis) is called a *frame* or *coordinate system.* For a fixed frame, points correspond to column matrices:

$$\mathbf{p} \xleftrightarrow{(E,\mathbf{o})} X .$$

The elements of the matrix associated with a point in a given frame are called the *coordinates* of the point in that frame. The correspondences between points, vectors and column matrices are very important, because they provide us with the computational tools we need to represent and manipulate these entities. Matrices are easy to represent as arrays in any modern programming language, and operations such as vector addition also are easy to implement. Point and vector properties are computed by using their coordinates or components in a convenient frame or basis. For example, the distance between two points **p** and **q** with coordinate matrices X and Y is evaluated by the familiar expression

$$\mathrm{d}(\mathbf{p},\mathbf{q}) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2} .$$

Finally, there is an additional operation on vectors, called the *vector product* (also known as *cross,* or *exterior* product), that is very useful, especially in 3-D. Here we define it in terms of components in a right-handed, orthonormal, 3-D basis:

$$\boldsymbol{x} \times \boldsymbol{y} = (x_2 y_3 - x_3 y_2)\boldsymbol{e}_1 + (x_3 y_1 - x_1 y_3)\boldsymbol{e}_2 + (x_1 y_2 - x_2 y_1)\boldsymbol{e}_3 .$$

The result of a cross product is not truly a vector, and its definition depends on the orientation or handedness of a basis. Right-handed orthonormal bases in 2 and 3-D are shown in Figure 2.1.2. In this course we always use right-handed bases, and we can ignore the subtleties of vector-product definitions .

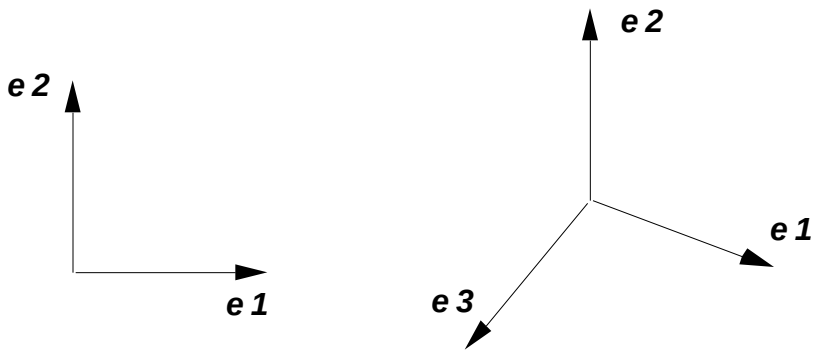

Figure 2.1.2 – Right-handed bases

The cross product of two parallel vectors is zero. For two non-parallel vectors, $\boldsymbol{x}$ and $\boldsymbol{y}$, the cross-product $\boldsymbol{x} \times \boldsymbol{y}$ is perpendicular to both $\boldsymbol{x}$ and $\boldsymbol{y}$. In particular, if E is a right-handed orthonormal basis in 3-D, then

$$\boldsymbol{e}_1 \times \boldsymbol{e}_2 = \boldsymbol{e}_3$$
$$\boldsymbol{e}_2 \times \boldsymbol{e}_3 = \boldsymbol{e}_1 .$$
$$\boldsymbol{e}_3 \times \boldsymbol{e}_1 = \boldsymbol{e}_2$$

These equations are convenient for completing a basis when two of its vectors are known.

2.2 Transformations

Moving, sizing, and deforming objects are fundamental operations in geometric modeling. Since objects are sets of points, what we need are *transformations* that map points onto other points. The following subsections discuss linear and affine transformations, which are the simplest and most commonly used in geometric modeling. For simplicity we assume a fixed origin, and make no distinction between points and vectors.

2.2.1 Linear Transformations

A transformation $\boldsymbol{T}$ from a vector space onto itself is linear if it distributes over linear combinations, i.e.,

$$\boldsymbol{T}(a\boldsymbol{x} + b\boldsymbol{y}) = a\boldsymbol{T}(\boldsymbol{x}) + b\boldsymbol{T}(\boldsymbol{y}).$$

Suppose we have two bases

$$E = \begin{bmatrix} \boldsymbol{e}_1 & \cdots & \boldsymbol{e}_n \end{bmatrix}$$
$$F = \begin{bmatrix} \boldsymbol{f}_1 & \cdots & \boldsymbol{f}_n \end{bmatrix}$$

and we want a linear transformation that maps the vectors of E onto the vectors of F (see Figure 2.2.1.1):

$$\boldsymbol{T}_{ef}(\boldsymbol{e}_i) = \boldsymbol{f}_i, \quad i = 1,\ldots,n.$$

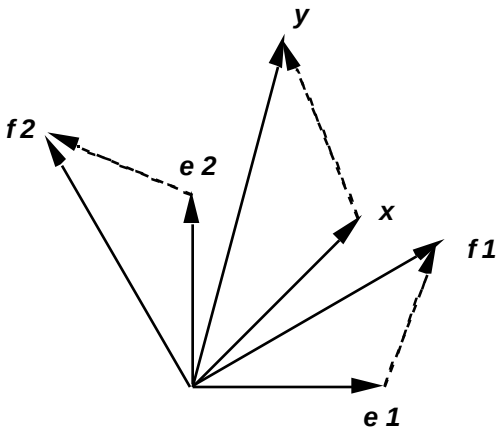

Figure 2.2.1.1 - Transforming a basis E and a vector $\boldsymbol{x}$.

What is the effect of such a transformation on an arbitrary vector? Let $\boldsymbol{x}$ be a vector and $\boldsymbol{y}$ its transformed version

$$\boldsymbol{y} = \boldsymbol{T}_{ef}(\boldsymbol{x}).$$

Write the vector $\boldsymbol{x}$ in terms of its components in basis E

$$\boldsymbol{x} = EX^e$$

and use linearity to obtain

$$\boldsymbol{y} = \boldsymbol{T}_{ef}(\boldsymbol{x}) = \left[\boldsymbol{T}_{ef}(\boldsymbol{e}_1) \quad \cdots \quad \boldsymbol{T}_{ef}(\boldsymbol{e}_n)\right] X^e = \left[\boldsymbol{f}_1 \quad \cdots \quad \boldsymbol{f}_n\right] X^e .$$

Now replace the basis vectors F by their components in basis E

$$\boldsymbol{y} = \left[EF_1^e \quad \cdots \quad EF_n^e\right] X^e = E\left[F_1^e \quad \cdots \quad F_n^e\right] X^e .$$

This shows that the components of $\boldsymbol{y}$ in basis E are

$$\boxed{Y^e = M^e X^e}$$

where M^e is an n by n matrix whose columns are the components of the vectors of F in the basis E:

$$\boxed{M^e = \left[F_1^e \quad \cdots \quad F_n^e\right]}$$

The last two equations are important for several reasons.

1. They show that, for a fixed basis E, each transformation corresponds to a square matrix. Earlier on we had a correspondence between vectors and column matrices, and now we have a correspondence between linear transformations and square matrices.
2. They give us computational tools for evaluating the effect of a transformation on a vector. We simply multiply the square matrix that corresponds to the transformation by the column matrix that corresponds to the vector. This is easily done via array operations in any modern programming language.
3. They tell us how to construct a matrix that maps the vectors of a basis onto another.

Let us illustrate this matrix construction procedure. Many geometric modeling systems attach coordinate frames to objects, and provide facilities for placing objects by aligning frames. Consider the rectangle shown on the left in Figure 2.2.1.2, and suppose we want to orient it such that it aligns with the rectangle on the right. All we need is a transformation with a matrix whose columns are the components of the vectors F in the basis E. (In 2-D it is easy to determine the required transformation by other means, but in a 3-D example it would not be as easy.)

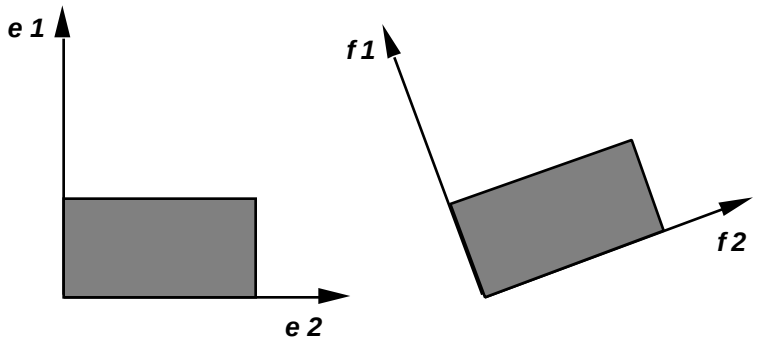

Figure 2.2.1.2 - Orienting an object by frame alignment

For a specific example, let us determine the matrix that corresponds to a counterclockwise rotation by an angle θ. The components of the vectors F are easy to calculate from elementary trigonometry, as shown in Figure 2.2.1.3.

