

Another class that is useful is material. This allows you to abstract the material behavior and later add materials transparently. A simple way to link objects and materials is to add a pointer to a material in the surface class, although more programmable behavior might be desirable. A big question is what to do with textures; are they part of the material class or do they live outside of the material class? This will be discussed more in Chapter 11.

4.7 Shadows

Once you have a basic ray tracing program, shadows can be added very easily. Recall from Section 4.5 that light comes from some direction \mathbf{l} . If we imagine ourselves at a point \mathbf{p} on a surface being shaded, the point is in shadow if we “look” in direction \mathbf{l} and see an object. If there are no objects, then the light is not blocked.

This is shown in Figure 4.17, where the ray $\mathbf{p} + t\mathbf{l}$ does not hit any objects and is thus not in shadow. The point \mathbf{q} is in shadow because the ray $\mathbf{q} + t\mathbf{l}$ does hit an object. The vector \mathbf{l} is the same for both points because the light is “far” away. This assumption will later be relaxed. The rays that determine in or out of shadow are called *shadow rays* to distinguish them from viewing rays.

To get the algorithm for shading, we add an if statement to determine whether the point is in shadow. In a naive implementation, the shadow ray will check for $t \in [0, \infty)$, but because of numerical imprecision, this can result in an intersection with the surface on which \mathbf{p} lies. Instead, the usual adjustment to avoid that problem is to test for $t \in [\epsilon, \infty)$ where ϵ is some small positive constant (Figure 4.18).

If we implement shadow rays for Phong lighting with Equation 4.3 then we have the following:

```

function raycolor( ray  $\mathbf{e} + t\mathbf{d}$ , real  $t_0$ , real  $t_1$  )
hit-record rec, srec
if (scene→hit( $\mathbf{e} + t\mathbf{d}$ ,  $t_0$ ,  $t_1$ , rec)) then
     $\mathbf{p} = \mathbf{e} + (\text{rec}.t) \mathbf{d}$ 
    color  $c = \text{rec}.k_a I_a$ 
    if (not scene→hit( $\mathbf{p} + s\mathbf{l}$ ,  $\epsilon$ ,  $\infty$ , srec)) then
        vector3  $\mathbf{h} = \text{normalized}(\text{normalized}(\mathbf{l}) + \text{normalized}(-\mathbf{d}))$ 
         $c = c + \text{rec}.k_d I \max(0, \text{rec}.\mathbf{n} \cdot \mathbf{l}) + (\text{rec}.k_s) I (\text{rec}.\mathbf{n} \cdot \mathbf{h})^{\text{rec}.p}$ 
    return  $c$ 
else
    return background-color

```

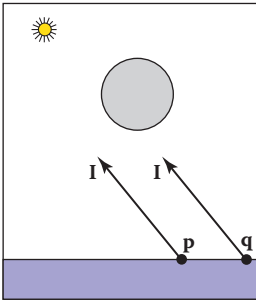


Figure 4.17. The point \mathbf{p} is not in shadow, while the point \mathbf{q} is in shadow.

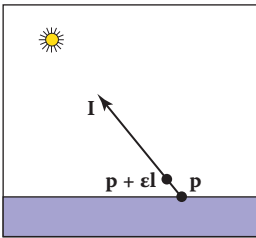


Figure 4.18. By testing in the interval starting at ϵ , we avoid numerical imprecision causing the ray to hit the surface \mathbf{p} is on.



Note that the ambient color is added whether \mathbf{p} is in shadow or not. If there are multiple light sources, we can send a shadow ray before evaluating the shading model for each light. The code above assumes that \mathbf{d} and \mathbf{l} are not necessarily unit vectors. This is crucial for \mathbf{d} , in particular, if we wish to cleanly add *instancing* later (see Section 13.2).

4.8 Ideal Specular Reflection

It is straightforward to add *ideal specular* reflection, or *mirror reflection*, to a ray-tracing program. The key observation is shown in Figure 4.19 where a viewer looking from direction \mathbf{e} sees what is in direction \mathbf{r} as seen from the surface. The vector \mathbf{r} is found using a variant of the Phong lighting reflection Equation (10.6). There are sign changes because the vector \mathbf{d} points toward the surface in this case, so,

$$\mathbf{r} = \mathbf{d} - 2(\mathbf{d} \cdot \mathbf{n})\mathbf{n}. \quad (4.5)$$

In the real world, some energy is lost when the light reflects from the surface, and this loss can be different for different colors. For example, gold reflects yellow more efficiently than blue, so it shifts the colors of the objects it reflects. This can be implemented by adding a recursive call in *raycolor*:

$$\text{color } c = c + k_m \text{raycolor}(\mathbf{p} + s\mathbf{r}, \epsilon, \infty)$$

where k_m (for “mirror reflection”) is the specular RGB color. We need to make sure we test for $s \in [\epsilon, \infty)$ for the same reason as we did with shadow rays; we don’t want the reflection ray to hit the object that generates it.

The problem with the recursive call above is that it may never terminate. For example, if a ray starts inside a room, it will bounce forever. This can be fixed by adding a maximum recursion depth. The code will be more efficient if a reflection ray is generated only if k_m is not zero (black).

4.9 Historical Notes

Ray tracing was developed early in the history of computer graphics (Appel, 1968) but was not used much until sufficient compute power was available (Kay & Greenberg, 1979; Whitted, 1980).

Ray tracing has a lower asymptotic time complexity than basic object-order rendering (Snyder & Barr, 1987; Muuss, 1995; S. Parker et al., 1999; Wald, Slusallek, Benthin, & Wagner, 2001). Although it was traditionally thought of

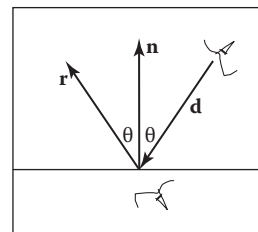


Figure 4.19. When looking into a perfect mirror, the viewer looking in direction \mathbf{d} will see whatever the viewer “below” the surface would see in direction \mathbf{r} .

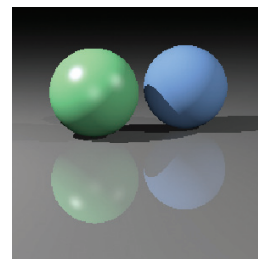


Figure 4.20. A simple scene rendered with diffuse and Blinn-Phong shading, shadows from three light sources, and specular reflection from the floor.



as an offline method, real-time ray tracing implementations are becoming more and more common.

Frequently Asked Questions

- Why is there no perspective matrix in ray tracing?

The perspective matrix in a z-buffer exists so that we can turn the perspective projection into a parallel projection. This is not needed in ray tracing, because it is easy to do the perspective projection implicitly by fanning the rays out from the eye.

- Can ray tracing be made interactive?

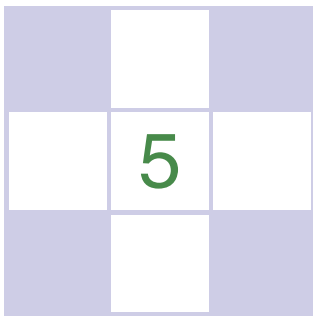
For sufficiently small models and images, any modern PC is sufficiently powerful for ray tracing to be interactive. In practice, multiple CPUs with a shared frame buffer are required for a full-screen implementation. Computer power is increasing much faster than screen resolution, and it is just a matter of time before conventional PCs can ray trace complex scenes at screen resolution.

- Is ray tracing useful in a hardware graphics program?

Ray tracing is frequently used for *picking*. When the user clicks the mouse on a pixel in a 3D graphics program, the program needs to determine which object is visible within that pixel. Ray tracing is an ideal way to determine that.

Exercises

1. What are the ray parameters of the intersection points between ray $(1, 1, 1) + t(-1, -1, -1)$ and the sphere centered at the origin with radius 1? Note: this is a good debugging case.
2. What are the barycentric coordinates and ray parameter where the ray $(1, 1, 1) + t(-1, -1, -1)$ hits the triangle with vertices $(1, 0, 0)$, $(0, 1, 0)$, and $(0, 0, 1)$? Note: this is a good debugging case.
3. Do a back of the envelope computation of the approximate time complexity of ray tracing on “nice” (non-adversarial) models. Split your analysis into the cases of preprocessing and computing the image, so that you can predict the behavior of ray tracing multiple frames for a static model.



Linear Algebra

Perhaps the most universal tools of graphics programs are the matrices that change or *transform* points and vectors. In the next chapter, we will see how a vector can be represented as a matrix with a single column, and how the vector can be represented in a different basis via multiplication with a square matrix. We will also describe how we can use such multiplications to accomplish changes in the vector such as scaling, rotation, and translation. In this chapter, we review basic linear algebra from a geometric perspective, focusing on intuition and algorithms that work well in the two- and three-dimensional case.

This chapter can be skipped by readers comfortable with linear algebra. However, there may be some enlightening tidbits even for such readers, such as the development of determinants and the discussion of singular and eigenvalue decomposition.

5.1 Determinants

We usually think of determinants as arising in the solution of linear equations. However, for our purposes, we will think of determinants as another way to multiply vectors. For 2D vectors \mathbf{a} and \mathbf{b} , the determinant $|\mathbf{ab}|$ is the area of the parallelogram formed by \mathbf{a} and \mathbf{b} (Figure 5.1). This is a signed area, and the sign is positive if \mathbf{a} and \mathbf{b} are right-handed and negative if they are left-handed. This means $|\mathbf{ab}| = -|\mathbf{ba}|$. In 2D we can interpret “right-handed” as meaning we rotate the first vector counterclockwise to close the smallest angle to the second vector. In 3D, the determinant must be taken with three vectors at a time. For

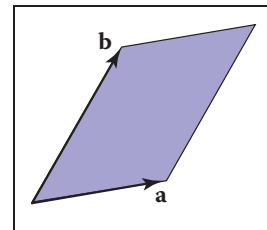


Figure 5.1. The signed area of the parallelogram is $|\mathbf{ab}|$, and in this case the area is positive.

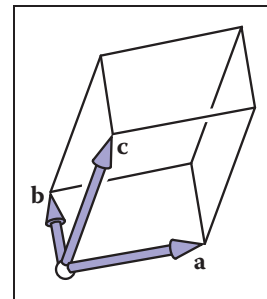


Figure 5.2. The signed volume of the parallelepiped shown is denoted by the determinant $|\mathbf{abc}|$, and in this case the volume is positive because the vectors form a right-handed basis.

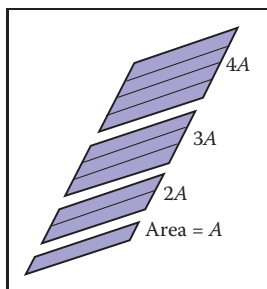


Figure 5.3. Scaling a parallelogram along one direction changes the area in the same proportion.

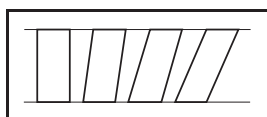


Figure 5.4. Shearing a parallelogram does not change its area. These four parallelograms have the same length base and thus the same area.

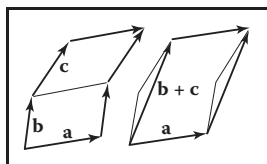


Figure 5.5. The geometry behind Equation 5.1. Both of the parallelograms on the left can be sheared to cover the single parallelogram on the right.

three 3D vectors, \mathbf{a} , \mathbf{b} , and \mathbf{c} , the determinant $|\mathbf{abc}|$ is the signed volume of the parallelepiped (3D parallelogram; a sheared 3D box) formed by the three vectors (Figure 5.2). To compute a 2D determinant, we first need to establish a few of its properties. We note that scaling one side of a parallelogram scales its area by the same fraction (Figure 5.3):

$$|(k\mathbf{a})\mathbf{b}| = |\mathbf{a}(k\mathbf{b})| = k|\mathbf{ab}|.$$

Also, we note that “shearing” a parallelogram does not change its area (Figure 5.4):

$$|(\mathbf{a} + k\mathbf{b})\mathbf{b}| = |\mathbf{a}(\mathbf{b} + k\mathbf{a})| = |\mathbf{ab}|.$$

Finally, we see that the determinant has the following property:

$$|\mathbf{a}(\mathbf{b} + \mathbf{c})| = |\mathbf{ab}| + |\mathbf{ac}|, \quad (5.1)$$

because as shown in Figure 5.5 we can “slide” the edge between the two parallelograms over to form a single parallelogram without changing the area of either of the two original parallelograms.

Now let’s assume a Cartesian representation for \mathbf{a} and \mathbf{b} :

$$\begin{aligned} |\mathbf{ab}| &= |(x_a\mathbf{x} + y_a\mathbf{y})(x_b\mathbf{x} + y_b\mathbf{y})| \\ &= x_ax_b|\mathbf{xx}| + x_ay_b|\mathbf{xy}| + y_ax_b|\mathbf{yx}| + y_ay_b|\mathbf{yy}| \\ &= x_ax_b(0) + x_ay_b(+1) + y_ax_b(-1) + y_ay_b(0) \\ &= x_ay_b - y_ax_b. \end{aligned}$$

This simplification uses the fact that $|\mathbf{vv}| = 0$ for any vector \mathbf{v} , because the parallelograms would all be collinear with \mathbf{v} and thus without area.

In three dimensions, the determinant of three 3D vectors \mathbf{a} , \mathbf{b} , and \mathbf{c} is denoted $|\mathbf{abc}|$. With Cartesian representations for the vectors, there are analogous rules for parallelepipeds as there are for parallelograms, and we can do an analogous expansion as we did for 2D:

$$\begin{aligned} |\mathbf{abc}| &= |(x_a\mathbf{x} + y_a\mathbf{y} + z_a\mathbf{z})(x_b\mathbf{x} + y_b\mathbf{y} + z_b\mathbf{z})(x_c\mathbf{x} + y_c\mathbf{y} + z_c\mathbf{z})| \\ &= x_ay_bz_c - x_az_by_c - y_ax_bz_c + y_az_bx_c + z_ax_by_c - z_ay_bx_c. \end{aligned}$$

As you can see, the computation of determinants in this fashion gets uglier as the dimension increases. We will discuss less error-prone ways to compute determinants in Section 5.3.