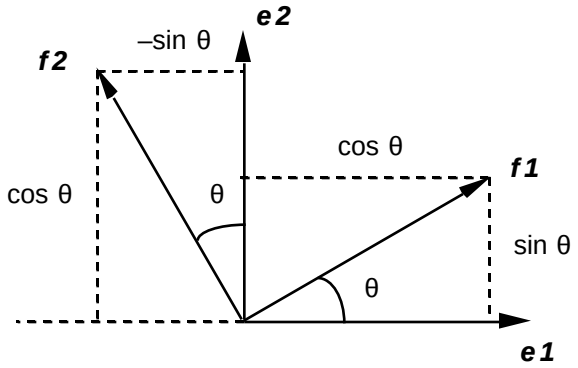

Figure 2.2.1.3 - Derivation of a rotation matrix

We obtain

$$F_1^e = \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix}, \quad F_2^e = \begin{bmatrix} -\sin\theta \\ \cos\theta \end{bmatrix},$$

and therefore the rotation matrix is

$$M^e = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}.$$

Composition of successive transformations corresponds to matrix multiplication

$$\boldsymbol{T}_2(\boldsymbol{T}_1(\boldsymbol{x})) \overset{E}{\leftrightarrow} M_2^e M_1^e X^e .$$

And the inverse transformation, which maps a basis F onto a basis E corresponds to the inverse matrix

$$M_{fe}^e = M_{ef}^{e\,-1}.$$

Matrix multiplication is not commutative, i.e., in general $AB \neq BA$ for arbitrary square matrices A and B. The inverse of a matrix product reverses the order of the matrices:

$$(AB)^{-1} = B^{-1}A^{-1}$$

Note that some linear transformations do not map a basis onto another. We will see examples of these later.

2.2.2 Specific Linear Transformations

Here we investigate several interesting linear transformations in 2-D. The results apply also to 3-D, with minor and obvious modifications.

Scaling – Consider the transformation with matrix

$$\begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}.$$

To study its effect on a vector we multiply the corresponding matrices

$$\begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax \\ by \end{bmatrix}.$$

Here we are denoting the components of a 2-D vector by the customary x and y. The result is a scaling by factors a and b along the x and y axes. If $a = b$ the scaling is uniform or isotropic and alters the size of an object but not its shape. If both scale factors equal unity, the transformation is the identity and does not modify the object.

Figure 2.2.2.1 illustrates anisotropic scaling by its effect on a square located at the origin.

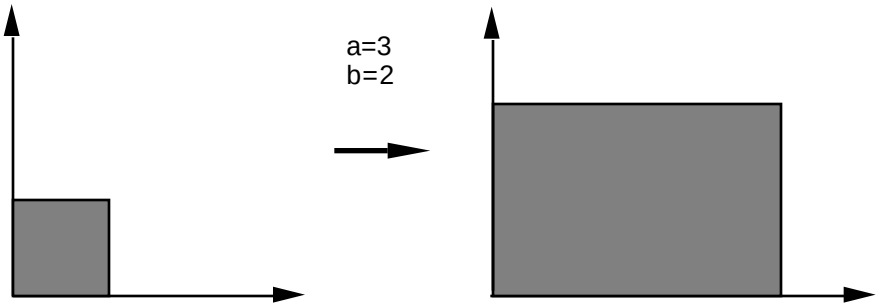

Figure 2.2.2.1 – Non-uniform scaling

Shear – Now let one of the off-diagonal elements of the matrix be non-zero. The result is a shear, with the following matrix, and with the effect shown in Figure 2.2.2.2.

$$\begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x + ay \\ y \end{bmatrix}.$$

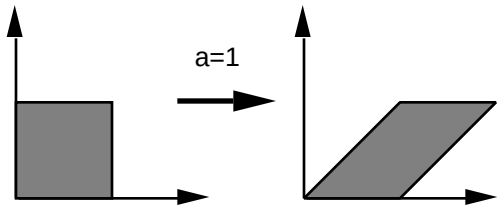

Figure 2.2.2.2 – Shear

Rotation – As we saw earlier, the matrix is

$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}.$$

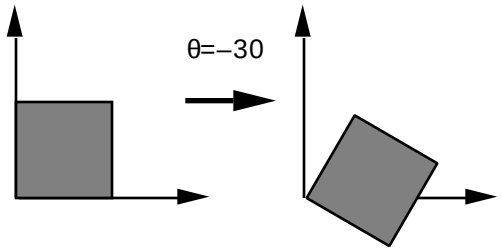

Figure 2.2.2.3 – Rotation

Reflection – Scalings with negative factors produce reflections. A reflection about the x axis is shown below.

$$\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -x \\ y \end{bmatrix}.$$

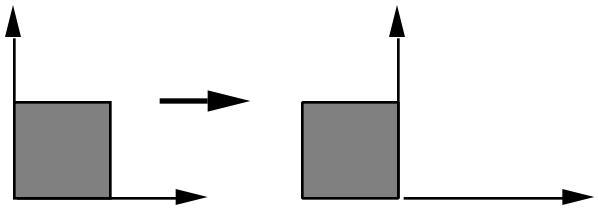

Figure 2.2.2.4 – Reflection about the vertical axis

Reflections about the horizontal axis, or about the origin can be constructed similarly.

Orthographic projection – Consider now

$$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x \\ 0 \end{bmatrix}.$$

This transformation zeroes the *y* component and does not affect the *x* component. It corresponds to a perpendicular or orthographic projection on the *x* axis.

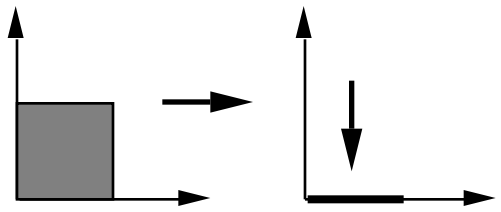

Figure 2.2.2.5 – Orthographic projection on the horizontal axis

Orthographic projection does not map a basis onto another basis. It is called a *singular* transformation, and cannot be inverted. The projection causes a loss of information about the *y* components of the vectors. Knowledge of the *x* component is insufficient to recover a vector, because many vectors project on the same point of the *x* axis.

2.2.3 Rigid Motions

A translation $\Delta$is a mapping that associates to each vector $\boldsymbol{x}$ the sum $\boldsymbol{x} + \delta$, where δ is a constant vector. Translations are not linear transformations and cannot be computed by matrix multiplication as we have been doing (but see Section 2.6.1 below). The components of a translated vector $\boldsymbol{y} = \boldsymbol{x} + \delta$ are

$$Y = X + D \ ,$$

where D is the column matrix that corresponds to the translation vector δ. A translation is shown in Figure 2.2.3.1.

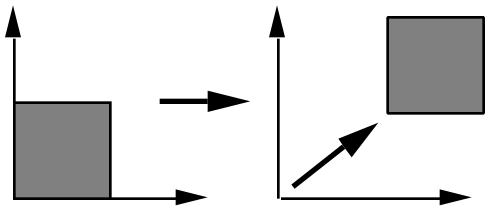

Figure 2.2.3.1 - Translation

Compositions of translations and linear transformations are called *affine transformations*. Both translations and linear transformations are practically important, and their non-uniform behavior with respect to components is computationally inconvenient. Separate procedures must be written for dealing with translations and linear transformations, and they cannot be composed by matrix multiplication. (We will see later that both transformations can be treated uniformly if we introduce homogeneous coordinates.)

Typically, in geometric modeling we do not want to change the shape of a transformed object. Transformations are applied primarily to locate and orient objects. Transformations that preserve distance are called *isometries* (from the Greek, meaning "same measure"). Isometries that also preserve the signed angles between vectors are called in this course *rigid motions*. (This is not entirely standard terminology; some texts consider "rigid motions" and "isometries" as synonyms.) It can be shown that rigid motions are affine and must be *compositions of translations and rotations*.

The matrix that corresponds to a rotation in an orthonormal basis is a special case of a so-called *orthogonal matrix*. These matrices can be inverted easily, by transposition:

$$M_{orth}^{-1} = M_{orth}^{t} \ .$$

General matrix inversion requires numerical procedures, which tend to be unstable when the matrix is almost singular, and always introduce numerical errors. But inversion of rotation matrices can be done swiftly and without round-off, by swapping rows with columns.

Sets that are related by a rigid motion are called *congruent*. In the geometric modeling jargon we often refer to an entire class of sets congruent to one another as a *rigid object*, and call each individual set an *instance* of the rigid object. The location and orientation of an instance, collectively called its *pose*, may be defined by a rigid motion that takes the set from an initial, standard pose to its final pose. Figure 2.2.3.2 shows several congruent triangles in the plane.

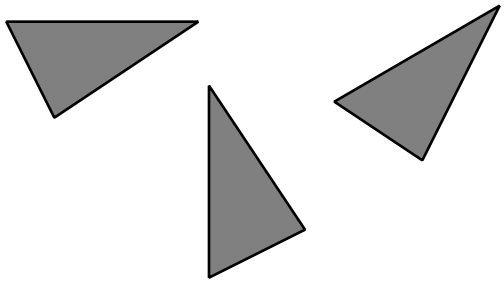

Figure 2.2.3.2 – Instances of a triangle

The notion of congruence is fundamental in Euclidean geometry. Euclid's original formulation defined congruence informally: two figures were said to be congruent if they could be "superposed". The rigorous definition in terms of rigid motions is only about one century old. In the spirit of Felix Klein's famous "Erlangen program", Euclidean geometry may be viewed as the study of those properties of geometric objects that are invariant under rigid motions, and two objects are considered equivalent if they are related by a rigid motion, i.e., if they are congruent. Rigid motion invariants (also called Euclidean properties) are such things as distances, angles, perpendicularity, and so on, which are the main subjects of study in high school geometry. We will see later that there are other kinds of geometries, each with its fundamental transformations, analogous to the rigid motions of Euclidean geometry, and with its notion of geometrical equivalence, analogous to congruence.