Example. Determinants arise naturally when computing the expression for one vector as a linear combination of two others—for example, if we wish to express a vector \mathbf{c} as a combination of vectors \mathbf{a} and \mathbf{b} :

$$\mathbf{c} = a_c\mathbf{a} + b_c\mathbf{b}.$$

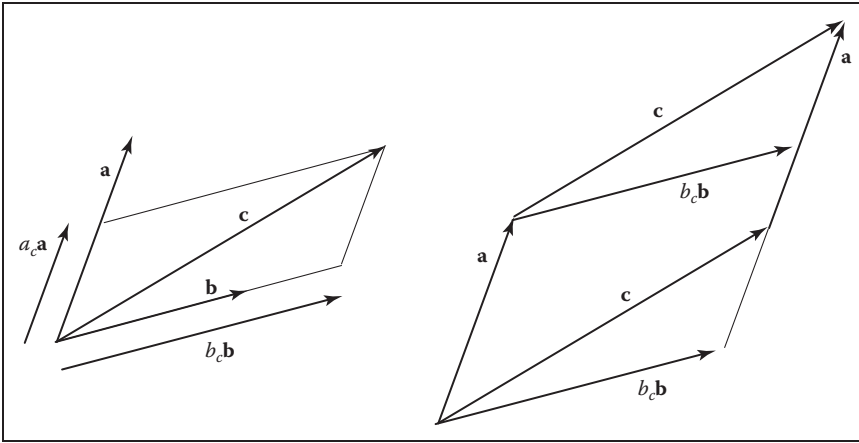


Figure 5.6. On the left, the vector \mathbf{c} can be represented using two basis vectors as $a_c \mathbf{a} + b_c \mathbf{b}$. On the right, we see that the parallelogram formed by \mathbf{a} and \mathbf{c} is a sheared version of the parallelogram formed by $b_c \mathbf{b}$ and \mathbf{a} .

We can see from Figure 5.6 that

$$|(b_c \mathbf{b})\mathbf{a}| = |\mathbf{c}\mathbf{a}|,$$

because these parallelograms are just sheared versions of each other. Solving for b_c yields

$$b_c = \frac{|\mathbf{c}\mathbf{a}|}{|\mathbf{b}\mathbf{a}|}.$$

An analogous argument yields

$$a_c = \frac{|\mathbf{b}\mathbf{c}|}{|\mathbf{b}\mathbf{a}|}.$$

This is the two-dimensional version of *Cramer's rule* which we will revisit in Section 5.3.2.



5.2 Matrices

A matrix is an array of numeric elements that follow certain arithmetic rules. An example of a matrix with two rows and three columns is

$$\begin{bmatrix} 1.7 & -1.2 & 4.2 \\ 3.0 & 4.5 & -7.2 \end{bmatrix}.$$

Matrices are frequently used in computer graphics for a variety of purposes including representation of spatial transforms. For our discussion, we assume the elements of a matrix are all real numbers. This chapter describes both the mechanics of matrix arithmetic and the *determinant* of “square” matrices, i.e., matrices with the same number of rows as columns.

5.2.1 Matrix Arithmetic

A matrix times a constant results in a matrix where each element has been multiplied by that constant, e.g.,

$$2 \begin{bmatrix} 1 & -4 \\ 3 & 2 \end{bmatrix} = \begin{bmatrix} 2 & -8 \\ 6 & 4 \end{bmatrix}.$$

Matrices also add element by element, e.g.,

$$\begin{bmatrix} 1 & -4 \\ 3 & 2 \end{bmatrix} + \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix} = \begin{bmatrix} 3 & -2 \\ 5 & 4 \end{bmatrix}.$$

For matrix multiplication, we “multiply” rows of the first matrix with columns of the second matrix:

$$\begin{bmatrix} a_{11} & \dots & a_{1m} \\ \vdots & & \vdots \\ a_{i1} & \dots & a_{im} \\ \vdots & & \vdots \\ a_{r1} & \dots & a_{rm} \end{bmatrix} \begin{bmatrix} b_{11} & \dots & b_{1j} & \dots & b_{1c} \\ \vdots & & \vdots & & \vdots \\ b_{m1} & \dots & b_{mj} & \dots & b_{mc} \end{bmatrix} = \begin{bmatrix} p_{11} & \dots & p_{1j} & \dots & p_{1c} \\ \vdots & & \vdots & & \vdots \\ p_{i1} & \dots & p_{ij} & \dots & p_{ic} \\ \vdots & & \vdots & & \vdots \\ p_{r1} & \dots & p_{rj} & \dots & p_{rc} \end{bmatrix}.$$

So the element p_{ij} of the resulting product is

$$p_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \dots + a_{im}b_{mj}. \quad (5.2)$$

Taking a product of two matrices is only possible if the number of columns of the left matrix is the same as the number of rows of the right matrix. For example,

$$\begin{bmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \end{bmatrix} \begin{bmatrix} 6 & 7 & 8 & 9 \\ 0 & 1 & 2 & 3 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 12 & 17 & 22 & 27 \\ 24 & 33 & 42 & 51 \end{bmatrix}.$$

Matrix multiplication is *not* commutative in most instances:

$$\mathbf{AB} \neq \mathbf{BA}. \quad (5.3)$$



Also, if $\mathbf{AB} = \mathbf{AC}$, it does not necessarily follow that $\mathbf{B} = \mathbf{C}$. Fortunately, matrix multiplication is associative and distributive:

$$\begin{aligned}(\mathbf{AB})\mathbf{C} &= \mathbf{A}(\mathbf{BC}), \\ \mathbf{A}(\mathbf{B} + \mathbf{C}) &= \mathbf{AB} + \mathbf{AC}, \\ (\mathbf{A} + \mathbf{B})\mathbf{C} &= \mathbf{AC} + \mathbf{BC}.\end{aligned}$$

5.2.2 Operations on Matrices

We would like a matrix analog of the inverse of a real number. We know the inverse of a real number x is $1/x$ and that the product of x and its inverse is 1. We need a matrix \mathbf{I} that we can think of as a “matrix one.” This exists only for square matrices and is known as the *identity matrix*; it consists of ones down the *diagonal* and zeroes elsewhere. For example, the four by four identity matrix is

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

The *inverse matrix* \mathbf{A}^{-1} of a matrix \mathbf{A} is the matrix that ensures $\mathbf{AA}^{-1} = \mathbf{I}$. For example,

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^{-1} = \begin{bmatrix} -2.0 & 1.0 \\ 1.5 & -0.5 \end{bmatrix} \quad \text{because} \quad \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} -2.0 & 1.0 \\ 1.5 & -0.5 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Note that the inverse of \mathbf{A}^{-1} is \mathbf{A} . So $\mathbf{AA}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$. The inverse of a product of two matrices is the product of the inverses, but with the order reversed:

$$(\mathbf{AB})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}. \quad (5.4)$$

We will return to the question of computing inverses later in the chapter.

The *transpose* \mathbf{A}^T of a matrix \mathbf{A} has the same numbers but the rows are switched with the columns. If we label the entries of \mathbf{A}^T as a'_{ij} then

$$a_{ij} = a'_{ji}.$$

For example,

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}^T = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}.$$

The transpose of a product of two matrices obeys a rule similar to Equation (5.4):

$$(\mathbf{AB})^T = \mathbf{B}^T \mathbf{A}^T.$$

The determinant of a square matrix is simply the determinant of the columns of the matrix, considered as a set of vectors. The determinant has several nice relationships to the matrix operations just discussed, which we list here for reference:

$$|\mathbf{AB}| = |\mathbf{A}| |\mathbf{B}|, \quad (5.5)$$

$$|\mathbf{A}^{-1}| = \frac{1}{|\mathbf{A}|}, \quad (5.6)$$

$$|\mathbf{A}^T| = |\mathbf{A}|. \quad (5.7)$$

5.2.3 Vector Operations in Matrix Form

In graphics, we use a square matrix to transform a vector represented as a matrix. For example, if you have a 2D vector $\mathbf{a} = (x_a, y_a)$ and want to rotate it by 90 degrees about the origin to form vector $\mathbf{a}' = (-y_a, x_a)$, you can use a product of a 2×2 matrix and a 2×1 matrix, called a *column vector*. The operation in matrix form is

$$\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_a \\ y_a \end{bmatrix} = \begin{bmatrix} -y_a \\ x_a \end{bmatrix}.$$

We can get the same result by using the transpose of this matrix and multiplying on the left (“premultiplying”) with a row vector:

$$\begin{bmatrix} x_a & y_a \end{bmatrix} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} = \begin{bmatrix} -y_a & x_a \end{bmatrix}.$$

These days, postmultiplication using column vectors is fairly standard, but in many older books and systems you will run across row vectors and premultiplication. The only difference is that the transform matrix must be replaced with its transpose.

We also can use matrix formalism to encode operations on just vectors. If we consider the result of the dot product as a 1×1 matrix, it can be written

$$\mathbf{a} \cdot \mathbf{b} = \mathbf{a}^T \mathbf{b}.$$

For example, if we take two 3D vectors we get

$$\begin{bmatrix} x_a & y_a & z_a \end{bmatrix} \begin{bmatrix} x_b \\ y_b \\ z_b \end{bmatrix} = [x_a x_b + y_a y_b + z_a z_b].$$



A related vector product is the *outer product* between two vectors, which can be expressed as a matrix multiplication with a column vector on the left and a row vector on the right: $\mathbf{a}\mathbf{b}^T$. The result is a matrix consisting of products of all pairs of an entry of \mathbf{a} with an entry of \mathbf{b} . For 3D vectors, we have

$$\begin{bmatrix} x_a \\ y_a \\ z_a \end{bmatrix} \begin{bmatrix} x_b & y_b & z_b \end{bmatrix} = \begin{bmatrix} x_a x_b & x_a y_b & x_a z_b \\ y_a x_b & y_a y_b & y_a z_b \\ z_a x_b & z_a y_b & z_a z_b \end{bmatrix}.$$

It is often useful to think of matrix multiplication in terms of vector operations. To illustrate using the three-dimensional case, we can think of a 3×3 matrix as a collection of three 3D vectors in two ways: either it is made up of three column vectors side-by-side, or it is made up of three row vectors stacked up. For instance, the result of a matrix-vector multiplication $\mathbf{y} = \mathbf{A}\mathbf{x}$ can be interpreted as a vector whose entries are the dot products of \mathbf{x} with the rows of \mathbf{A} . Naming these row vectors \mathbf{r}_i , we have

$$\begin{bmatrix} | \\ \mathbf{y} \\ | \end{bmatrix} = \begin{bmatrix} - & \mathbf{r}_1 & - \\ - & \mathbf{r}_2 & - \\ - & \mathbf{r}_3 & - \end{bmatrix} \begin{bmatrix} | \\ \mathbf{x} \\ | \end{bmatrix};$$

$$y_i = \mathbf{r}_i \cdot \mathbf{x}.$$

Alternatively, we can think of the same product as a sum of the three columns \mathbf{c}_i of \mathbf{A} , weighted by the entries of \mathbf{x} :

$$\begin{bmatrix} | \\ \mathbf{y} \\ | \end{bmatrix} = \begin{bmatrix} | & | & | \\ \mathbf{c}_1 & \mathbf{c}_2 & \mathbf{c}_3 \\ | & | & | \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix};$$

$$\mathbf{y} = x_1 \mathbf{c}_1 + x_2 \mathbf{c}_2 + x_3 \mathbf{c}_3.$$

Using the same ideas, one can understand a matrix-matrix product \mathbf{AB} as an array containing the pairwise dot products of all rows of \mathbf{A} with all columns of \mathbf{B} (cf. (5.2)); as a collection of products of the matrix \mathbf{A} with all the column vectors of \mathbf{B} , arranged left to right; as a collection of products of all the row vectors of \mathbf{A} with the matrix \mathbf{B} , stacked top to bottom; or as the sum of the pairwise outer products of all columns of \mathbf{A} with all rows of \mathbf{B} . (See Exercise 8.)

These interpretations of matrix multiplication can often lead to valuable geometric interpretations of operations that may otherwise seem very abstract.

5.2.4 Special Types of Matrices

The identity matrix is an example of a *diagonal matrix*, where all nonzero elements occur along the diagonal. The diagonal consists of those elements whose column index equals the row index counting from the upper left.



The idea of an orthogonal matrix corresponds to the idea of an *orthonormal* basis, not just a set of *orthogonal* vectors—an unfortunate glitch in terminology.

The identity matrix also has the property that it is the same as its transpose. Such matrices are called *symmetric*.

The identity matrix is also an *orthogonal* matrix, because each of its columns considered as a vector has length 1 and the columns are orthogonal to one another. The same is true of the rows (see Exercise 2). The determinant of any orthogonal matrix is either $+1$ or -1 .

A very useful property of orthogonal matrices is that they are nearly their own inverses. Multiplying an orthogonal matrix by its transpose results in the identity,

$$\mathbf{R}^T \mathbf{R} = I = \mathbf{R} \mathbf{R}^T \quad \text{for orthogonal } \mathbf{R}.$$

This is easy to see because the entries of $\mathbf{R}^T \mathbf{R}$ are dot products between the columns of \mathbf{R} . Off-diagonal entries are dot products between orthogonal vectors, and the diagonal entries are dot products of the (unit-length) columns with themselves.

Example. The matrix

$$\begin{bmatrix} 8 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 9 \end{bmatrix}$$

is diagonal, and therefore symmetric, but not orthogonal (the columns are orthogonal but they are not unit length).

The matrix

$$\begin{bmatrix} 1 & 1 & 2 \\ 1 & 9 & 7 \\ 2 & 7 & 1 \end{bmatrix}$$

is symmetric, but not diagonal or orthogonal.

The matrix

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

is orthogonal, but neither diagonal nor symmetric.



5.3 Computing with Matrices and Determinants

Recall from Section 5.1 that the determinant takes n n -dimensional vectors and combines them to get a signed n -dimensional volume of the n -dimensional parallelepiped defined by the vectors. For example, the determinant in 2D is the area

of the parallelogram formed by the vectors. We can use matrices to handle the mechanics of computing determinants.

If we have 2D vectors \mathbf{r} and \mathbf{s} , we denote the determinant $|\mathbf{rs}|$; this value is the signed area of the parallelogram formed by the vectors. Suppose we have two 2D vectors with Cartesian coordinates (a, b) and (A, B) (Figure 5.7). The determinant can be written in terms of column vectors or as a shorthand:

$$\left| \begin{bmatrix} a \\ b \end{bmatrix} \quad \begin{bmatrix} A \\ B \end{bmatrix} \right| \equiv \begin{vmatrix} a & A \\ b & B \end{vmatrix} = aB - Ab. \quad (5.8)$$


Note that the determinant of a matrix is the same as the determinant of its transpose:

$$\begin{vmatrix} a & A \\ b & B \end{vmatrix} = \begin{vmatrix} a & b \\ A & B \end{vmatrix} = aB - Ab.$$

This means that for any parallelogram in 2D there is a “sibling” parallelogram that has the same area but a different shape (Figure 5.8). For example, the parallelogram defined by vectors $(3, 1)$ and $(2, 4)$ has area 10, as does the parallelogram defined by vectors $(3, 2)$ and $(1, 4)$.