2.3 Free and Applied Vectors

We defined translation of a vector $\boldsymbol{x}$ as the addition to $\boldsymbol{x}$ of a vector δ, as shown in Figure 2.3.1. Vector translation does not correspond to the intuitive notion of translation of an "arrow" by translating its endpoints, without changing the length or the direction of the arrow. An alternative notion of vector that behaves more like an arrow also is useful in geometric modeling, as the following example illustrates.

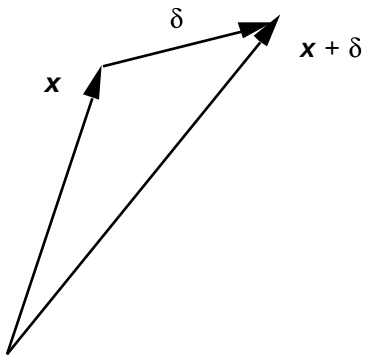

Figure 2.3.1 – Translation by vector addition

Consider the right, circular cylinder shown on the left in Figure 2.3.2. The cylinder is characterized completely by two scalar parameters—its diameter D and height H—plus a point $\mathbf{c}$—the center of a base—and a vector $\boldsymbol{a}$ along the cylinder's axis. Suppose now that we want to move the cylinder to a different location and orientation, shown on the right in the figure. Mathematically, moving the cylinder corresponds to applying a rigid motion $\boldsymbol{T}$ to it. How can we compute the values $\mathbf{c}'$ and $\boldsymbol{a}'$ that characterize the cylinder after the application of $\boldsymbol{T}$? Clearly $\mathbf{c}' = \boldsymbol{T}(\mathbf{c})$. But $\boldsymbol{a}' \neq \boldsymbol{T}(\boldsymbol{a})$ because $\boldsymbol{T}$ has a translational component.

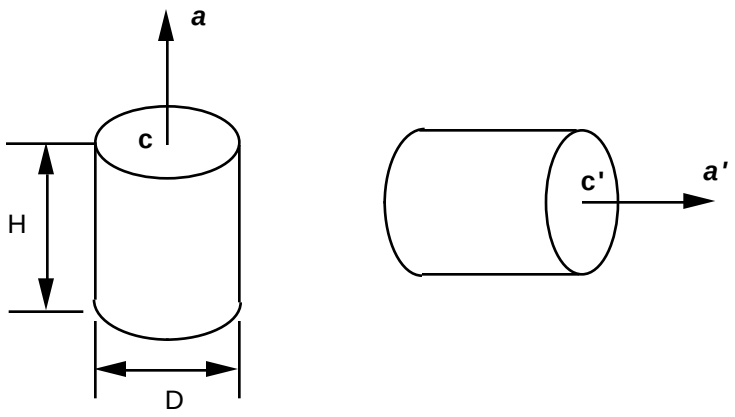

Figure 2.3.2 – Moving a cylinder

This example, and many similar situations, can be handled conveniently if we introduce a new entity, called a *free vector*, that is not affected by translations. Free vectors and the ordinary vectors defined earlier differ only in their behavior with respect to translations.

The cylinder in our example can be described by scalars D and H, point **c**, and free vector $\boldsymbol{a}$. Intuitively, it is helpful to think of $\boldsymbol{a}$ as being attached to the point **c**. This notion may be formalized by defining yet another entity, called an *applied vector*, which consists of a pair (**p**, $\boldsymbol{x}$), where **p** is a point and $\boldsymbol{x}$ a free vector. Equivalently, we can define an applied vector as a pair of endpoints (**p**, **q**) with **q** = **p** + $\boldsymbol{x}$. An applied vector is transformed by applying a transformation to both endpoints. In Figure 2.3.2 the pair (**c**, $\boldsymbol{a}$) is an applied vector, which transforms as shown on the right in the figure.

Free and applied vectors are used extensively in geometric modeling. For example, the normal direction to a surface is often represented by a free vector plus the point at which the normal is calculated, i.e., by an applied vector. (Point information is unnecessary for planar surfaces, which have a single, constant normal.) Tangential directions for curves are treated similarly.

2.4 Change of Basis

Let $\boldsymbol{x}$ be a vector with components X^e in basis E. Consider a new basis F, obtained from E by a transformation $\boldsymbol{T}_{ef}$, with a corresponding matrix M^e_{ef} in basis E. Each vector of the new basis can be written in terms of its components as

$$\boldsymbol{f}_i = EF_i^e$$

and we can summarize these n equations by the matrix equality

$$F = \begin{bmatrix} \boldsymbol{f}_1 & \cdots & \boldsymbol{f}_n \end{bmatrix} = \begin{bmatrix} EF_1^e & \cdots & EF_n^e \end{bmatrix} = EM^e_{ef}.$$

What are the components of $\boldsymbol{x}$ in the new basis? Since

$$\boldsymbol{x} = EX^e = FX^f = EM^e_{ef}X^f$$

we conclude that

$$\boxed{X^e = M^e_{ef}X^f}$$

Recall that when we apply a transformation $\boldsymbol{T}_{ef}$ to a vector $\boldsymbol{x}$, the vector is transformed into a second vector $\boldsymbol{y}$, and the column matrices of the two vectors are related by

$$\boxed{Y^e = M^e_{ef}X^e}$$

The last two equations are very similar but correspond to different procedures. The first governs the change of components of a fixed vector when a basis changes. The second gives the components of a transformed vector in a fixed basis. Note that the same matrix is involved in the two equations, but there is a "reversal of direction" between the two matrix actions. The change of basis equation can also be written in terms of the inverse matrix

$$X^f = M^{e\,-1}_{ef}X^e.$$

The need for inversion has a simple geometric interpretation, illustrated by the example of Figure 2.4.1. Consider a vector $\boldsymbol{x}$ in base E, on the left in the figure. If we rotate the basis by an angle θ to obtain basis F, as shown in the center of the figure, the components of $\boldsymbol{x}$ change. Now let us apply in basis E the inverse transformation to $\boldsymbol{x}$, rotating it by the angle $-\theta$, so as to obtain $\boldsymbol{y}$, as shown on the right. The components of $\boldsymbol{y}$ in basis E are the same as the components of the original $\boldsymbol{x}$ in basis F.

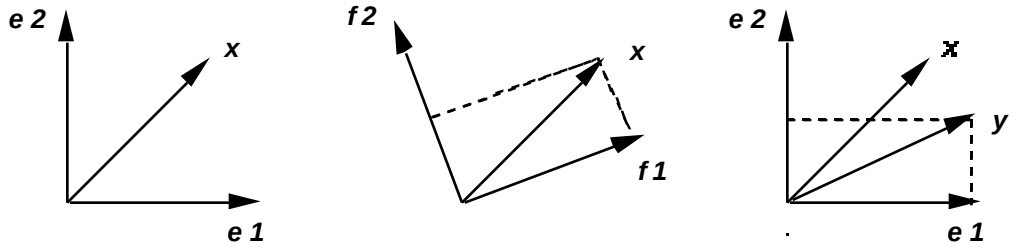

Figure 2.4.1 – Vector transformation versus change of basis

2.5 Homogeneous Coordinates

We begin this section with a pragmatic view of homogeneous coordinate methods. We then explain them geometrically, and finally show how they can be used to compute perspective projections.

2.5.1Transformations in Homogeneous Coordinates

Translations and linear transformations can be treated more uniformly if we introduce a different system of coordinates, called *homogeneous coordinates*. For simplicity we work in 2-D, but generalizations to 3-D or n-D are straightforward. We continue to make no distinction between points and ordinary vectors. Suppose that we have a vector $\boldsymbol{x}$ with components X, and want to apply to it a linear transformation with matrix M, so as to obtain another vector $\boldsymbol{y}$ with components Y. We introduce an additional component and associate with the vector $\boldsymbol{x}$ the column matrix

$$X^* = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} X \\ 1 \end{bmatrix}.$$

The elements of X^* are called *homogeneous coordinates*. We also add a third row and column to the linear transformation matrices as follows

$$M^* = \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

This matrix can be written in block format as

$$M^* = \begin{bmatrix} M & 0 \\ 0 & 1 \end{bmatrix},$$

where M is the usual 2 by 2 linear transformation matrix, and the two-zero row and column are both denoted by 0. Multiplying the matrices

$$M^* X^* = \begin{bmatrix} M & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ 1 \end{bmatrix} = \begin{bmatrix} MX \\ 1 \end{bmatrix} = \begin{bmatrix} Y \\ 1 \end{bmatrix} = Y^*$$

shows how to evaluate the effects of a linear transformation in the new, augmented-matrix format. Scalings, shears, rotations, and so on, can be achieved by replacing M in the 3 by 3 matrix above by the various matrices we discussed earlier.