Example. The geometric meaning of the 3D determinant is helpful in seeing why certain formulas make sense. For example, the equation of the plane through the points (x_i, y_i, z_i) for $i = 0, 1, 2$ is

$$\begin{vmatrix} x - x_0 & x - x_1 & x - x_2 \\ y - y_0 & y - y_1 & y - y_2 \\ z - z_0 & z - z_1 & z - z_2 \end{vmatrix} = 0.$$

Each column is a vector from point (x_i, y_i, z_i) to point (x, y, z) . The volume of the parallelepiped with those vectors as sides is zero only if (x, y, z) is coplanar with the three other points. Almost all equations involving determinants have similarly simple underlying geometry. 

As we saw earlier, we can compute determinants by a brute force expansion where most terms are zero, and there is a great deal of bookkeeping on plus and minus signs. The standard way to manage the algebra of computing determinants is to use a form of *Laplace’s expansion*. The key part of computing the determinant this way is to find *cofactors* of various matrix elements. Each element of a square matrix has a cofactor which is the determinant of a matrix with one fewer row and column possibly multiplied by minus one. The smaller matrix is obtained by eliminating the row and column that the element in question is in. For example, for a 10×10 matrix, the cofactor of a_{82} is the determinant of the 9×9 matrix with the 8th row and 2nd column eliminated. The sign of a cofactor is positive if

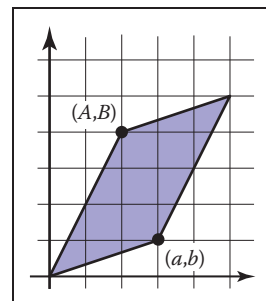


Figure 5.7. The 2D determinant in Equation 5.8 is the area of the parallelogram formed by the 2D vectors.

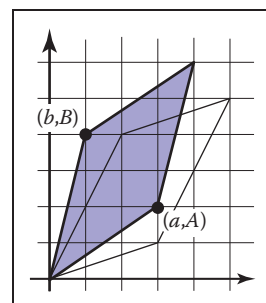


Figure 5.8. The sibling parallelogram has the same area as the parallelogram in Figure 5.7.



the sum of the row and column indices is even and negative otherwise. This can be remembered by a checkerboard pattern:

$$\begin{bmatrix} + & - & + & - & \cdots \\ - & + & - & + & \cdots \\ + & - & + & - & \cdots \\ - & + & - & + & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}.$$

So, for a 4×4 matrix,

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}.$$

The cofactors of the first row are

$$\begin{aligned} a_{11}^c &= \begin{vmatrix} a_{22} & a_{23} & a_{24} \\ a_{32} & a_{33} & a_{34} \\ a_{42} & a_{43} & a_{44} \end{vmatrix}, & a_{12}^c &= - \begin{vmatrix} a_{21} & a_{23} & a_{24} \\ a_{31} & a_{33} & a_{34} \\ a_{41} & a_{43} & a_{44} \end{vmatrix}, \\ a_{13}^c &= \begin{vmatrix} a_{21} & a_{22} & a_{24} \\ a_{31} & a_{32} & a_{34} \\ a_{41} & a_{42} & a_{44} \end{vmatrix}, & a_{14}^c &= - \begin{vmatrix} a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \end{vmatrix}. \end{aligned}$$

The determinant of a matrix is found by taking the sum of products of the elements of any row or column with their cofactors. For example, the determinant of the 4×4 matrix above taken about its second column is


$$|\mathbf{A}| = a_{12}a_{12}^c + a_{22}a_{22}^c + a_{32}a_{32}^c + a_{42}a_{42}^c.$$

We could do a similar expansion about any row or column and they would all yield the same result. Note the recursive nature of this expansion.

Example. A concrete example for the determinant of a particular 3×3 matrix by expanding the cofactors of the first row is

$$\begin{aligned} \begin{vmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{vmatrix} &= 0 \begin{vmatrix} 4 & 5 \\ 7 & 8 \end{vmatrix} - 1 \begin{vmatrix} 3 & 5 \\ 6 & 8 \end{vmatrix} + 2 \begin{vmatrix} 3 & 4 \\ 6 & 7 \end{vmatrix} \\ &= 0(32 - 35) - 1(24 - 30) + 2(21 - 24) \\ &= 0. \end{aligned}$$



We can deduce that the volume of the parallelepiped formed by the vectors defined by the columns (or rows since the determinant of the transpose is the same) is zero. This is equivalent to saying that the columns (or rows) are not linearly independent. Note that the sum of the first and third rows is twice the second row, which implies linear dependence. 

5.3.1 Computing Inverses

Determinants give us a tool to compute the inverse of a matrix. It is a very inefficient method for large matrices, but often in graphics our matrices are small. A key to developing this method is that the determinant of a matrix with two identical rows is zero. This should be clear because the volume of the n -dimensional parallelepiped is zero if two of its sides are the same. Suppose we have a 4×4 \mathbf{A} and we wish to find its inverse \mathbf{A}^{-1} . The inverse is

$$\mathbf{A}^{-1} = \frac{1}{|\mathbf{A}|} \begin{bmatrix} a_{11}^c & a_{21}^c & a_{31}^c & a_{41}^c \\ a_{12}^c & a_{22}^c & a_{32}^c & a_{42}^c \\ a_{13}^c & a_{23}^c & a_{33}^c & a_{43}^c \\ a_{14}^c & a_{24}^c & a_{34}^c & a_{44}^c \end{bmatrix}.$$

Note that this is just the transpose of the matrix where elements of \mathbf{A} are replaced by their respective cofactors multiplied by the leading constant (1 or -1). This matrix is called the *adjoint* of \mathbf{A} . The adjoint is the transpose of the *cofactor* matrix of \mathbf{A} . We can see why this is an inverse. Look at the product $\mathbf{A}\mathbf{A}^{-1}$ which we expect to be the identity. If we multiply the first row of \mathbf{A} by the first column of the adjoint matrix we need to get $|\mathbf{A}|$ (remember the leading constant above divides by $|\mathbf{A}|$):

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix} \begin{bmatrix} a_{11}^c & \cdot & \cdot & \cdot \\ a_{12}^c & \cdot & \cdot & \cdot \\ a_{13}^c & \cdot & \cdot & \cdot \\ a_{14}^c & \cdot & \cdot & \cdot \end{bmatrix} = \begin{bmatrix} |\mathbf{A}| & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix}.$$

This is true because the elements in the first row of \mathbf{A} are multiplied exactly by their cofactors in the first column of the adjoint matrix which is exactly the determinant. The other values along the diagonal of the resulting matrix are $|\mathbf{A}|$ for analogous reasons. The zeros follow a similar logic:

$$\begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ a_{21} & a_{22} & a_{23} & a_{24} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix} \begin{bmatrix} a_{11}^c & \cdot & \cdot & \cdot \\ a_{12}^c & \cdot & \cdot & \cdot \\ a_{13}^c & \cdot & \cdot & \cdot \\ a_{14}^c & \cdot & \cdot & \cdot \end{bmatrix} = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ 0 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix}.$$



Note that this product is a determinant of *some* matrix:

$$a_{21}a_{11}^c + a_{22}a_{12}^c + a_{23}a_{13}^c + a_{24}a_{14}^c.$$

The matrix in fact is

$$\begin{bmatrix} a_{21} & a_{22} & a_{23} & a_{24} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}.$$

Because the first two rows are identical, the matrix is singular, and thus, its determinant is zero.

The argument above does not apply just to four by four matrices; using that size just simplifies typography. For any matrix, the inverse is the adjoint matrix divided by the determinant of the matrix being inverted. The adjoint is the transpose of the cofactor matrix, which is just the matrix whose elements have been replaced by their cofactors.

Example. The inverse of one particular three by three matrix whose determinant is 6 is

$$\begin{aligned} \begin{bmatrix} 1 & 1 & 2 \\ 1 & 3 & 4 \\ 0 & 2 & 5 \end{bmatrix}^{-1} &= \frac{1}{6} \begin{bmatrix} \begin{vmatrix} 3 & 4 \\ 2 & 5 \end{vmatrix} & -\begin{vmatrix} 1 & 2 \\ 2 & 5 \end{vmatrix} & \begin{vmatrix} 1 & 2 \\ 3 & 4 \end{vmatrix} \\ -\begin{vmatrix} 1 & 4 \\ 0 & 5 \end{vmatrix} & \begin{vmatrix} 1 & 2 \\ 0 & 5 \end{vmatrix} & -\begin{vmatrix} 1 & 2 \\ 1 & 4 \end{vmatrix} \\ \begin{vmatrix} 1 & 3 \\ 0 & 2 \end{vmatrix} & -\begin{vmatrix} 1 & 1 \\ 0 & 2 \end{vmatrix} & \begin{vmatrix} 1 & 1 \\ 1 & 3 \end{vmatrix} \end{bmatrix} \\ &= \frac{1}{6} \begin{bmatrix} 7 & -1 & -2 \\ -5 & 5 & -2 \\ 2 & -2 & 2 \end{bmatrix}. \end{aligned}$$

You can check this yourself by multiplying the matrices and making sure you get the identity. 

5.3.2 Linear Systems

We often encounter linear systems in graphics with “ n equations and n unknowns,” usually for $n = 2$ or $n = 3$. For example,

$$3x + 7y + 2z = 4,$$

$$2x - 4y - 3z = -1,$$

$$5x + 2y + z = 1.$$



Here x , y , and z are the “unknowns” for which we wish to solve. We can write this in matrix form:

$$\begin{bmatrix} 3 & 7 & 2 \\ 2 & -4 & -3 \\ 5 & 2 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 4 \\ -1 \\ 1 \end{bmatrix}.$$

A common shorthand for such systems is $\mathbf{Ax} = \mathbf{b}$ where it is assumed that \mathbf{A} is a square matrix with known constants, \mathbf{x} is an unknown column vector (with elements x , y , and z in our example), and \mathbf{b} is a column matrix of known constants.

There are many ways to solve such systems, and the appropriate method depends on the properties and dimensions of the matrix \mathbf{A} . Because in graphics we so frequently work with systems of size $n \leq 4$, we’ll discuss here a method appropriate for these systems, known as *Cramer’s rule*, which we saw earlier, from a 2D geometric viewpoint, in the example on page 90. Here, we show this algebraically. The solution to the above equation is

$$x = \frac{\begin{vmatrix} 4 & 7 & 2 \\ -1 & -4 & -3 \\ 1 & 2 & 1 \end{vmatrix}}{\begin{vmatrix} 3 & 7 & 2 \\ 2 & -4 & -3 \\ 5 & 2 & 1 \end{vmatrix}}; \quad y = \frac{\begin{vmatrix} 3 & 4 & 2 \\ 2 & -1 & -3 \\ 5 & 1 & 1 \end{vmatrix}}{\begin{vmatrix} 3 & 7 & 2 \\ 2 & -4 & -3 \\ 5 & 2 & 1 \end{vmatrix}}; \quad z = \frac{\begin{vmatrix} 3 & 7 & 4 \\ 2 & -4 & -1 \\ 5 & 2 & 1 \end{vmatrix}}{\begin{vmatrix} 3 & 7 & 2 \\ 2 & -4 & -3 \\ 5 & 2 & 1 \end{vmatrix}}.$$

The rule here is to take a ratio of determinants, where the denominator is $|\mathbf{A}|$ and the numerator is the determinant of a matrix created by replacing a column of \mathbf{A} with the column vector \mathbf{b} . The column replaced corresponds to the position of the unknown in vector \mathbf{x} . For example, y is the second unknown and the second column is replaced. Note that if $|\mathbf{A}| = 0$, the division is undefined and there is no solution. This is just another version of the rule that if \mathbf{A} is singular (zero determinant) then there is no unique solution to the equations.

5.4 Eigenvalues and Matrix Diagonalization

Square matrices have *eigenvalues* and *eigenvectors* associated with them. The eigenvectors are those *nonzero* vectors whose directions do not change when multiplied by the matrix. For example, suppose for a matrix \mathbf{A} and vector \mathbf{a} , we have

$$\mathbf{Aa} = \lambda \mathbf{a}. \quad (5.9)$$

This means we have stretched or compressed \mathbf{a} , but its direction has not changed. The scale factor λ is called the eigenvalue associated with eigenvector \mathbf{a} . Knowing

the eigenvalues and eigenvectors of matrices is helpful in a variety of practical applications. We will describe them to gain insight into geometric transformation matrices and as a step toward singular values and vectors described in the next section.

If we assume a matrix has at least one eigenvector, then we can do a standard manipulation to find it. First, we write both sides as the product of a square matrix with the vector \mathbf{a} :

$$\mathbf{A}\mathbf{a} = \lambda\mathbf{I}\mathbf{a}, \quad (5.10)$$

where \mathbf{I} is an identity matrix. This can be rewritten

$$\mathbf{A}\mathbf{a} - \lambda\mathbf{I}\mathbf{a} = 0. \quad (5.11)$$

Because matrix multiplication is distributive, we can group the matrices:

$$(\mathbf{A} - \lambda\mathbf{I})\mathbf{a} = 0. \quad (5.12)$$

This equation can only be true if the matrix $(\mathbf{A} - \lambda\mathbf{I})$ is singular, and thus its determinant is zero. The elements in this matrix are the numbers in \mathbf{A} except along the diagonal. For example, for a 2×2 matrix the eigenvalues obey

$$\begin{vmatrix} a_{11} - \lambda & a_{12} \\ a_{21} & a_{22} - \lambda \end{vmatrix} = \lambda^2 - (a_{11} + a_{22})\lambda + (a_{11}a_{22} - a_{12}a_{21}) = 0. \quad (5.13)$$

Because this is a quadratic equation, we know there are exactly two solutions for λ . These solutions may or may not be unique or real. A similar manipulation for an $n \times n$ matrix will yield an n th-degree polynomial in λ . Because it is not possible, in general, to find exact explicit solutions of polynomial equations of degree greater than four, we can only compute eigenvalues of matrices 4×4 or smaller by analytic methods. For larger matrices, numerical methods are the only option.