Until now we have gained nothing with this approach, and we have lost some computational efficiency because 3 by 3 matrices require more storage and more computation than their 2-D counterparts. But let us now investigate what happens if the elements of the third column of the matrix become non-zero. Consider

$$M^* = \begin{bmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{bmatrix},$$

and apply it to a generic vector:

$$Y^* = M^* X^* = \begin{bmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + a \\ y + b \\ 1 \end{bmatrix}.$$

This is precisely the result of translating $\boldsymbol{x}$ by a vector with components (a, b). Therefore we have found a method for computing both translations and linear transformations by matrix multiplication. In particular, rigid motions in the plane are associated with 3 by 3 matrices. In 3-D they correspond to 4 by 4 matrices. For reference, the three matrices that correspond to rotations about the x, y and z axes are:

$$x: \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad y: \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad z: \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Uniform treatment of translations and rotations is computationally important. It implies that we only need one procedure to implement both, and that matrix-multiplication hardware can be used for both. We will see later that homogeneous coordinates also can deal with projections, which are needed for displaying objects.

2.5.2 Geometric Interpretation

Homogeneous-coordinate methods were introduced above as convenient "recipes". But they have a rich body of mathematics and geometric intuition underlying them. Here we explore it briefly. First we generalize slightly, and write the homogeneous coordinates of an Euclidean point **p** as

$$P^* = \begin{bmatrix} x \\ y \\ w \end{bmatrix}.$$

We have increased the dimension of our space by one. In addition, since we identify points at w=1 with Euclidean points, we have placed the standard Euclidean plane at w=1. Figure 2.5.2.1 illustrates this construction.

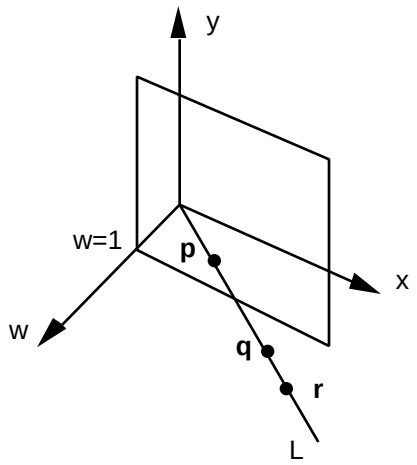

Figure 2.5.2.1 - The Euclidean plane imbedded in an auxiliary 3-space

Now we connect an Euclidean point **p** with the origin, obtaining a straight line L. It is clear that for each **p** there is a corresponding L. Furthermore, since L goes through the origin, it is uniquely determined by any point **q** lying on it. This implies that **p** also is uniquely determined by any **q** of L. The coordinates of any such **q** are called the *homogeneous coordinates* of **p**. If we multiply the three coordinates by the same number k, we obtain another point **r** of the same L. Points **q** and **r** (or any other points of L) are equivalent, in that they all define the same line and also the same Euclidean point **p**. Therefore, the homogeneous coordinates of **p** may be multiplied by any non-zero number without affecting the point. In particular, if $w \neq 1$ and is not zero, we can always scale all the components so as to *normalize* the coordinates:

$$\begin{bmatrix} x/w \\ y/w \\ 1 \end{bmatrix}.$$

The set of all lines through the origin of our auxiliary 3-D space is called the *projective plane*. The elements of the projective plane are called *projective points.* (This terminology

can be confusing since projective points are actually lines.) For each Euclidean point **p** there is a corresponding line L and projective point $\mathbf{p}$. Therefore we can manipulate Euclidean points through operations on their projective counterparts. This is precisely what we are doing when we multiply the homogeneous coordinates of a point by a 3 by 3 matrix. Figure 2.5.2.2 illustrates the procedure. Our goal is to apply an affine transformation **T** to an Euclidean point **p**. We do this in a roundabout fashion. First we imbed the Euclidean point in projective space, obtaining $\mathbf{p}$. Then we apply a *projective transformation* $\mathbf{T}^*$, which corresponds to a matrix M^*, and generate a transformed projective point $\mathbf{q}$. Finally, we normalize, i.e. project the result on w=1, to get the desired **q**.

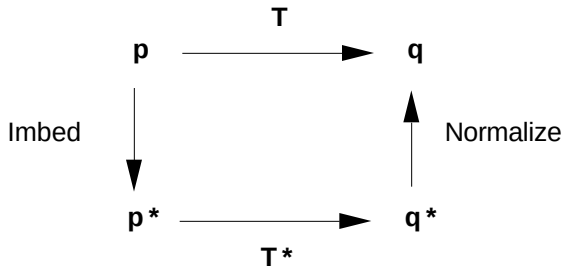

Figure 2.5.2.2

An Euclidean line lying in w=1 corresponds to a plane through the origin of the auxiliary 3-space. Such planes are the *projective lines.* Therefore, each Euclidean line has a corresponding projective line.

The projective plane is "larger" than the Euclidean plane, because there are lines through the origin that do not intersect the w=1 plane. These are the lines that lie in the w=0 plane. That is, the projective plane has an additional projective line, which is the w=0 plane of the auxiliary 3-space. A projective point with homogeneous coordinates (a, b, 0) does not have a corresponding Euclidean point. What is the geometric meaning of such a point?

To answer this question let us consider an Euclidean point with homogeneous coordinates

$$\mathbf{p}(t) = \begin{bmatrix} at + c \\ bt + d \\ 1 \end{bmatrix}.$$

When t varies from minus infinity to plus infinity, this point traces in w=1 an Euclidean line with direction defined by (a, b). Because these are homogeneous coordinates, we can divide all of them by t, without affecting the corresponding point:

$$\mathbf{p}(t) = \begin{bmatrix} a + c/t \\ b + c/t \\ 1/t \end{bmatrix}.$$

When t tends to either plus or minus infinity, the homogeneous coordinates of **p** tend to

$$\begin{bmatrix} a \\ b \\ 0 \end{bmatrix}$$

Therefore this is a *point at infinity* along the line of direction (*a, b*). It is not an Euclidean point, but it is a projective point. We see that the projective plane consists of all the points of the Euclidean plane augmented by the points at infinity.

Note that all parallel lines with a given direction have the same point at infinity. More surprising is the fact that the same point at infinity is reached travelling towards both plus and minus infinity. This implies that a projective line is more like a circle than an ordinary line. The projective plane is a closed surface with interesting properties (which we will not investigate in this course).

2.5.3 Change of Frames

Consider two frames **E** = (*E*, **q**) and **F** = (*F*, **r**), as shown in Figure 2.5.3.1. A generic point **p** corresponds to two different vectors in **E** and **F**, satisfying the following relations:

$$\mathbf{p} \overset{\mathbf{E}}{\leftrightarrow} \boldsymbol{x}$$

$$\mathbf{p} \overset{\mathbf{F}}{\leftrightarrow} \boldsymbol{y}$$

$$\boldsymbol{x} = \delta + \boldsymbol{y}$$

$$\delta = \mathbf{r} - \mathbf{q}$$

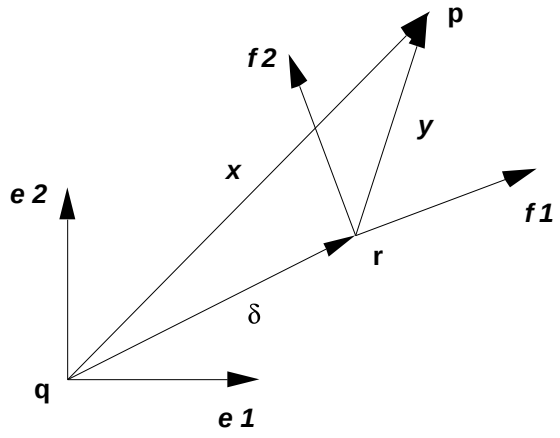

Figure 2.5.3.1 – Change of frames

The coordinate matrices of **p** in frames **E** and **F**, denoted P^e and P^f, are given by

$$P^f = Y^f$$
$$P^e = X^e = Y^e + D^e = M_{ef}^e Y^f + D^e = M_{ef}^e P^f + D^e$$

where D^e is the column matrix of the components of δ in basis E, and M_{ef}^e, as usual, denotes the matrix that corresponds in basis E to the linear transformation that maps the vectors of basis E onto those of basis F. The second equation immediately above follows from coordinate relations in vector translation and change of basis, derived earlier. The two equations may be summarized in matrix form as

$$\begin{bmatrix} P^e \\ 1 \end{bmatrix} = \begin{bmatrix} M_{ef}^e & D^e \\ 0 & 1 \end{bmatrix} \begin{bmatrix} P^f \\ 1 \end{bmatrix}$$

or

$$\left(P^*\right)^e = \left(M_{ef}^*\right)^e \left(P^*\right)^f$$

with

$$\left(P^*\right)^e = \begin{bmatrix} P^e \\ 1 \end{bmatrix} \quad \left(P^*\right)^f = \begin{bmatrix} P^f \\ 1 \end{bmatrix} \quad \left(M_{ef}^*\right)^e = \begin{bmatrix} M_{ef}^e & D^e \\ 0 & 1 \end{bmatrix}$$

These equations show that the effect of a change of frame on the homogeneous coordinates of a point is analogous to the effect of a change of basis on the components of a vector. Whereas a change of basis is associated with multiplication by a matrix M_{ef}^e, a change of frame involves an augmented matrix. This matrix has a simple interpretation. First note that the frame **F** may be defined by an affine transformation that maps the origin **q** of frame **E** onto the new origin **r**, and maps the basis E onto the new basis F. This transformation may be decomposed into a rotation that maps E onto F, followed by a translation by the vector δ. In homogeneous coordinates the composition corresponds to the matrix product

$$\begin{bmatrix} I & D^e \\ 0 & 1 \end{bmatrix} \begin{bmatrix} M_{ef}^e & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} M_{ef}^e & D^e \\ 0 & 1 \end{bmatrix} = \left(M_{ef}^*\right)^e .$$

Therefore $\left(M_{ef}^*\right)^e$ is the matrix that corresponds to the projective transformation that maps frame **E** onto frame **F**. This is a direct analog of the situation we encountered in a change of basis.