An important special case where eigenvalues and eigenvectors are particularly simple is symmetric matrices (where $\mathbf{A} = \mathbf{A}^T$). The eigenvalues of real symmetric matrices are always real numbers, and if they are also distinct, their eigenvectors are mutually orthogonal. Such matrices can be put into *diagonal form*:

$$\mathbf{A} = \mathbf{Q}\mathbf{D}\mathbf{Q}^T, \quad (5.14)$$

where \mathbf{Q} is an orthogonal matrix and \mathbf{D} is a diagonal matrix. The columns of \mathbf{Q} are the eigenvectors of \mathbf{A} and the diagonal elements of \mathbf{D} are the eigenvalues of \mathbf{A} . Putting \mathbf{A} in this form is also called the *eigenvalue decomposition*, because it decomposes \mathbf{A} into a product of simpler matrices that reveal its eigenvectors and eigenvalues.

Recall that an *orthogonal* matrix has *orthonormal* rows and *orthonormal* columns.



Example. Given the matrix

$$\mathbf{A} = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix},$$

the eigenvalues of \mathbf{A} are the solutions to

$$\lambda^2 - 3\lambda + 1 = 0.$$

We approximate the exact values for compactness of notation:

$$\lambda = \frac{3 \pm \sqrt{5}}{2}, \approx \begin{bmatrix} 2.618 \\ 0.382 \end{bmatrix}.$$

Now we can find the associated eigenvector. The first is the nontrivial (not $x = y = 0$) solution to the homogeneous equation,

$$\begin{bmatrix} 2 - 2.618 & 1 \\ 1 & 1 - 2.618 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

This is approximately $(x, y) = (0.8507, 0.5257)$. Note that there are infinitely many solutions parallel to that 2D vector, and we just picked the one of unit length. Similarly the eigenvector associated with λ_2 is $(x, y) = (-0.5257, 0.8507)$. This means the diagonal form of \mathbf{A} is (within some precision due to our numeric approximation):

$$\begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 0.8507 & -0.5257 \\ 0.5257 & 0.8507 \end{bmatrix} \begin{bmatrix} 2.618 & 0 \\ 0 & 0.382 \end{bmatrix} \begin{bmatrix} 0.8507 & 0.5257 \\ -0.5257 & 0.8507 \end{bmatrix}.$$

We will revisit the geometry of this matrix as a transform in the next chapter.

5.4.1 Singular Value Decomposition

We saw in the last section that any symmetric matrix can be diagonalized, or decomposed into a convenient product of orthogonal and diagonal matrices. However, most matrices we encounter in graphics are not symmetric, and the eigenvalue decomposition for nonsymmetric matrices is not nearly so convenient or illuminating, and in general involves complex-valued eigenvalues and eigenvectors even for real-valued inputs.

There is another generalization of the symmetric eigenvalue decomposition to nonsymmetric (and even non-square) matrices; it is the *singular value decomposition* (SVD). The main difference between the eigenvalue decomposition of a symmetric matrix and the SVD of a nonsymmetric matrix is that the orthogonal matrices on the left and right sides are not required to be the same in the SVD:

$$\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T.$$

We would recommend learning in this order: symmetric eigenvalues/vectors, singular values/vectors, and *then* nonsymmetric eigenvalues, which are much trickier.

Here \mathbf{U} and \mathbf{V} are two, potentially different, orthogonal matrices, whose columns are known as the left and right *singular vectors* of \mathbf{A} , and \mathbf{S} is a diagonal matrix whose entries are known as the *singular values* of \mathbf{A} . When \mathbf{A} is symmetric and has all nonnegative eigenvalues, the SVD and the eigenvalue decomposition are the same.

There is another relationship between singular values and eigenvalues that can be used to compute the SVD (though this is not the way an industrial-strength SVD implementation works). First we define $\mathbf{M} = \mathbf{A}\mathbf{A}^T$. We assume that we can perform a SVD on \mathbf{M} :

$$\mathbf{M} = \mathbf{A}\mathbf{A}^T = (\mathbf{U}\mathbf{S}\mathbf{V}^T)(\mathbf{U}\mathbf{S}\mathbf{V}^T)^T = \mathbf{U}\mathbf{S}(\mathbf{V}^T\mathbf{V})\mathbf{S}\mathbf{U}^T = \mathbf{U}\mathbf{S}^2\mathbf{U}^T.$$

The substitution is based on the fact that $(\mathbf{BC})^T = \mathbf{C}^T\mathbf{B}^T$, that the transpose of an orthogonal matrix is its inverse, and the transpose of a diagonal matrix is the matrix itself. The beauty of this new form is that \mathbf{M} is symmetric and $\mathbf{U}\mathbf{S}^2\mathbf{U}^T$ is its eigenvalue decomposition, where \mathbf{S}^2 contains the (all nonnegative) eigenvalues. Thus, we find that the singular values of a matrix are the square roots of the eigenvalues of the product of the matrix with its transpose, and the left singular vectors are the eigenvectors of that product. A similar argument allows \mathbf{V} , the matrix of right singular vectors, to be computed from $\mathbf{A}^T\mathbf{A}$.

Example. We now make this concrete with an example:

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}; \quad \mathbf{M} = \mathbf{A}\mathbf{A}^T = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}.$$

We saw the eigenvalue decomposition for this matrix in the previous section. We observe immediately

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.8507 & -0.5257 \\ 0.5257 & 0.8507 \end{bmatrix} \begin{bmatrix} \sqrt{2.618} & 0 \\ 0 & \sqrt{0.382} \end{bmatrix} \mathbf{V}^T.$$

We can solve for \mathbf{V} algebraically:

$$\mathbf{V} = (\mathbf{S}^{-1}\mathbf{U}^T\mathbf{A})^T.$$

The inverse of \mathbf{S} is a diagonal matrix with the reciprocals of the diagonal elements of \mathbf{S} . This yields

$$\begin{aligned} \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} &= \mathbf{U} \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix} \mathbf{V}^T \\ &= \begin{bmatrix} 0.8507 & -0.5257 \\ 0.5257 & 0.8507 \end{bmatrix} \begin{bmatrix} 1.618 & 0 \\ 0 & 0.618 \end{bmatrix} \begin{bmatrix} 0.5257 & 0.8507 \\ -0.8507 & 0.5257 \end{bmatrix}. \end{aligned}$$



This form used the standard symbol σ_i for the i th singular value. Again, for a symmetric matrix, the eigenvalues and the singular values are the same ($\sigma_i = \lambda_i$). We will examine the geometry of SVD further in Section 6.1.6.

Frequently Asked Questions

• Why is matrix multiplication defined the way it is rather than just element by element?

Element by element multiplication is a perfectly good way to define matrix multiplication, and indeed it has nice properties. However, in practice it is not very useful. Ultimately, most matrices are used to transform column vectors, e.g., in 3D you might have

$$\mathbf{b} = \mathbf{M}\mathbf{a},$$

where \mathbf{a} and \mathbf{b} are vectors and \mathbf{M} is a 3×3 matrix. To allow geometric operations such as rotation, combinations of all three elements of \mathbf{a} must go into each element of \mathbf{b} . That requires us to either go row-by-row or column-by-column through \mathbf{M} . That choice is made based on composition of matrices having the desired property,

$$\mathbf{M}_2(\mathbf{M}_1\mathbf{a}) = (\mathbf{M}_2\mathbf{M}_1)\mathbf{a}$$

which allows us to use one composite matrix $\mathbf{C} = \mathbf{M}_2\mathbf{M}_1$ to transform our vector. This is valuable when many vectors will be transformed by the same composite matrix. So, in summary, the somewhat weird rule for matrix multiplication is engineered to have these desired properties.

• Sometimes I hear that eigenvalues and singular values are the same thing and sometimes that one is the square of the other. Which is right?

If a real matrix \mathbf{A} is symmetric, and its eigenvalues are nonnegative, then its eigenvalues and singular values are the same. If \mathbf{A} is not symmetric, the matrix $\mathbf{M} = \mathbf{A}\mathbf{A}^T$ is symmetric and has nonnegative real eigenvalues. The singular values of \mathbf{A} and \mathbf{A}^T are the same and are the square roots of the singular/eigenvalues of \mathbf{M} . Thus, when the square root statement is made, it is because two different matrices (with a very particular relationship) are being talked about: $\mathbf{M} = \mathbf{A}\mathbf{A}^T$.

Notes

The discussion of determinants as volumes is based on *A Vector Space Approach to Geometry* (Hausner, 1998). Hausner has an excellent discussion of vector analysis and the fundamentals of geometry as well. The geometric derivation of Cramer's rule in 2D is taken from *Practical Linear Algebra: A Geometry Toolbox* (Farin & Hansford, 2004). That book also has geometric interpretations of other linear algebra operations such as Gaussian elimination. The discussion of eigenvalues and singular values is based primarily on *Linear Algebra and Its Applications* (Strang, 1988). The example of SVD of the shear matrix is based on a discussion in *Computer Graphics and Geometric Modeling* (Salomon, 1999).

Exercises

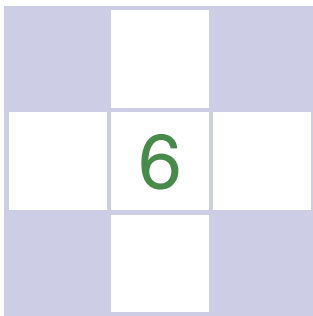
1. Write an implicit equation for the 2D line through points (x_0, y_0) and (x_1, y_1) using a 2D determinant.
2. Show that if the columns of a matrix are orthonormal, then so are the rows.
3. Prove the properties of matrix determinants stated in Equations (5.5)–(5.7).
4. Show that the eigenvalues of a diagonal matrix are its diagonal elements.
5. Show that for a square matrix \mathbf{A} , $\mathbf{A}\mathbf{A}^T$ is a symmetric matrix.
6. Show that for three 3D vectors \mathbf{a} , \mathbf{b} , \mathbf{c} , the following identity holds: $|\mathbf{abc}| = (\mathbf{a} \times \mathbf{b}) \cdot \mathbf{c}$.
7. Explain why the volume of the tetrahedron with side vectors \mathbf{a} , \mathbf{b} , \mathbf{c} (see Figure 5.2) is given by $|\mathbf{abc}|/6$.
8. Demonstrate the four interpretations of matrix-matrix multiplication by taking the following matrix-matrix multiplication code, rearranging the nested loops, and interpreting the resulting code in terms of matrix and vector operations.

```
function mat-mult(in a[m][p], in b[p][n], out c[m][n]) {
    // the array c is initialized to zero
    for i = 1 to m
        for j = 1 to n
            for k = 1 to p
                c[i][j] += a[i][k] * b[k][j]
    }
```



9. Prove that if \mathbf{A} , \mathbf{Q} , and \mathbf{D} satisfy Equation (5.14), \mathbf{v} is the i th row of \mathbf{Q} , and λ is the i th entry on the diagonal of \mathbf{D} , then \mathbf{v} is an eigenvector of \mathbf{A} with eigenvalue λ .
10. Prove that if \mathbf{A} , \mathbf{Q} , and \mathbf{D} satisfy Equation (5.14), the eigenvalues of \mathbf{A} are all distinct, and \mathbf{v} is an eigenvector of \mathbf{A} with eigenvalue λ , then for some i , \mathbf{v} is the i th row of \mathbf{Q} and λ is the i th entry on the diagonal of \mathbf{D} .
11. Given the (x, y) coordinates of the three vertices of a 2D triangle, explain why the area is given by

$$\frac{1}{2} \begin{vmatrix} x_0 & x_1 & x_2 \\ y_0 & y_1 & y_2 \\ 1 & 1 & 1 \end{vmatrix}.$$



Transformation Matrices

The machinery of linear algebra can be used to express many of the operations required to arrange objects in a 3D scene, view them with cameras, and get them onto the screen. *Geometric transformations* like rotation, translation, scaling, and projection can be accomplished with matrix multiplication, and the *transformation matrices* used to do this are the subject of this chapter.

We will show how a set of points transforms if the points are represented as offset vectors from the origin, and we will use the clock shown in Figure 6.1 as an example of a point set. So think of the clock as a bunch of points that are the ends of vectors whose tails are at the origin. We also discuss how these transforms operate differently on locations (points), displacement vectors, and surface normal vectors.

6.1 2D Linear Transformations

We can use a 2×2 matrix to change, or transform, a 2D vector:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_{11}x + a_{12}y \\ a_{21}x + a_{22}y \end{bmatrix}.$$

This kind of operation, which takes in a 2-vector and produces another 2-vector by a simple matrix multiplication, is a *linear transformation*.

By this simple formula we can achieve a variety of useful transformations, depending on what we put in the entries of the matrix, as will be discussed in the following sections. For our purposes, consider moving along the x -axis a horizontal move and along the y -axis, a vertical move.

6.1.1 Scaling

The most basic transform is a *scale* along the coordinate axes. This transform can change length and possibly direction:

$$\text{scale}(s_x, s_y) = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}.$$

Note what this matrix does to a vector with Cartesian components (x, y) :

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

So, just by looking at the matrix of an axis-aligned scale, we can read off the two scale factors.

Example. The matrix that shrinks x and y uniformly by a factor of two is (Figure 6.1)

$$\text{scale}(0.5, 0.5) = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}.$$

A matrix which halves in the horizontal and increases by three-halves in the vertical is (see Figure 6.2)

$$\text{scale}(0.5, 1.5) = \begin{bmatrix} 0.5 & 0 \\ 0 & 1.5 \end{bmatrix}.$$

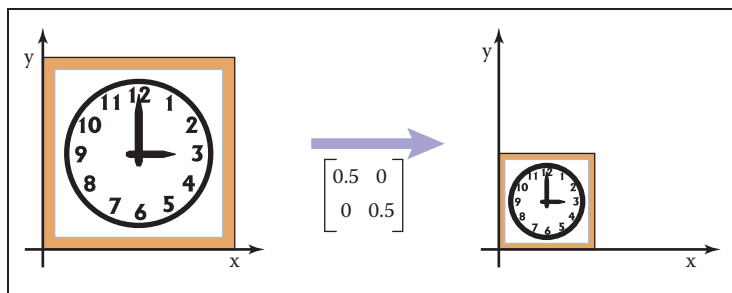


Figure 6.1. Scaling uniformly by half for each axis: The axis-aligned scale matrix has the proportion of change in each of the diagonal elements and zeroes in the off-diagonal elements.