We know that the columns of matrix M_{ef}^e consist of the components of the vectors of F in the basis E. Therefore, the frame **F** = (**r**, F) corresponds to the homogeneous-coordinate matrix

$$\left(M_{ef}^*\right)^e = \begin{bmatrix} F_1^e & \cdots & F_n^e & D^e \\ 0 & \cdots & 0 & 1 \end{bmatrix}.$$

This matrix may also be interpreted as follows. The first *n* columns contain the coordinates of the points at infinity in the directions of the vectors of basis *F*. The last column is the set of coordinates of an Euclidean point, the origin **r** of the frame. All of these coordinates are relative to frame **E**.

2.5.4 Perspective

Thus far we have only used homogeneous-coordinate matrices with a last row whose off-diagonal elements are null. Let us now investigate wnat happens when they are non-null. Consider the product

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1/d & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 - x/d \end{bmatrix}.$$

(We use a $-1/d$ term for reasons that will be obvious soon.) The result is no longer on the $w=1$ plane. Normalizing it we obtain

$$\begin{bmatrix} \dfrac{x}{1 - x/d} \\ \dfrac{y}{1 - x/d} \\ 1 \end{bmatrix}.$$

What is the physical meaning of this transformation? We will answer this question with the help of Figure 2.5.4.1, which shows how to project a point on the y axis of the Euclidean plane from a center of projection **v** lying on the x axis at $x=d$. By similarity of triangles

$$\frac{y'}{y} = \frac{d}{d - x}, \quad y' = \frac{y}{1 - x/d}.$$

This is precisely the y coordinate we computed above by matrix multiplication.

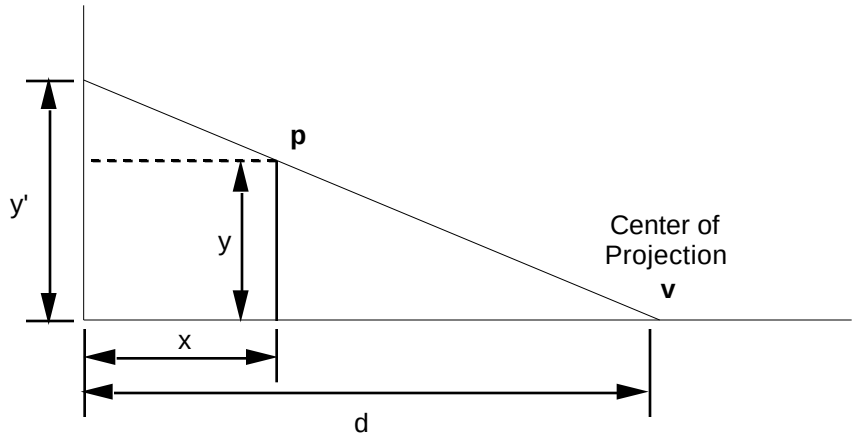

Figure 2.5.4.1 – Central projection of a 2-D point on the vertical axis

In 3-D, an analogous argument shows that multiplication by the matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/d & 1 \end{bmatrix}$$

provides us with the x and y coordinates of the projection of a point on the xy plane, from a center of projection on the z axis at $z=d$. Projection on a plane is a fundamental operation for the generation of a display—see Figure 2.5.4.2.

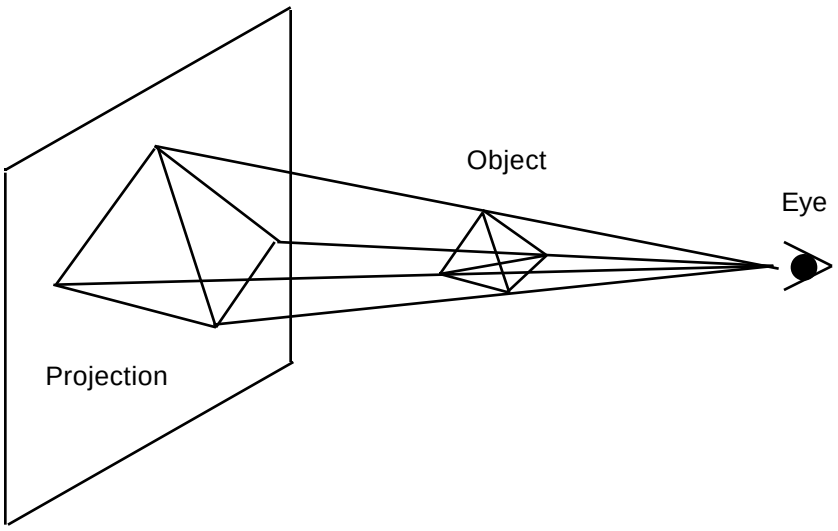

Figure 2.5.4.2 – Drawing an object by projecting it on a plane

The two-dimensional perspective transformation discussed earlier affected both the x and y coordinates of a point. Figure 2.5.4.3 illustrates the effect of a perspective on a rectangle. The result is *not* a 1-D projection on the y axis. Rather, it is a deformed 2-D object. *Orthographic* projection of this deformed object on the y axis produces the desired 1-D image.

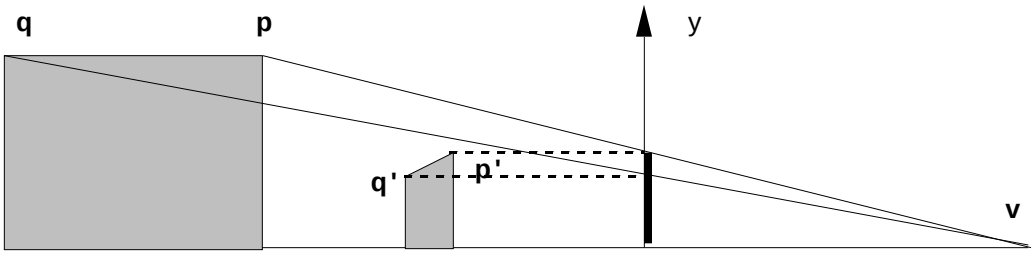

Figure 2.5.4.3 – Perspective transformation applied to a 2-D solid

In 3-D the perspective transformation produces a deformed 3-D object, which must be projected orthographically onto the *xy* plane to generate the desired 2-D image. Computing a planar projection involves matrix multiplication, followed by normalization and orthographic projection. This latter involves essentially no computation, since it amounts to ignoring the *z* coordinate. But normalization is relatively expensive, because it requires a division.

It is easy to prove that a perspective maps lines onto lines. It also maps line segments onto line segments, but there is a subtlety, illustrated in Figure 2.5.4.4. Observe that the projection of the line segment **pq** on the horizontal axis and with viewpoint (i.e., center of projection) **v** extends to infinity on the right and on the left. (It is a segment of a projective line that goes through a point at infinity.) This happens because **q** is behind the viewpoint, and one of the projecting rays is horizontal and meets the horizontal "screen" at infinity.

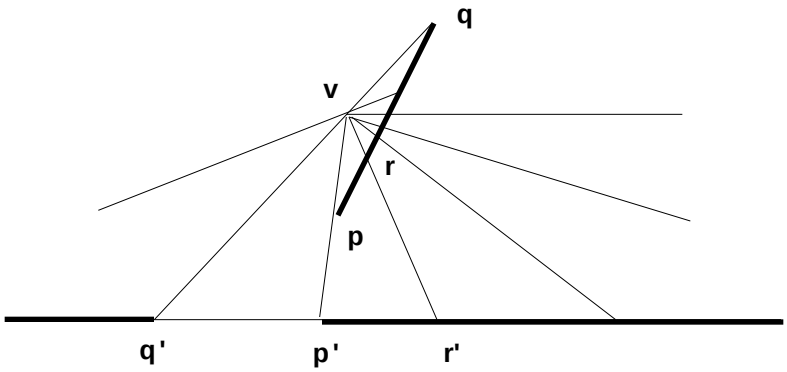

Figure 2.5.4.4 – Projection of a line segment

A line drawing of a polyhedral object may be produced by projecting the endpoints of its edges and connecting the projected endpoints with line segments, provided that the object does not extend behind the viewpoint. To ensure that this condition is satisfied in an arbitrary scene, one must clip it, i.e., remove those portions that lie behind the viewpoint, before computing the projection of the scene.