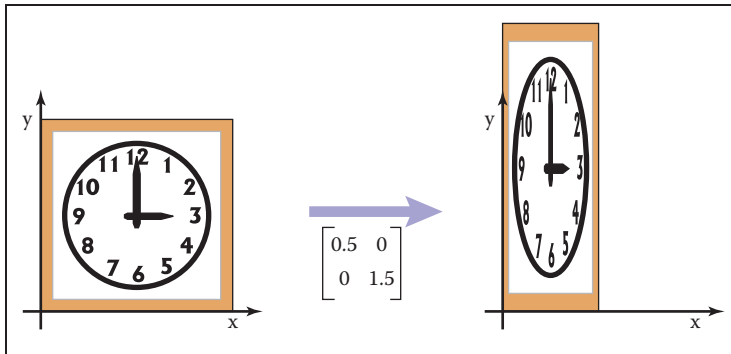


Figure 6.2. Scaling nonuniformly in x and y : The scaling matrix is diagonal with non-equal elements. Note that the square outline of the clock becomes a rectangle and the circular face becomes an ellipse.

6.1.2 Shearing

A shear is something that pushes things sideways, producing something like a deck of cards across which you push your hand; the bottom card stays put and cards move more the closer they are to the top of the deck. The horizontal and vertical shear matrices are

$$\text{shear-}x(s) = \begin{bmatrix} 1 & s \\ 0 & 1 \end{bmatrix}, \quad \text{shear-}y(s) = \begin{bmatrix} 1 & 0 \\ s & 1 \end{bmatrix}.$$

Example. The transform that shears horizontally so that vertical lines become 45° lines leaning toward the right is (see Figure 6.3)

$$\text{shear-}x(1) = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}.$$

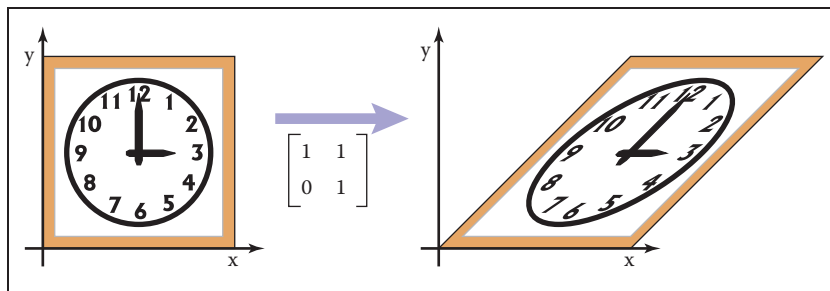


Figure 6.3. An x -shear matrix moves points to the right in proportion to their y -coordinate. Now the square outline of the clock becomes a parallelogram and, as with scaling, the circular face of the clock becomes an ellipse.

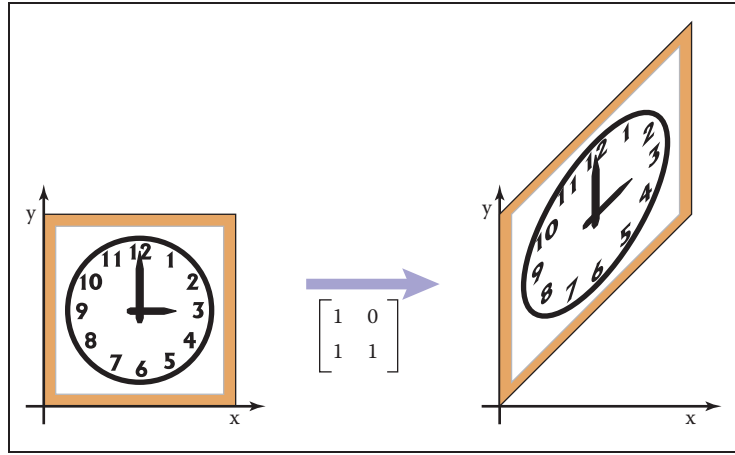



Figure 6.4. A y-shear matrix moves points up in proportion to their x -coordinate.

An analogous transform vertically is (see Figure 6.4)

$$\text{shear-}y(1) = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}.$$

In both cases, the square outline of the sheared clock becomes a parallelogram, and the circular face of the sheared clock becomes an ellipse. 

In fact, the image of a circle under any matrix transformation is an ellipse.

Another way to think of a shear is in terms of rotation of only the vertical (or horizontal) axis. The shear transform that takes a vertical axis and tilts it clockwise by an angle ϕ is

$$\begin{bmatrix} 1 & \tan \phi \\ 0 & 1 \end{bmatrix}.$$

Similarly, the shear matrix which rotates the horizontal axis counterclockwise by angle ϕ is

$$\begin{bmatrix} 1 & 0 \\ \tan \phi & 1 \end{bmatrix}.$$

6.1.3 Rotation

Suppose we want to rotate a vector \mathbf{a} by an angle ϕ counterclockwise to get vector \mathbf{b} (Figure 6.5). If \mathbf{a} makes an angle α with the x -axis, and its length is $r = \sqrt{x_a^2 + y_a^2}$, then we know that

$$x_a = r \cos \alpha,$$

$$y_a = r \sin \alpha.$$

Because \mathbf{b} is a rotation of \mathbf{a} , it also has length r . Because it is rotated an angle ϕ from \mathbf{a} , \mathbf{b} makes an angle $(\alpha + \phi)$ with the x -axis. Using the trigonometric addition identities (Section 2.3.3):

$$\begin{aligned}x_b &= r \cos(\alpha + \phi) = r \cos \alpha \cos \phi - r \sin \alpha \sin \phi, \\y_b &= r \sin(\alpha + \phi) = r \sin \alpha \cos \phi + r \cos \alpha \sin \phi.\end{aligned}\quad (6.1)$$

Substituting $x_a = r \cos \alpha$ and $y_a = r \sin \alpha$ gives

$$\begin{aligned}x_b &= x_a \cos \phi - y_a \sin \phi, \\y_b &= y_a \cos \phi + x_a \sin \phi.\end{aligned}$$

In matrix form, the transformation that takes \mathbf{a} to \mathbf{b} is then

$$\text{rotate}(\phi) = \begin{bmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{bmatrix}.$$

Example. A matrix that rotates vectors by $\pi/4$ radians (45 degrees) is (see Figure 6.6)

$$\begin{bmatrix} \cos \frac{\pi}{4} & -\sin \frac{\pi}{4} \\ \sin \frac{\pi}{4} & \cos \frac{\pi}{4} \end{bmatrix} = \begin{bmatrix} 0.707 & -0.707 \\ 0.707 & 0.707 \end{bmatrix}.$$

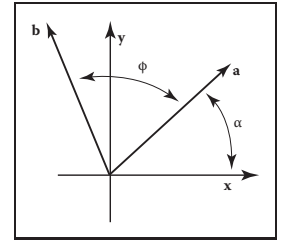


Figure 6.5. The geometry for Equation (6.1).

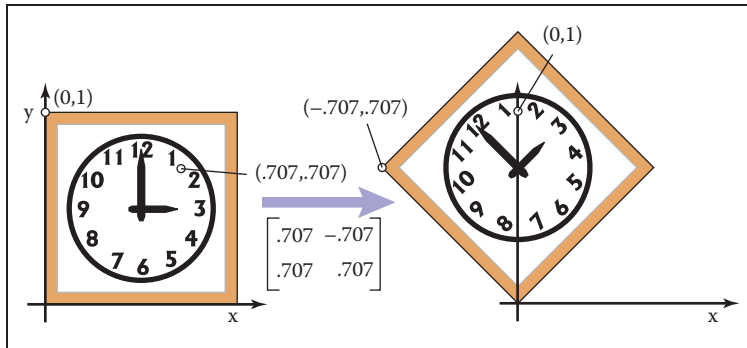


Figure 6.6. A rotation by 45° . Note that the rotation is counterclockwise and that $\cos(45^\circ) = \sin(45^\circ) \approx .707$.

A matrix that rotates by $\pi/6$ radians (30 degrees) in the *clockwise* direction is a rotation by $-\pi/6$ radians in our framework (see Figure 6.7):

$$\begin{bmatrix} \cos \frac{-\pi}{6} & -\sin \frac{-\pi}{6} \\ \sin \frac{-\pi}{6} & \cos \frac{-\pi}{6} \end{bmatrix} = \begin{bmatrix} 0.866 & 0.5 \\ -0.5 & 0.866 \end{bmatrix}.$$

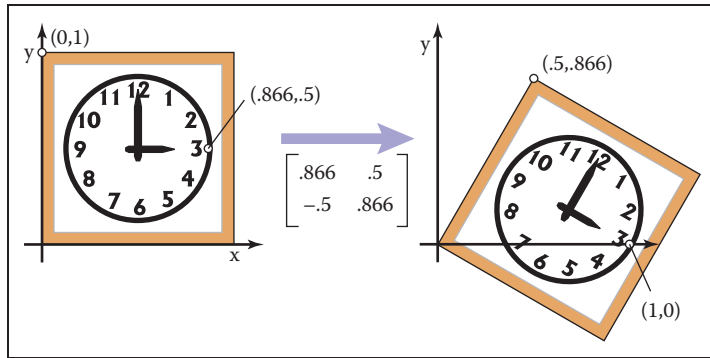


Figure 6.7. A rotation by -30 degrees. Note that the rotation is clockwise and that $\cos(-30^\circ) \approx .866$ and $\sin(-30^\circ) = -.5$.

Because the norm of each row of a rotation matrix is one ($\sin^2 \phi + \cos^2 \phi = 1$), and the rows are orthogonal ($\cos \phi (-\sin \phi) + \sin \phi \cos \phi = 0$), we see that rotation matrices are orthogonal matrices (Section 5.2.4). By looking at the matrix we can read off two pairs of orthonormal vectors: the two columns, which are the vectors to which the transformation sends the canonical basis vectors $(1, 0)$ and $(0, 1)$; and the rows, which are the vectors that the transformations sends *to* the canonical basis vectors.

Said briefly, $\mathbf{R}\mathbf{e}_i = \mathbf{u}_i$ and $\mathbf{R}\mathbf{v}_i = \mathbf{u}_i$, for a rotation with columns \mathbf{u}_i and rows \mathbf{v}_i .

6.1.4 Reflection

We can reflect a vector across either of the coordinate axes by using a scale with one negative scale factor (see Figures 6.8 and 6.9):

$$\text{reflect-y} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \text{reflect-x} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}.$$

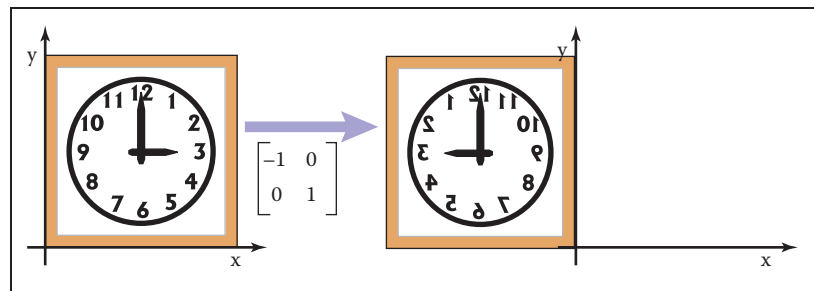


Figure 6.8. A reflection about the y -axis is achieved by multiplying all x -coordinates by -1 .

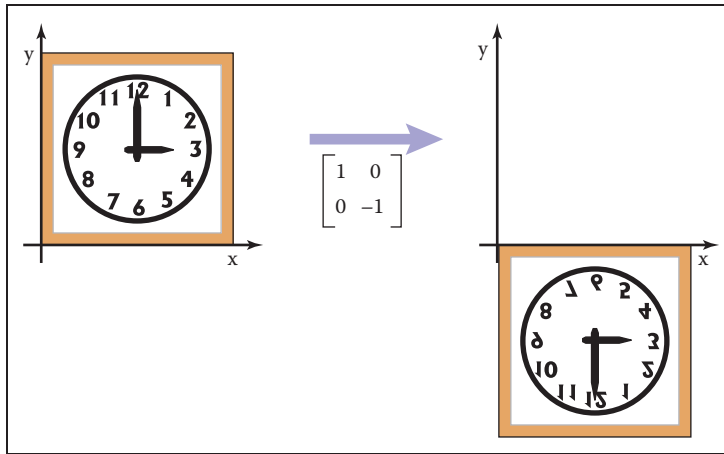


Figure 6.9. A reflection about the x -axis is achieved by multiplying all y -coordinates by -1 .

While one might expect that the matrix with -1 in both elements of the diagonal is also a reflection, in fact it is just a rotation by π radians.

This rotation can also be called a “reflection through the origin.”

6.1.5 Composition and Decomposition of Transformations

It is common for graphics programs to apply more than one transformation to an object. For example, we might want to first apply a scale \mathbf{S} , and then a rotation \mathbf{R} . This would be done in two steps on a 2D vector \mathbf{v}_1 :

$$\text{first, } \mathbf{v}_2 = \mathbf{S}\mathbf{v}_1, \text{ then, } \mathbf{v}_3 = \mathbf{R}\mathbf{v}_2.$$

Another way to write this is

$$\mathbf{v}_3 = \mathbf{R}(\mathbf{S}\mathbf{v}_1).$$

Because matrix multiplication is associative, we can also write

$$\mathbf{v}_3 = (\mathbf{RS})\mathbf{v}_1.$$

In other words, we can represent the effects of transforming a vector by two matrices in sequence using a single matrix of the same size, which we can compute by multiplying the two matrices: $\mathbf{M} = \mathbf{RS}$ (Figure 6.10).

It is *very important* to remember that these transforms are applied from the *right side first*. So the matrix $\mathbf{M} = \mathbf{RS}$ first applies \mathbf{S} and then \mathbf{R} .

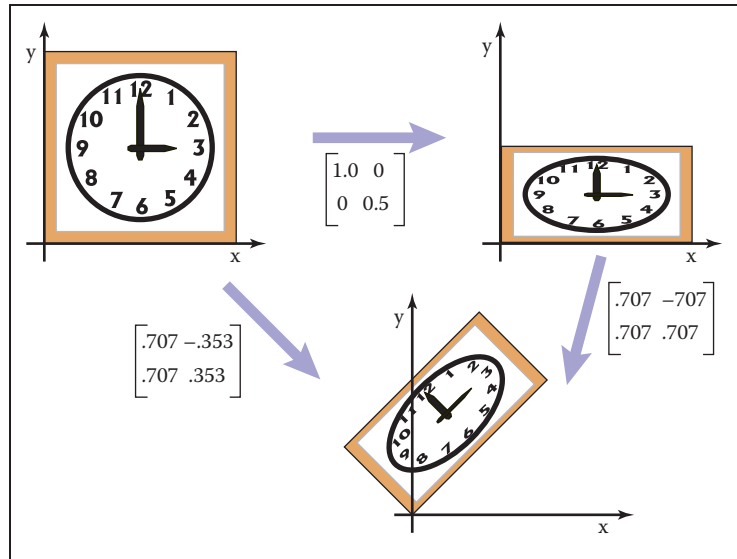


Figure 6.10. Applying the two transform matrices in sequence is the same as applying the product of those matrices once. This is a key concept that underlies most graphics hardware and software.

Example. Suppose we want to scale by one-half in the vertical direction and then rotate by $\pi/4$ radians (45 degrees). The resulting matrix is

$$\begin{bmatrix} 0.707 & -0.707 \\ 0.707 & 0.707 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 0.5 \end{bmatrix} = \begin{bmatrix} 0.707 & -0.353 \\ 0.707 & 0.353 \end{bmatrix}.$$

It is important to always remember that matrix multiplication is not commutative. So the order of transforms *does* matter. In this example, rotating first, and then scaling, results in a different matrix (see Figure 6.11):

$$\begin{bmatrix} 1 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} 0.707 & -0.707 \\ 0.707 & 0.707 \end{bmatrix} = \begin{bmatrix} 0.707 & -0.707 \\ 0.353 & 0.353 \end{bmatrix}.$$

Example. Using the scale matrices we have presented, nonuniform scaling can only be done along the coordinate axes. If we wanted to stretch our clock by 50% along one of its diagonals, so that 8:00 through 1:00 move to the northwest and 2:00 through 7:00 move to the southeast, we can use rotation matrices in combination with an axis-aligned scaling matrix to get the result we want. The idea is to use a rotation to align the scaling axis with a coordinate axis, then scale along that axis, then rotate back. In our example, the scaling axis is the “backslash” diagonal of the square, and we can make it parallel to the x -axis with

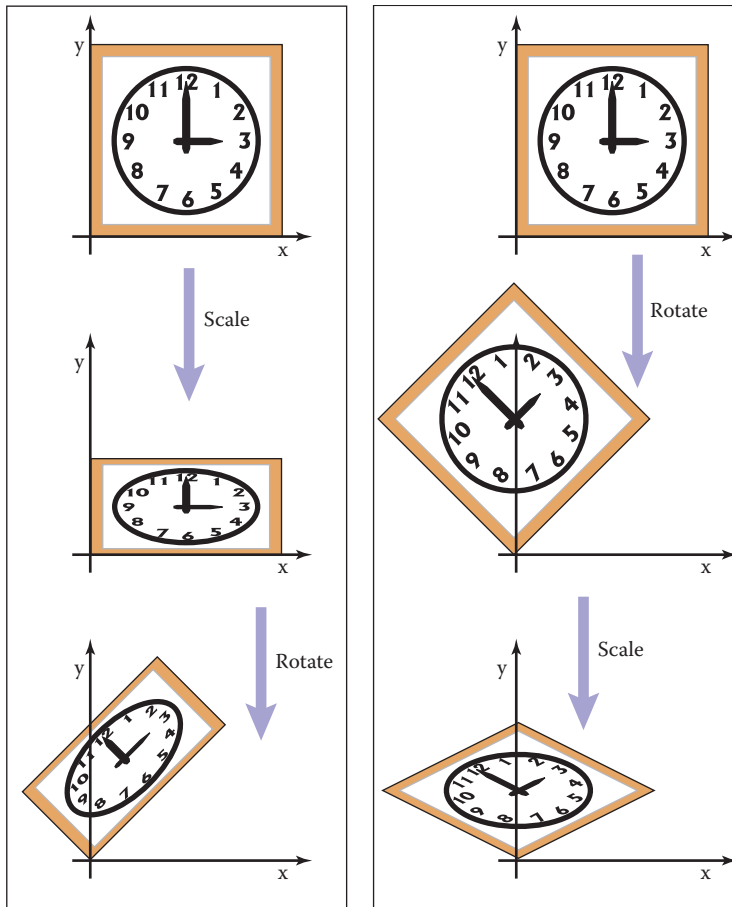


Figure 6.11. The order in which two transforms are applied is usually important. In this example, we do a scale by one-half in y and then rotate by 45° . Reversing the order in which these two transforms are applied yields a different result.

a rotation by $+45^\circ$. Putting these operations together, the full transformation is

$$\text{rotate}(-45^\circ) \text{scale}(1.5, 1) \text{rotate}(45^\circ).$$

In mathematical notation, this can be written \mathbf{RSR}^T . The result of multiplying the three matrices together is

$$\begin{bmatrix} 1.25 & -0.25 \\ -0.25 & 1.25 \end{bmatrix}$$



Remember to read the transformations from right to left.

It is no coincidence that this matrix is symmetric—try applying the transpose-of-product rule to the formula \mathbf{RSR}^T .

Building up a transformation from rotation and scaling transformations actually works for any linear transformation, and this fact leads to a powerful way of thinking about these transformations, as explored in the next section.

6.1.6 Decomposition of Transformations

Sometimes it's necessary to “undo” a composition of transformations, taking a transformation apart into simpler pieces. For instance, it's often useful to present a transformation to the user for manipulation in terms of separate rotations and scale factors, but a transformation might be represented internally simply as a

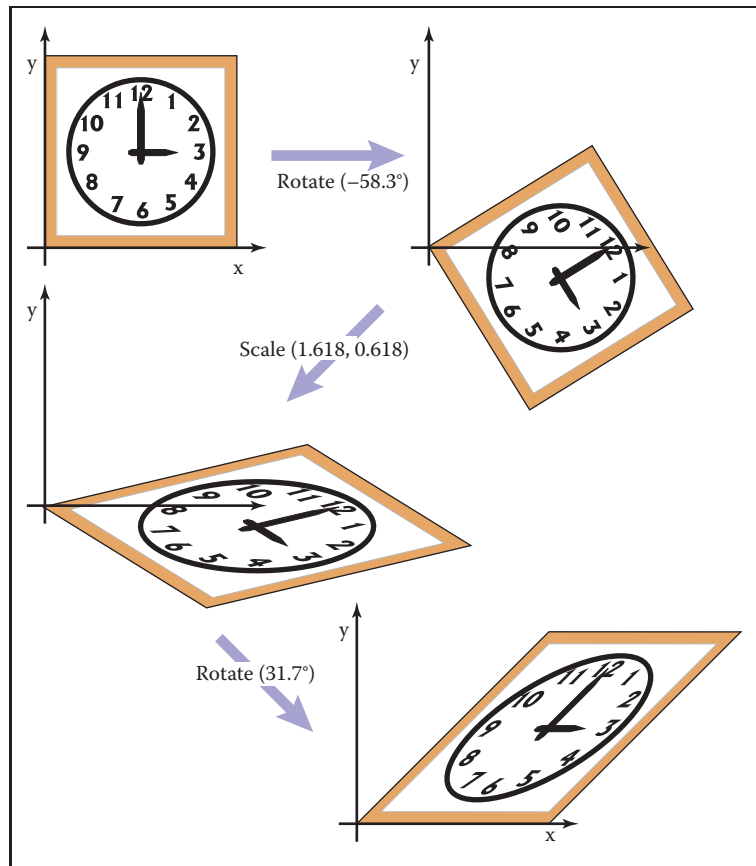


Figure 6.12. Singular Value Decomposition (SVD) for a shear matrix. Any 2D matrix can be decomposed into a product of rotation, scale, rotation. Note that the circular face of the clock must become an ellipse because it is just a rotated and scaled circle.



matrix, with the rotations and scales already mixed together. This kind of manipulation can be achieved if the matrix can be computationally disassembled into the desired pieces, the pieces adjusted, and the matrix reassembled by multiplying the pieces together again.

It turns out that this decomposition, or factorization, is possible, regardless of the entries in the matrix—and this fact provides a fruitful way of thinking about transformations and what they do to geometry that is transformed by them.

Symmetric Eigenvalue Decomposition

Let's start with symmetric matrices. Recall from Section 5.4 that a symmetric matrix can always be taken apart using the eigenvalue decomposition into a product of the form

$$\mathbf{A} = \mathbf{R}\mathbf{S}\mathbf{R}^T$$

where \mathbf{R} is an orthogonal matrix and \mathbf{S} is a diagonal matrix; we will call the columns of \mathbf{R} (the eigenvectors) by the names \mathbf{v}_1 and \mathbf{v}_2 , and we'll call the diagonal entries of \mathbf{S} (the eigenvalues) by the names λ_1 and λ_2 .

In geometric terms we can now recognize \mathbf{R} as a rotation and \mathbf{S} as a scale, so this is just a multi-step geometric transformation (Figure 6.13):

1. Rotate \mathbf{v}_1 and \mathbf{v}_2 to the x - and y -axes (the transform by \mathbf{R}^T).
2. Scale in x and y by (λ_1, λ_2) (the transform by \mathbf{S}).
3. Rotate the x - and y -axes back to \mathbf{v}_1 and \mathbf{v}_2 (the transform by \mathbf{R}).

Looking at the effect of these three transforms together, we can see that they have the effect of a nonuniform scale along a pair of axes. As with an axis-aligned scale, the axes are perpendicular, but they aren't the coordinate axes; instead they

If you like to count dimensions: a symmetric 2×2 matrix has 3 degrees of freedom, and the eigenvalue decomposition rewrites them as a rotation angle and two scale factors.

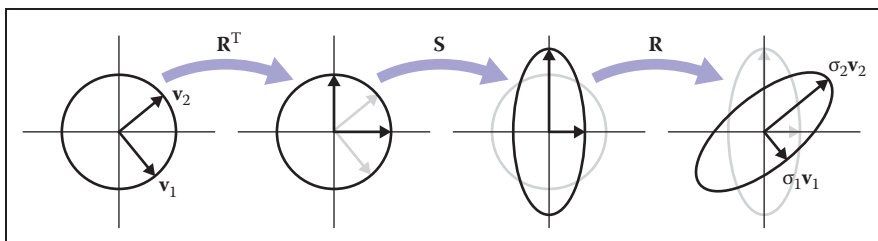


Figure 6.13. What happens when the unit circle is transformed by an arbitrary symmetric matrix \mathbf{A} , also known as a non-axis-aligned, nonuniform scale. The two perpendicular vectors \mathbf{v}_1 and \mathbf{v}_2 , which are the eigenvectors of \mathbf{A} , remain fixed in direction but get scaled. In terms of elementary transformations, this can be seen as first rotating the eigenvectors to the canonical basis, doing an axis-aligned scale, and then rotating the canonical basis back to the eigenvectors.

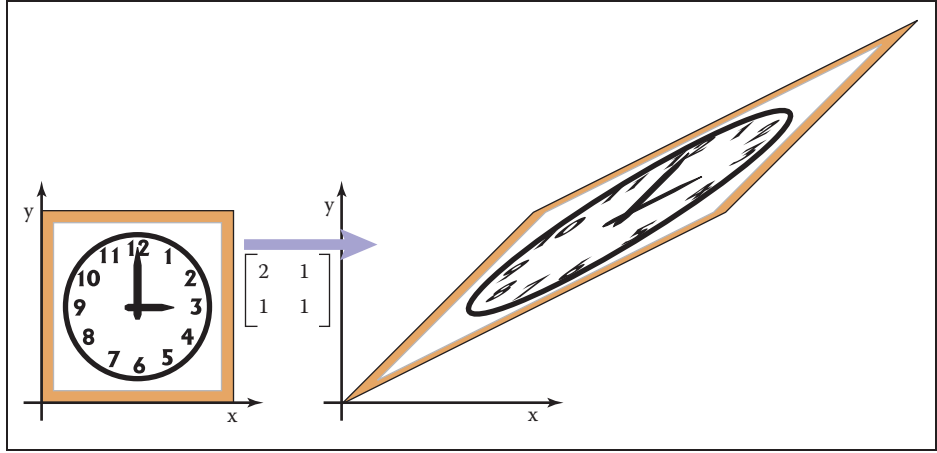


Figure 6.14. A symmetric matrix is always a scale along some axis. In this case it is along the $\phi = 31.7^\circ$ direction which means the real eigenvector for this matrix is in that direction.

are the eigenvectors of \mathbf{A} . This tells us something about what it means to be a symmetric matrix: symmetric matrices are just scaling operations—albeit potentially nonuniform and non-axis-aligned ones.

Example. Recall the example from Section 5.4:

$$\begin{aligned}
 \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} &= \mathbf{R} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \mathbf{R}^T \\
 &= \begin{bmatrix} 0.8507 & -0.5257 \\ 0.5257 & 0.8507 \end{bmatrix} \begin{bmatrix} 2.618 & 0 \\ 0 & 0.382 \end{bmatrix} \begin{bmatrix} 0.8507 & 0.5257 \\ -0.5257 & 0.8507 \end{bmatrix} \\
 &= \text{rotate } (31.7^\circ) \text{ scale } (2.618, 0.382) \text{ rotate } (-31.7^\circ).
 \end{aligned}$$

The matrix above, then, according to its eigenvalue decomposition, scales in a direction 31.7° counterclockwise from three o'clock (the x -axis). This is a touch before 2 p.m. on the clockface as is confirmed by Figure 6.14.

We can also reverse the diagonalization process; to scale by (λ_1, λ_2) with the first scaling direction an angle ϕ clockwise from the x -axis, we have

$$\begin{aligned}
 \begin{bmatrix} \cos \phi & \sin \phi \\ -\sin \phi & \cos \phi \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \begin{bmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{bmatrix} &= \\
 \begin{bmatrix} \lambda_1 \cos^2 \phi + \lambda_2 \sin^2 \phi & (\lambda_2 - \lambda_1) \cos \phi \sin \phi \\ (\lambda_2 - \lambda_1) \cos \phi \sin \phi & \lambda_2 \cos^2 \phi + \lambda_1 \sin^2 \phi \end{bmatrix}.
 \end{aligned}$$

We should take heart that this is a symmetric matrix as we know must be true since we constructed it from a symmetric eigenvalue decomposition. □

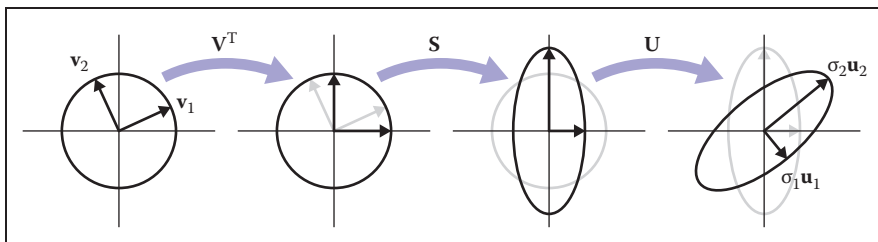


Figure 6.15. What happens when the unit circle is transformed by an arbitrary matrix \mathbf{A} . The two perpendicular vectors \mathbf{v}_1 and \mathbf{v}_2 , which are the right singular vectors of \mathbf{A} , get scaled and changed in direction to match the left singular vectors, \mathbf{u}_1 and \mathbf{u}_2 . In terms of elementary transformations, this can be seen as first rotating the right singular vectors to the canonical basis, doing an axis-aligned scale, and then rotating the canonical basis to the left singular vectors.