2.6 Applications in Robotics and Simulation

A robotic manipulator is a kinematic chain, i.e., a collection of solid bodies—called *links*—connected at *joints.* The most common joints are the *revolute* joint, which corresponds to rotational motion between two links, and the *prismatic* joint, which corresponds to a translation. Most of the industrial robot "arms" in use today have only revolute joints. Figure 2.6.1 shows an idealized robot with two links and two revolute joints. The first, vertical link rotates by an angle θ about its axis, and the second link rotates by an angle ϕ in the vertical plane defined by the two links. The angles θ and ϕ are called in robotics *joint angles*.

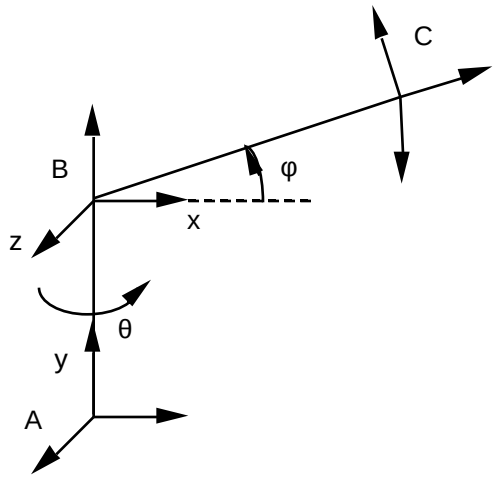

Figure 2.6.1 – Stick-figure model for a 2-link robot

A robot interacts with the objects involved in a task primarily through its "hand", or end effector. The pose of the hand with respect to the "lab" frame is crucial, and one must be able to relate hand coordinates to lab coordinates. This can be done by rigidly attaching coordinate frames to each link, and then performing successive changes of frames along the kinematic chain. Frame A in the figure is the base, or lab frame. Frame B is attached to the first link, and therefore rotates about the y (vertical) axis of A. Frame C is attached to the second link and rotates about the z axis of B. C is the hand frame of the robot. The relationship between hand and base coordinates may be expressed in terms of the matrices that describe the relative motions of the two links in each joint:

$$X^a = M^a_{ab} X^b = M^a_{ab} M^b_{bc} X^c .$$

Frame B may be constructed by first translating A by the length L_1 of the first link along y, and then rotating by θ about the y axis in basis A. Therefore

$$M^a_{ab}(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & L_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & L_1 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Frame *C* may be obtained by a translation along the *x* axis of *B* by the length L_2 of the second link, followed by a rotation about the *z* axis of *B*:

$$M^b_{bc}(\phi) = \begin{bmatrix} \cos\phi & -\sin\phi & 0 & 0 \\ \sin\phi & \cos\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & L_2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\phi & -\sin\phi & 0 & L_2\cos\phi \\ \sin\phi & \cos\phi & 0 & L_2\sin\phi \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Note that these matrices depend on the joint angles. Let us fix some point in the hand, with coordinates X^c. The coordinates of this point in the lab frame are functions of the joint angles, and can be computed by multiplying the matrices above:

$$X^a = M^a_{ab}(\theta) M^b_{bc}(\phi) X^c$$

The robot's motors control the joint angles directly, and the relationship between hand coordinates and joint angles is fundamental in robotics. It gives us the trajectory in lab coordinates for any temporal evolution of the joint angles. The velocity and the acceleration are obtained by differentiating the positional relations. In robot programming typically one seeks to drive the hand along some specific trajectory in lab coordinates. To find the corresponding joint angles involves inverting the relationships derived above. This problem is often called "inverse kinematics".

Suppose now that we want to create a graphic simulation of the robot motion. The first link is a fixed line segment in frame *B*, with endpoints on the *y* axis of *B* at $y=0$ and $y=-L_1$. The positions of these endpoints in frame *A* may be computed by the equation

$$X^a = M^a_{ab}(\theta) X^b,$$

by using the appropriate coordinates X^b for the endpoints. Now we repeatedly increment the angle θ by some small amount, compute the endpoints of the first link, and display it at its updated position. The result is an animation of the motion of the first link. A similar procedure may be used for the second link, since we also have an equation for computing the lab coordinates of fixed points in *C*.

This simulation procedure is not restricted to robotic problems or to simple stick objects. Suppose we have, for example, a model of a human figure with various articulations and "links" of complex shapes. We associate a frame with each link and compute the matrices that relate the frames in terms of some appropriate parameters. Then, we step along the parameters and display the objects in their updated poses.

Realistic simulation of machinery can be done by the techniques outlined above. However, smooth and realistic animation of the motion of humans and animals raises difficult problems. For example, how should we change the joint angles at the ankles, knees and

hips to achieve a realistic walk? These issues are beyond the scope of this course. They are addressed in advanced courses in computer graphics and animation.

2.7 Applications in Rendering

Processing a model to generate a display is called *rendering* in computer graphics. There are many sophisticated techniques to produce realistic displays. Underlying all of these is the need to map points of the model in "world coordinates" onto points of the screen, and this can be done by homogeneous-coordinate transformations. Computationally, what we need is a function `WorldToScreen` that takes an argument `WorldPoint` in 3-D and produces a 2-D `ScreenPoint`. Given this function, a simple line drawing of a polyhedron can be generated as follows.

```
Clip model to remove points behind view point
for each edge of the model do
DrawLine(WorldToScreen(EndPoint1),
         WorldToScreen(EndPoint2))
end
```

In this pseudo-code, `DrawLine` is a primitive drawing routine that operates in screen coordinates. Drawing packages often provide a more convenient primitive routine that draws a *polyline*, i.e., a connected set of line segments defined by an array of vertices. `DrawLine` is simply a more restricted version of `DrawPolyline`, operating only on two vertices. Clipping may be used also to select a region of the object to be rendered.

How is the `WorldToScreen` function specified and implemented? Figure 2.7.1 illustrates the geometry involved.

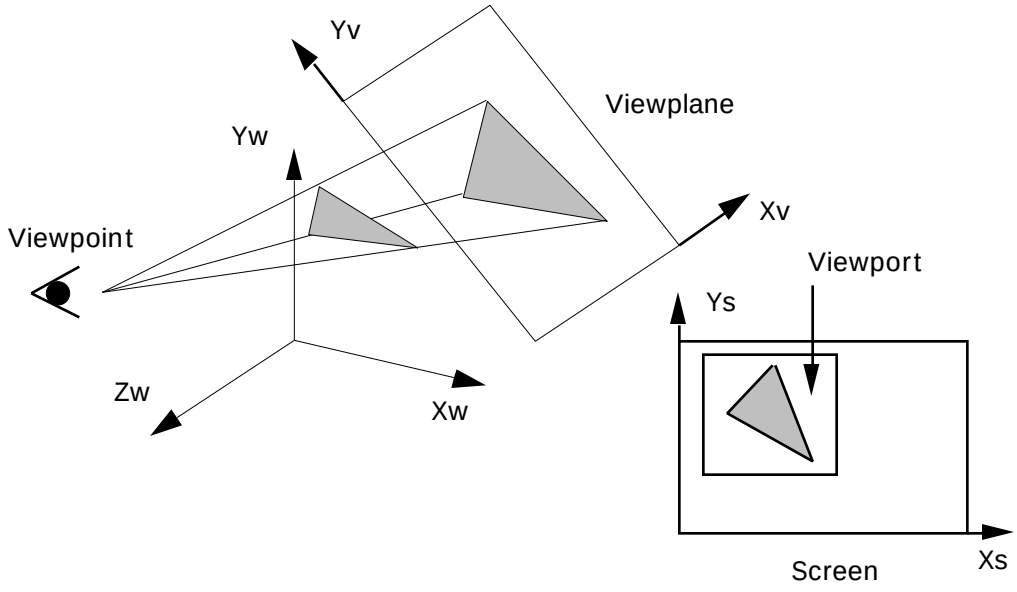

Figure 2.7.1 – World to screen transformation

The world to screen transformation is also called the *viewing transformation*. It is the composition of a projection onto the 2-D *viewplane* with a 2-D transformation between the viewplane and the *viewport*, which is the region of the screen where the image is to appear. There are several ways of specifying a set of *viewing parameters* to define the viewing transformation. The specification should be easy to understand by users, and therefore should refer to entities whose geometrical meaning is clear. Computation of the various matrices involved should be transparent to users.

Standard or quasi-standard graphic packages such as PHIGS or OpenGL use sets of viewing parameters that provide a user with great flexibility in view specification. Here we discuss a simple set of parameters which is convenient for debugging geometric algorithms. It makes several assumptions about the relations between the geometric entities involved in view specification, and trades flexibility for ease of use.

The viewing transformation is specified by the following parameters.

4. The viewpoint **p**.
5. A sphere of radius R and centered at a reference point **r**.

The user should ensure that the sphere he or she specifies encloses the object to be displayed, and that the sphere does not enclose the viewpoint. We make the following assumptions.

1. The reference point is the origin of the (x_v, y_v) coordinate system in the viewplane.
2. The viewpoint and the reference point define the line of sight. The line of sight is perpendicular to the viewplane.
3. The orientation of the viewplane coordinate system is as defined in Figure 2.7.2. In the figure we assume that the entire configuration (viewpoint, viewplane, sphere, and reference point) has been translated so that the reference point is at the origin. The frame (x_t, y_t, z_t) is constructed as follows. Its z axis coincides with the line of sight **rv**. The x axis is tangent to the parallel to the sphere at the point where the line of sight intersects the sphere. And the y axis is tangent to the meridian at the same point. The frame (x_v, y_v, z_v) is (x_t, y_t, z_t) translated to the origin, and therefore the two frames have the same orientation.
4. The viewport coincides with the entire window, whose width W and height H are known to the system through interaction with the window manager. (The window need not cover the whole screen.)