Singular Value Decomposition

A very similar kind of decomposition can be done with nonsymmetric matrices as well: it's the Singular Value Decomposition (SVD), also discussed in Section 5.4.1. The difference is that the matrices on either side of the diagonal matrix are no longer the same:

$$\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T$$

The two orthogonal matrices that replace the single rotation \mathbf{R} are called \mathbf{U} and \mathbf{V} , and their columns are called \mathbf{u}_i (the *left singular vectors*) and \mathbf{v}_i (the *right singular vectors*), respectively. In this context, the diagonal entries of \mathbf{S} are called *singular values* rather than eigenvalues. The geometric interpretation is very similar to that of the symmetric eigenvalue decomposition (Figure 6.15):

1. Rotate \mathbf{v}_1 and \mathbf{v}_2 to the x - and y -axes (the transform by \mathbf{V}^T).
2. Scale in x and y by (σ_1, σ_2) (the transform by \mathbf{S}).
3. Rotate the x - and y -axes to \mathbf{u}_1 and \mathbf{u}_2 (the transform by \mathbf{U}).


The principal difference is between a single rotation and two different orthogonal matrices. This difference causes another, less important, difference. Because the SVD has different singular vectors on the two sides, there is no need for negative singular values: we can always flip the sign of a singular value, reverse the direction of one of the associated singular vectors, and end up with the same transformation again. For this reason, the SVD always produces a diagonal matrix with all positive entries, but the matrices \mathbf{U} and \mathbf{V} are not guaranteed to be rotations—they could include reflection as well. In geometric applications like graphics this is an inconvenience, but a minor one: it is easy to differentiate rotations from reflections by checking the determinant, which is $+1$ for rotations

For dimension counters: a general 2×2 matrix has 4 degrees of freedom, and the SVD rewrites them as two rotation angles and two scale factors. One more bit is needed to keep track of reflections, but that doesn't add a dimension.

and -1 for reflections, and if rotations are desired, one of the singular values can be negated, resulting in a rotation–scale–rotation sequence where the reflection is rolled in with the scale, rather than with one of the rotations.

Example. The example used in Section 5.4.1 is in fact a shear matrix (Figure 6.12):

$$\begin{aligned} \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} &= \mathbf{R}_2 \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix} \mathbf{R}_1 \\ &= \begin{bmatrix} 0.8507 & -0.5257 \\ 0.5257 & 0.8507 \end{bmatrix} \begin{bmatrix} 1.618 & 0 \\ 0 & 0.618 \end{bmatrix} \begin{bmatrix} 0.5257 & 0.8507 \\ -0.8507 & 0.5257 \end{bmatrix} \\ &= \text{rotate } (31.7^\circ) \text{ scale } (1.618, 0.618) \text{ rotate } (-58.3^\circ). \end{aligned}$$

An immediate consequence of the existence of SVD is that all the 2D transformation matrices we have seen can be made from rotation matrices and scale matrices. Shear matrices are a convenience, but they are not required for expressing transformations. 

In summary, every matrix can be decomposed via SVD into a rotation times a scale times another rotation. Only symmetric matrices can be decomposed via eigenvalue diagonalization into a rotation times a scale times the inverse-rotation, and such matrices are a simple scale in an arbitrary direction. The SVD of a symmetric matrix will yield the same triple product as eigenvalue decomposition via a slightly more complex algebraic manipulation.

Paeth Decomposition of Rotations

Another decomposition uses shears to represent nonzero rotations (Paeth, 1990). The following identity allows this:

$$\begin{bmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{bmatrix} = \begin{bmatrix} 1 & \frac{\cos \phi - 1}{\sin \phi} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \sin \phi & 1 \end{bmatrix} \begin{bmatrix} 1 & \frac{\cos \phi - 1}{\sin \phi} \\ 0 & 1 \end{bmatrix}.$$

For example, a rotation by $\pi/4$ (45 degrees) is (see Figure 6.16)

$$\text{rotate}\left(\frac{\pi}{4}\right) = \begin{bmatrix} 1 & 1 - \sqrt{2} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \frac{\sqrt{2}}{2} & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 - \sqrt{2} \\ 0 & 1 \end{bmatrix}. \quad (6.2)$$

This particular transform is useful for raster rotation because shearing is a very efficient raster operation for images; it introduces some jagginess, but will

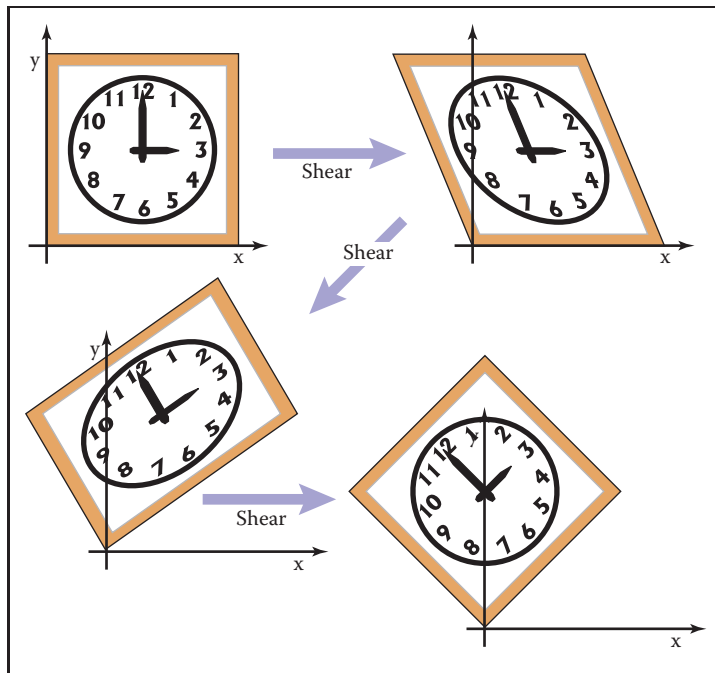


Figure 6.16. Any 2D rotation can be accomplished by three shears in sequence. In this case a rotation by 45° is decomposed as shown in Equation 6.2.

leave no holes. The key observation is that if we take a raster position (i, j) and apply a horizontal shear to it, we get

$$\begin{bmatrix} 1 & s \\ 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} i + sj \\ j \end{bmatrix}.$$

If we round sj to the nearest integer, this amounts to taking each row in the image and moving it sideways by some amount—a different amount for each row. Because it is the same displacement within a row, this allows us to rotate with no gaps in the resulting image. A similar action works for a vertical shear. Thus, we can implement a simple raster rotation easily.

6.2 3D Linear Transformations

The linear 3D transforms are an extension of the 2D transforms. For example, a scale along Cartesian axes is

$$\text{scale}(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix}. \quad (6.3)$$

Rotation is considerably more complicated in 3D than in 2D, because there are more possible axes of rotation. However, if we simply want to rotate about the z -axis, which will only change x - and y -coordinates, we can use the 2D rotation matrix with no operation on z :

$$\text{rotate-}z(\phi) = \begin{bmatrix} \cos \phi & -\sin \phi & 0 \\ \sin \phi & \cos \phi & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Similarly we can construct matrices to rotate about the x -axis and the y -axis:

$$\text{rotate-}x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \phi & -\sin \phi \\ 0 & \sin \phi & \cos \phi \end{bmatrix},$$

$$\text{rotate-}y(\phi) = \begin{bmatrix} \cos \phi & 0 & \sin \phi \\ 0 & 1 & 0 \\ -\sin \phi & 0 & \cos \phi \end{bmatrix}.$$

To understand why the minus sign is in the lower left for the y -axis rotation, think of the three axes in a circular sequence: y after x ; z after y ; x after z .

We will discuss rotations about arbitrary axes in the next section.

As in two dimensions, we can shear along a particular axis, for example,

$$\text{shear-}x(d_y, d_z) = \begin{bmatrix} 1 & d_y & d_z \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

As with 2D transforms, any 3D transformation matrix can be decomposed using SVD into a rotation, scale, and another rotation. Any symmetric 3D matrix has an eigenvalue decomposition into rotation, scale, and inverse-rotation. Finally, a 3D rotation can be decomposed into a product of 3D shear matrices.

6.2.1 Arbitrary 3D Rotations

As in 2D, 3D rotations are *orthogonal* matrices. Geometrically, this means that the three rows of the matrix are the Cartesian coordinates of three mutually orthogonal unit vectors as discussed in Section 2.4.5. The columns are three, potentially different, mutually orthogonal unit vectors. There are an infinite number of such rotation matrices. Let's write down such a matrix:

$$\mathbf{R}_{uvw} = \begin{bmatrix} x_u & y_u & z_u \\ x_v & y_v & z_v \\ x_w & y_w & z_w \end{bmatrix}.$$



Here, $\mathbf{u} = x_u\mathbf{x} + y_u\mathbf{y} + z_u\mathbf{z}$ and so on for \mathbf{v} and \mathbf{w} . Since the three vectors are orthonormal we know that

$$\begin{aligned}\mathbf{u} \cdot \mathbf{u} &= \mathbf{v} \cdot \mathbf{v} = \mathbf{w} \cdot \mathbf{w} = 1, \\ \mathbf{u} \cdot \mathbf{v} &= \mathbf{v} \cdot \mathbf{w} = \mathbf{w} \cdot \mathbf{u} = 0.\end{aligned}$$

We can infer some of the behavior of the rotation matrix by applying it to the vectors \mathbf{u} , \mathbf{v} and \mathbf{w} . For example,

$$\mathbf{R}_{uvw}\mathbf{u} = \begin{bmatrix} x_u & y_u & z_u \\ x_v & y_v & z_v \\ x_w & y_w & z_w \end{bmatrix} \begin{bmatrix} x_u \\ y_u \\ z_u \end{bmatrix} = \begin{bmatrix} x_u x_u + y_u y_u + z_u z_u \\ x_v x_u + y_v y_u + z_v z_u \\ x_w x_u + y_w y_u + z_w z_u \end{bmatrix}.$$

Note that those three rows of $\mathbf{R}_{uvw}\mathbf{u}$ are all dot products:

$$\mathbf{R}_{uvw}\mathbf{u} = \begin{bmatrix} \mathbf{u} \cdot \mathbf{u} \\ \mathbf{v} \cdot \mathbf{u} \\ \mathbf{w} \cdot \mathbf{u} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \mathbf{x}.$$

Similarly, $\mathbf{R}_{uvw}\mathbf{v} = \mathbf{y}$, and $\mathbf{R}_{uvw}\mathbf{w} = \mathbf{z}$. So \mathbf{R}_{uvw} takes the basis \mathbf{uvw} to the corresponding Cartesian axes via rotation.

If \mathbf{R}_{uvw} is a rotation matrix with orthonormal rows, then \mathbf{R}_{uvw}^T is also a rotation matrix with orthonormal columns, and in fact is the inverse of \mathbf{R}_{uvw} (the inverse of an orthogonal matrix is always its transpose). An important point is that for transformation matrices, the algebraic inverse is also the geometric inverse. So if \mathbf{R}_{uvw} takes \mathbf{u} to \mathbf{x} , then \mathbf{R}_{uvw}^T takes \mathbf{x} to \mathbf{u} . The same should be true of \mathbf{v} and \mathbf{y} as we can confirm:

$$\mathbf{R}_{uvw}^T\mathbf{y} = \begin{bmatrix} x_u & x_v & x_w \\ y_u & y_v & y_w \\ z_u & z_v & z_w \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} x_v \\ y_v \\ z_v \end{bmatrix} = \mathbf{v}.$$

So we can always create rotation matrices from orthonormal bases.

If we wish to rotate about an arbitrary vector \mathbf{a} , we can form an orthonormal basis with $\mathbf{w} = \mathbf{a}$, rotate that basis to the canonical basis \mathbf{xyz} , rotate about the z -axis, and then rotate the canonical basis back to the \mathbf{uvw} basis. In matrix form, to rotate about the w -axis by an angle ϕ :

$$\begin{bmatrix} x_u & x_v & x_w \\ y_u & y_v & y_w \\ z_u & z_v & z_w \end{bmatrix} \begin{bmatrix} \cos \phi & -\sin \phi & 0 \\ \sin \phi & \cos \phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_u & y_u & z_u \\ x_v & y_v & z_v \\ x_w & y_w & z_w \end{bmatrix}.$$

Here we have \mathbf{w} a unit vector in the direction of \mathbf{a} (i.e., \mathbf{a} divided by its own length). But what are \mathbf{u} and \mathbf{v} ? A method to find reasonable \mathbf{u} and \mathbf{v} is given in Section 2.4.6.

If we have a rotation matrix and we wish to have the rotation in axis-angle form, we can compute the one real eigenvalue (which will be $\lambda = 1$), and the corresponding eigenvector is the axis of rotation. This is the one axis that is not changed by the rotation.

See Chapter 16 for a comparison of the few most-used ways to represent rotations, besides rotation matrices.

6.2.2 Transforming Normal Vectors

While most 3D vectors we use represent positions (offset vectors from the origin) or directions, such as where light comes from, some vectors represent *surface normals*. Surface normal vectors are perpendicular to the tangent plane of a surface. These normals do not transform the way we would like when the underlying surface is transformed. For example, if the points of a surface are transformed by a matrix \mathbf{M} , a vector \mathbf{t} that is tangent to the surface and is multiplied by \mathbf{M} will be tangent to the transformed surface. However, a surface normal vector \mathbf{n} that is transformed by \mathbf{M} may not be normal to the transformed surface (Figure 6.17).