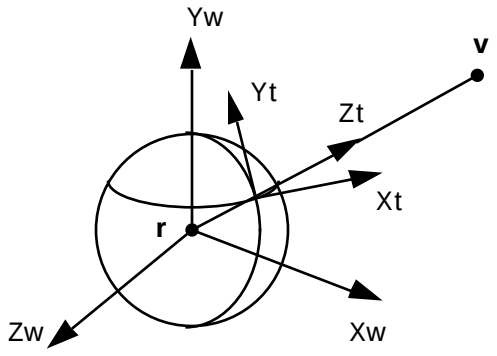

Figure 2.7.2 – Orientation of the viewplane frame

When the viewpoint is on the positive y_w axis (top view) by convention we set

$$\begin{aligned} x_v &= x_w \\ y_v &= -z_w \\ z_v &= y_w \end{aligned}.$$

And for a bottom view, when the viewpoint is on the negative y_w, axis we set

$$\begin{aligned} x_v &= x_w \\ y_v &= z_w \\ z_v &= -y_w \end{aligned}.$$

When the viewpoint is not on the y axis of the world coordinate system, the geometry in Figure 2.7.2 implies that the viewplane frame can be computed as follows:

$$\begin{aligned} \boldsymbol{z}_v &= \frac{\mathbf{v}-\mathbf{r}}{\|\mathbf{v}-\mathbf{r}\|} \\ \boldsymbol{x}_v &= \frac{\boldsymbol{y}_w \times \boldsymbol{z}_v}{\|\boldsymbol{y}_w \times \boldsymbol{z}_v\|} \\ \boldsymbol{y}_v &= \boldsymbol{z}_v \times \boldsymbol{x}_v \end{aligned}.$$

Here we denote by $\boldsymbol{x}_v$ the unit vector along the x_v axis, and similarly for the other vectors.

The viewing transformation may be computed by moving the object, viewpoint, sphere, and viewplane so that the reference point moves to the origin, the viewpoint moves to the positive z axis of the world frame, and the x and y axes of the viewplane coordinate system become coincident with those of the world frame. Because we moved the entire configuration, the projection of the object on the viewplane, in viewplane coordinates, is not affected by the motion. This projection is easy to compute in the new position, simply by using the familiar perspective transformation matrix that corresponds to a viewpoint on the z axis.

How do we find the correct motion? It is a translation that takes the reference point to the origin, followed by a rotation that maps the viewplane basis vectors onto the world basis. This rotation is easy to compute, because it is the inverse of a transformation T_{wv} that takes the world basis E_w onto the viewplane basis E_v. This latter has a corresponding matrix whose columns are the components of the vectors of E_v in basis E_w, as we saw in Section 2.2.1.

After projection on the viewplane, we need to scale the result so as to fit in the available screen viewport. Figure 2.7.3 illustrates the geometry involved.

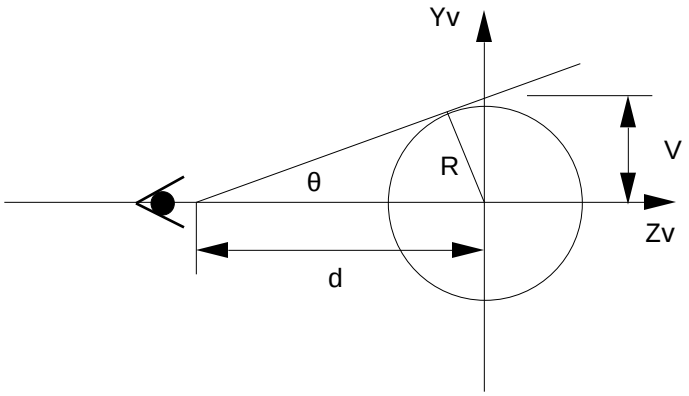

Figure 2.7.3 – Projecting a sphere on the viewplane

Note that the projection of the sphere is a disk of radius V slightly larger than R. By using simple trigonometry we find the value for V:

$$d = \|\mathbf{v} - \mathbf{r}\|$$

$$\sin\theta = \frac{R}{d}$$

$$V = \frac{R}{\cos\theta} = \frac{R}{\sqrt{1-\sin^2\theta}} = \frac{R}{\sqrt{1-(R/d)^2}}$$

The sphere projection is tightly enclosed by a square of size $2V$, centered at the origin of the viewplane. Now we have to map it into a viewport of width W and height H, as shown in Figure 2.7.4. Recall that we assume that the viewport occupies the entire window. First we scale uniformly to ensure that the transformed square fits into the available viewport area. The scale factor k must be such that $2kV$ is less than or equal to the minimum dimension of the viewport, which in general is not square, and therefore:

$$k = \frac{\min(W, H)}{2V} \ .$$

Next, we translate the scaled square so that its center coincides with the center of the viewport. (We could also choose to place it elsewhere in the viewport, but the center is perhaps best.) This requires a translation by $W/2$ in x and $H/2$ in y.

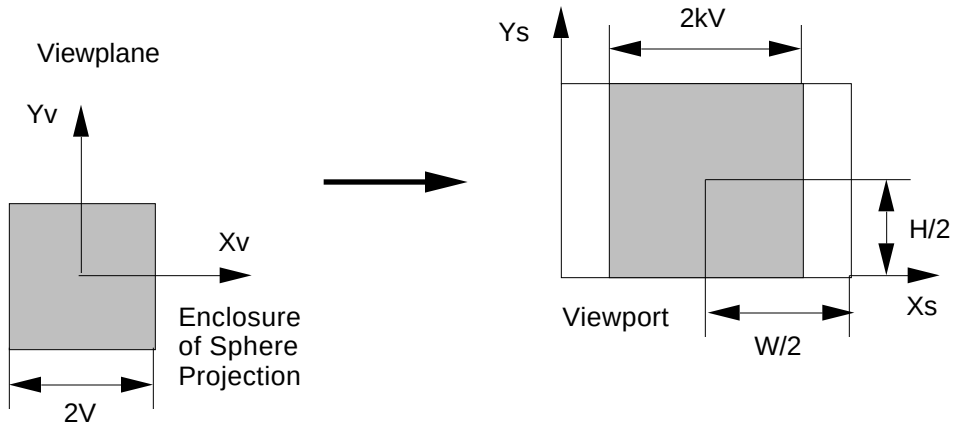

Figure 2.7.4 – Scaling and translating the sphere projection into the viewport

We can summarize the entire procedure as follows.

1. Translate the center of the sphere to the origin. This is a translation by the vector –**r**, with corresponding matrix

$$M_1 = \begin{bmatrix} 1 & 0 & 0 & -r_x \\ 0 & 1 & 0 & -r_y \\ 0 & 0 & 1 & -r_z \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

2. Rotate the axes so that the viewplane frame coincides with the world frame. To do this, first compute the viewplane basis vectors $(\boldsymbol{x}_v, \boldsymbol{y}_v, \boldsymbol{z}_v)$ as explained earlier in this section. Then, construct the matrix that corresponds to the world to viewplane basis transformation:

$$\begin{bmatrix} X_v & Y_v & Z_v & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

   Finally, invert it, which can be done by transposition because the transformation is a rotation.

$$M_2 = \begin{bmatrix} X_v^t & 0 \\ Y_v^t & 0 \\ Z_v^t & 0 \\ 0 & 1 \end{bmatrix}.$$

3. Apply a perspective with viewpoint on the *z* axis at a distance

$$d = \|\mathbf{v} - \mathbf{r}\| .$$

   The corresponding matrix is

$$M_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/d & 1 \end{bmatrix}.$$

4. Project orthographically on the xy plane:

$$M_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

5. Scale to fit in the viewport. First compute V and k as explained above, and then construct the matrix:

$$M_5 = \begin{bmatrix} k & 0 & 0 & 0 \\ 0 & k & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

6. Translate to the center of the viewport:

$$M_6 = \begin{bmatrix} 1 & 0 & 0 & W/2 \\ 0 & 1 & 0 & H/2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

7. Compute the viewing matrix by composing all the previous transformations:

$$M_{view} = M_6 M_5 M_4 M_3 M_2 M_1.$$

To transform a point from world to screen we multiply its world homogeneous coordinates by the viewing matrix, and then *normalize* the homogeneous coordinates of the result by dividing by the fourth, or w, coordinate.

We explained how to compute a viewing matrix for a simple set of viewing parameters by moving the entire scene (including the viewplane) to a convenient position, projecting, scaling, and translating to the screen. The same approach can be used with more elaborate viewing parameters. The viewing matrix is transparent to the users of a graphics package, but must be computed internally by the package whenever a user changes the viewing parameters.

2.8 Mathematical Underpinnings

There are two major aproaches for developing Euclidean geometry rigorously. They are the *synthetic* approach, based on a modern version of Euclid's postulates, which were first formulated around 300 B.C., and the *analytic* approach, which can be traced back to Descartes in the 1600s and uses algebraic methods. Synthetic geometry is a more sophisticated version of high school geometry. The analytic approach is followed in this text because it is the most useful computationally. Most of the material summarized below may be found in Halmos' classic text [Halmos 1958]. Many other good books cover linear algebra and geometry, for example [Bloom 1979], [Nomizu 1979], and [Porteous 1981].

A *vector space* defined over the real numbers is a set V together with two operations, called vector addition, denoted $\boldsymbol{x} + \boldsymbol{y}$, and scalar multiplication, denoted $a\boldsymbol{x}$ (or $a.\boldsymbol{x}$), that have the following properties, for all vectors $\boldsymbol{x}$, $\boldsymbol{y}$, $\boldsymbol{z}$ of V, and for all reals a and b. The first four properties pertain to the vector sum, and the last four to the scalar multiplication.

1. Commutativity: $\boldsymbol{x} + \boldsymbol{y} = \boldsymbol{y} + \boldsymbol{x}$
2. Associativity: $\boldsymbol{x} + (\boldsymbol{y} + \boldsymbol{z}) = (\boldsymbol{x} + \boldsymbol{y}) + \boldsymbol{z}$
3. Existence of identity: There exists a unique vector $\boldsymbol{0}$, called the zero or null vector, such that $\boldsymbol{x} + \boldsymbol{0} = \boldsymbol{x}$
4. Existence of inverse: For each $\boldsymbol{x}$ there is a unique vector $-\boldsymbol{x}$ such that $\boldsymbol{x} + (-\boldsymbol{x}) = \boldsymbol{0}$
5. Associativity: $a(b\boldsymbol{x}) = (ab)\boldsymbol{x}$
6. Existence of identity: $1\boldsymbol{x} = \boldsymbol{x}$
7. Vector distributivity: $a(\boldsymbol{x} + \boldsymbol{y}) = a\boldsymbol{x} + a\boldsymbol{y}$
8. Scalar distributivity: $(a + b)\boldsymbol{x} = a\boldsymbol{x} + b\boldsymbol{x}$

Vector spaces can be defined more generally over any set of scalars that constitute an algebraic *field*. (For basic notions of algebra see any text on discrete structures, e.g. [Preparata & Yeh 1973].) The same axioms apply, but real numbers are replaced by field elements.

An *affine space* is a set A of elements called *points*, together with a vector space V and a mapping, called *point difference*, that takes two points $\mathbf{p}$, $\mathbf{q}$ of A and produces a vector $\boldsymbol{x}$ of V, and has the following properties. (Point difference is denoted simply by a minus sign.)

1. For all $\mathbf{p}$, $\mathbf{q}$, $\mathbf{r}$ of A, $\mathbf{p} - \mathbf{r} = (\mathbf{q} - \mathbf{r}) + (\mathbf{p} - \mathbf{q})$
2. For every point $\mathbf{q}$ of A and for every vector $\boldsymbol{x}$ of V there is one and only one point $\mathbf{p}$ such that $\boldsymbol{x} = \mathbf{p} - \mathbf{q}$

Property 2 implies that one can add a vector to a point and obtain another point. It also establishes a one-to-one correspondence between A and V for a fixed "origin" $\mathbf{q}$. This correspondence can be used to extend operations defined in V to corresponding operations in A. But care must be taken to ensure that the results are not dependent on the selected origin. For example, one can show that $\mathbf{p} + \mathbf{q}$ depends on the choice of origin, and therefore is an illicit operation, whereas the average $(\mathbf{p} + \mathbf{q})/2$ is independent of origin. For other examples see [Goldman 1985].

A transformation $\boldsymbol{T}$ in a vector space V is *affine* if there exists a linear transformation $\boldsymbol{T}^*$ on V such that, for all $\boldsymbol{x}$, $\boldsymbol{y}$ of V

$$\boldsymbol{T}(\boldsymbol{x}) - \boldsymbol{T}(\boldsymbol{y}) = \boldsymbol{T}^*(\boldsymbol{x} - \boldsymbol{y}) .$$

One can show that compositions of affine transformation also are affine, and that all affine tranformations are compositions of translations and linear transformations.

Linear and affine transformations can be defined in an affine space A by using the one-to-one correspondence between A and the underlying vector space V. If a vector $\boldsymbol{x}$ corresponds to a point $\mathbf{p}$ (when $\mathbf{q}$ is selected as the origin for A), then we define $\boldsymbol{T}(\mathbf{p})$ as the point that corresponds to the vector $\boldsymbol{T}(\boldsymbol{x})$. For this definition to be meaningful for some specific $\boldsymbol{T}$, one must show that it does not depend on the origin $\mathbf{q}$.

The theory of determinants can be constructed rigorously by using vector space concepts. It can be shown that all the matrices that correspond to a linear transformation $\boldsymbol{T}$ (in different bases) have the same determinant. Transformations that have positive determinants are called *direct*, whereas transformations with negative determinants are called *opposite.* If a direct transformation maps a basis onto another, the two bases are said to be *equally oriented*, or to have the same *orientation*. If the transformation between two bases is opposite, the bases have *different orientation*. The bases of V can be divided in two equivalence classes, such that any two bases in one class have the same orientation, and any two bases from different classes have different orientations [Artzy 1978, Bloom 1979]. V is *oriented* if one of the two equivalence classes has been selected as *positive.* This selection of positive orientation is arbitrary, but it is customary to assign a positive orientation to "right-handed" bases.

The *inner*, or *dot* product, denoted $\boldsymbol{x}\,.\,\boldsymbol{y}$, is an operation that takes two vectors $\boldsymbol{x}$, $\boldsymbol{y}$ of a vector space V and produces a scalar, and that satisfies the following properties. For any $\boldsymbol{x}$, $\boldsymbol{y}$, $\boldsymbol{z}$ in V and any scalar a:

1. Commutativity: $\boldsymbol{x}\,.\,\boldsymbol{y} = \boldsymbol{y}\,.\,\boldsymbol{x}$
2. Distributivity: $\boldsymbol{x}\,.(\boldsymbol{y} + \boldsymbol{z}) = \boldsymbol{x}\,.\,\boldsymbol{y} + \boldsymbol{x}\,.\,\boldsymbol{z}$
3. Associativity: $\boldsymbol{x}\,.(a\boldsymbol{y}) = a(\boldsymbol{x}\,.\,\boldsymbol{y})$
4. $\boldsymbol{x}\,.\,\boldsymbol{x} \geq 0$
5. $\boldsymbol{x}\,.\,\boldsymbol{x} = 0$ if and only if $\boldsymbol{x} = \boldsymbol{0}$

A transformation $\boldsymbol{T}$ in a vector space V is an *isometry* if it preserves the norms of vector differences, i.e., if for all $\boldsymbol{x}$, $\boldsymbol{y}$ in V

$$\|\boldsymbol{T}(\boldsymbol{x}) - \boldsymbol{T}(\boldsymbol{y})\| = \|\boldsymbol{x} - \boldsymbol{y}\|\,.$$

A general isometry is always the composition of a translation with another isometry that does not affect the origin. Isometries that leave the zero vector invariant are called *orthogonal transformations*. It can be shown that all orthogonal transformations are linear [Bloom 1979], and therefore a general isometry is an affine transformation. It can also be shown [Halmos 1958] that a linear transformation is orthogonal if and only if the following conditions are satisfied:

$$\|\boldsymbol{T}(\boldsymbol{x})\| = \|\boldsymbol{x}\|$$
$$\boldsymbol{T}(\boldsymbol{x}).\boldsymbol{T}(\boldsymbol{y}) = \boldsymbol{x}.\boldsymbol{y}$$
$$\|\boldsymbol{T}(\boldsymbol{x}) - \boldsymbol{T}(\boldsymbol{y})\| = \|\boldsymbol{x} - \boldsymbol{y}\|$$

These conditions are equivalent: any one of them implies the other two. Thus, orthogonal transformations are those linear transformations that preserve norms and inner products of vectors. They map orthonormal bases into orthonormal bases. A matrix that corresponds to

an orthogonal transformation in an orthonormal basis is also called *orthogonal*, and has a determinant that equals either +1 or –1.

A *rotation* is defined rigorously as an orthogonal transformation with positive determinant. Rotations can be used to define precisely the notion of *signed angle.* We give here a brief outline of how this can be done, and refer the reader to texts such as [Artzy 1978], [Bloom 1979] or [Dieudonné 1969] for details. Consider the Euclidean plane and choose a positive orientation for it. Given two unit vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ of the plane there is a unique rotation R that maps $\boldsymbol{x}$ to $\boldsymbol{y}$. We associate with R an entity θ called the *angle* between $\boldsymbol{x}$ and $\boldsymbol{y}$ (in that order). We set $\theta = 0$ when $R = I$, the identity transformation, and define the angle associated with the composition $R_2 \cdot R_1$ to be the sum of the corresponding angles, $\theta_1 + \theta_2$. This suffices to derive most of the standard trigonometric notions and expressions. However, to associate a "measure", i.e., a real number, with an angle one must venture outside of geometry. Thus, consider a circle of unit radius and a rotation that corresponds to a given angle and maps a unit vector onto another. The measure of the angle in radians is the real number obtained by computing the length of the arc of circle defined by the two vectors. Clearly this construction involves notions from integral calculus.