We can derive a transform matrix \mathbf{N} which does take \mathbf{n} to a vector perpendicular to the transformed surface. One way to attack this issue is to note that a surface normal vector and a tangent vector are perpendicular, so their dot product is zero, which is expressed in matrix form as

$$\mathbf{n}^T \mathbf{t} = 0. \quad (6.4)$$

If we denote the desired transformed vectors as $\mathbf{t}_M = \mathbf{M}\mathbf{t}$ and $\mathbf{n}_N = \mathbf{N}\mathbf{n}$, our goal is to find \mathbf{N} such that $\mathbf{n}_N^T \mathbf{t}_M = 0$. We can find \mathbf{N} by some algebraic

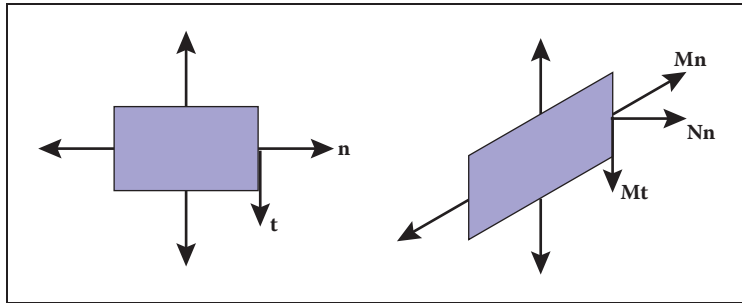


Figure 6.17. When a normal vector is transformed using the same matrix that transforms the points on an object, the resulting vector may not be perpendicular to the surface as is shown here for the sheared rectangle. The tangent vector, however, does transform to a vector tangent to the transformed surface.



tricks. First, we can sneak an identity matrix into the dot product, and then take advantage of $\mathbf{M}^{-1}\mathbf{M} = \mathbf{I}$:

$$\mathbf{n}^T \mathbf{t} = \mathbf{n}^T \mathbf{I} \mathbf{t} = \mathbf{n}^T \mathbf{M}^{-1} \mathbf{M} \mathbf{t} = 0.$$

Although the manipulations above don't obviously get us anywhere, note that we can add parentheses that make the above expression more obviously a dot product:

$$(\mathbf{n}^T \mathbf{M}^{-1}) (\mathbf{M} \mathbf{t}) = (\mathbf{n}^T \mathbf{M}^{-1}) \mathbf{t}_M = 0.$$

This means that the row vector that is perpendicular to \mathbf{t}_M is the left part of the expression above. This expression holds for any of the tangent vectors in the tangent plane. Since there is only one direction in 3D (and its opposite) that is perpendicular to all such tangent vectors, we know that the left part of the expression above must be the row vector expression for \mathbf{n}_N , i.e., it is \mathbf{n}_N^T , so this allows us to infer \mathbf{N} :

$$\mathbf{n}_N^T = \mathbf{n}^T \mathbf{M}^{-1},$$

so we can take the transpose of that to get

$$\mathbf{n}_N = (\mathbf{n}^T \mathbf{M}^{-1})^T = (\mathbf{M}^{-1})^T \mathbf{n}. \quad (6.5)$$

Therefore, we can see that the matrix which correctly transforms normal vectors so they remain normal is $\mathbf{N} = (\mathbf{M}^{-1})^T$, i.e., the transpose of the inverse matrix. Since this matrix may change the length of \mathbf{n} , we can multiply it by an arbitrary scalar and it will still produce \mathbf{n}_N with the right direction. Recall from Section 5.3 that the inverse of a matrix is the transpose of the cofactor matrix divided by the determinant. Because we don't care about the length of a normal vector, we can skip the division and find that for a 3×3 matrix,

$$\mathbf{N} = \begin{bmatrix} m_{11}^c & m_{12}^c & m_{13}^c \\ m_{21}^c & m_{22}^c & m_{23}^c \\ m_{31}^c & m_{32}^c & m_{33}^c \end{bmatrix}.$$

This assumes the element of \mathbf{M} in row i and column j is m_{ij} . So the full expression for \mathbf{N} is

$$\mathbf{N} = \begin{bmatrix} m_{22}m_{33} - m_{23}m_{32} & m_{23}m_{31} - m_{21}m_{33} & m_{21}m_{32} - m_{22}m_{31} \\ m_{13}m_{32} - m_{12}m_{33} & m_{11}m_{33} - m_{13}m_{31} & m_{12}m_{31} - m_{11}m_{32} \\ m_{12}m_{23} - m_{13}m_{22} & m_{13}m_{21} - m_{11}m_{23} & m_{11}m_{22} - m_{12}m_{21} \end{bmatrix}.$$



6.3 Translation and Affine Transformations

We have been looking at methods to change vectors using a matrix \mathbf{M} . In two dimensions, these transforms have the form,

$$\begin{aligned}x' &= m_{11}x + m_{12}y, \\y' &= m_{21}x + m_{22}y.\end{aligned}$$

We cannot use such transforms to *move* objects, only to scale and rotate them. In particular, the origin $(0, 0)$ always remains fixed under a linear transformation. To move, or *translate*, an object by shifting all its points the same amount, we need a transform of the form,

$$\begin{aligned}x' &= x + x_t, \\y' &= y + y_t.\end{aligned}$$

There is just no way to do that by multiplying (x, y) by a 2×2 matrix. One possibility for adding translation to our system of linear transformations is to simply associate a separate translation vector with each transformation matrix, letting the matrix take care of scaling and rotation and the vector take care of translation. This is perfectly feasible, but the bookkeeping is awkward and the rule for composing two transformations is not as simple and clean as with linear transformations.

Instead, we can use a clever trick to get a single matrix multiplication to do both operations together. The idea is simple: represent the point (x, y) by a 3D vector $[x \ y \ 1]^T$, and use 3×3 matrices of the form

$$\begin{bmatrix} m_{11} & m_{12} & x_t \\ m_{21} & m_{22} & y_t \\ 0 & 0 & 1 \end{bmatrix}.$$

The fixed third row serves to copy the 1 into the transformed vector, so that all vectors have a 1 in the last place, and the first two rows compute x' and y' as linear combinations of x , y , and 1:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & x_t \\ m_{21} & m_{22} & y_t \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11}x + m_{12}y + x_t \\ m_{21}x + m_{22}y + y_t \\ 1 \end{bmatrix}.$$

The single matrix implements a linear transformation followed by a translation! This kind of transformation is called an *affine transformation*, and this way of implementing affine transformations by adding an extra dimension is called *homogeneous coordinates* (Roberts, 1965; Riesenfeld, 1981; Penna & Patterson, 1986). Homogeneous coordinates not only clean up the code for transformations,



but this scheme also makes it obvious how to compose two affine transformations: simply multiply the matrices.

A problem with this new formalism arises when we need to transform vectors that are not supposed to be positions—they represent directions, or offsets between positions. Vectors that represent directions or offsets should not change when we translate an object. Fortunately, we can arrange for this by setting the third coordinate to zero:

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

If there is a scaling/rotation transformation in the upper-left 2×2 entries of the matrix, it will apply to the vector, but the translation still multiplies with the zero and is ignored. Furthermore, the zero is copied into the transformed vector, so direction vectors remain direction vectors after they are transformed.

This is exactly the behavior we want for vectors, so they fit smoothly into the system: the extra (third) coordinate will be either 1 or 0 depending on whether we are encoding a position or a direction. We actually do need to store the homogeneous coordinate so we can distinguish between locations and other vectors. For example,

$$\begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix} \text{ is a location} \quad \text{and} \quad \begin{bmatrix} 3 \\ 2 \\ 0 \end{bmatrix} \text{ is a displacement or direction.}$$

Later, when we do perspective viewing, we will see that it is useful to allow the homogeneous coordinate to take on values other than one or zero.

Homogeneous coordinates are used nearly universally to represent transformations in graphics systems. In particular, homogeneous coordinates underlie the design and operation of renderers implemented in graphics hardware. We will see in Chapter 7 that homogeneous coordinates also make it easy to draw scenes in perspective, another reason for their popularity.

Homogeneous coordinates can be considered just a clever way to handle the bookkeeping for translation, but there is also a different, geometric interpretation. The key observation is that when we do a 3D shear based on the z -coordinate we get this transform:

$$\begin{bmatrix} 1 & 0 & x_t \\ 0 & 1 & y_t \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x + x_t z \\ y + y_t z \\ z \end{bmatrix}.$$

Note that this almost has the form we want in x and y for a 2D translation, but has a z hanging around that doesn't have a meaning in 2D. Now comes the key

This gives an explanation for the name “homogeneous:” translation, rotation, and scaling of positions and directions all fit into a single system.

Homogeneous coordinates are also ubiquitous in computer vision.

decision: we will add a coordinate $z = 1$ to all 2D locations. This gives us

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

By associating a ($z = 1$)-coordinate with all 2D points, we now can encode translations into matrix form. For example, to first translate in 2D by (x_t, y_t) and then rotate by angle ϕ we would use the matrix

$$\mathbf{M} = \begin{bmatrix} \cos \phi & -\sin \phi & 0 \\ \sin \phi & \cos \phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & x_t \\ 0 & 1 & y_t \\ 0 & 0 & 1 \end{bmatrix}.$$

Note that the 2D rotation matrix is now 3×3 with zeros in the “translation slots.” With this type of formalism, which uses shears along $z = 1$ to encode translations, we can represent any number of 2D shears, 2D rotations, and 2D translations as one composite 3D matrix. The bottom row of that matrix will always be $(0, 0, 1)$, so we don’t really have to store it. We just need to remember it is there when we multiply two matrices together.

In 3D, the same technique works: we can add a fourth coordinate, a homogeneous coordinate, and then we have translations:

$$\begin{bmatrix} 1 & 0 & 0 & x_t \\ 0 & 1 & 0 & y_t \\ 0 & 0 & 1 & z_t \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x + x_t \\ y + y_t \\ z + z_t \\ 1 \end{bmatrix}.$$

Again, for a direction vector, the fourth coordinate is zero and the vector is thus unaffected by translations.

Example (Windowing transformations). Often in graphics we need to create a transform matrix that takes points in the rectangle $[x_l, x_h] \times [y_l, y_h]$ to the rectangle $[x'_l, x'_h] \times [y'_l, y'_h]$. This can be accomplished with a single scale and translate in sequence. However, it is more intuitive to create the transform from a sequence of three operations (Figure 6.18):

1. Move the point (x_l, y_l) to the origin.
2. Scale the rectangle to be the same size as the target rectangle.
3. Move the origin to point (x'_l, y'_l) .

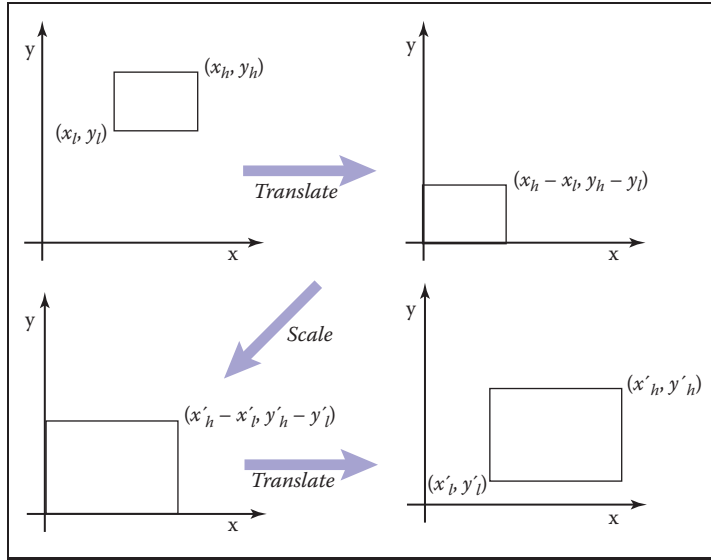


Figure 6.18. To take one rectangle (window) to the other, we first shift the lower-left corner to the origin, then scale it to the new size, and then move the origin to the lower-left corner of the target rectangle.

Remembering that the right-hand matrix is applied first, we can write

$$\begin{aligned}
 \text{window} &= \text{translate}(x'_l, y'_l) \text{ scale} \left(\frac{x'_h - x'_l}{x_h - x_l}, \frac{y'_h - y'_l}{y_h - y_l} \right) \text{translate}(-x_l, -y_l) \\
 &= \begin{bmatrix} 1 & 0 & x'_l \\ 0 & 1 & y'_l \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{x'_h - x'_l}{x_h - x_l} & 0 & 0 \\ 0 & \frac{y'_h - y'_l}{y_h - y_l} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_l \\ 0 & 1 & -y_l \\ 0 & 0 & 1 \end{bmatrix} \\
 &= \begin{bmatrix} \frac{x'_h - x'_l}{x_h - x_l} & 0 & \frac{x'_l x_h - x'_h x_l}{x_h - x_l} \\ 0 & \frac{y'_h - y'_l}{y_h - y_l} & \frac{y'_l y_h - y'_h y_l}{y_h - y_l} \\ 0 & 0 & 1 \end{bmatrix}. \tag{6.6}
 \end{aligned}$$

It is perhaps not surprising to some readers that the resulting matrix has the form it does, but the constructive process with the three matrices leaves no doubt as to the correctness of the result.

An exactly analogous construction can be used to define a 3D windowing transformation, which maps the box $[x_l, x_h] \times [y_l, y_h] \times [z_l, z_h]$ to the box

$[x'_l, x'_h] \times [y'_l, y'_h] \times [z'_l, z'_h]$:

$$\begin{bmatrix} \frac{x'_h - x'_l}{x_h - x_l} & 0 & 0 & \frac{x'_l x_h - x'_h x_l}{x_h - x_l} \\ 0 & \frac{y'_h - y'_l}{y_h - y_l} & 0 & \frac{y'_l y_h - y'_h y_l}{y_h - y_l} \\ 0 & 0 & \frac{z'_h - z'_l}{z_h - z_l} & \frac{z'_l z_h - z'_h z_l}{z_h - z_l} \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (6.7)$$

It is interesting to note that if we multiply an arbitrary matrix composed of scales, shears, and rotations with a simple translation (translation comes second), we get

$$\begin{bmatrix} 1 & 0 & 0 & x_t \\ 0 & 1 & 0 & y_t \\ 0 & 0 & 1 & z_t \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & x_t \\ a_{21} & a_{22} & a_{23} & y_t \\ a_{31} & a_{32} & a_{33} & z_t \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Thus, we can look at any matrix and think of it as a scaling/rotation part and a translation part because the components are nicely separated from each other.

An important class of transforms are *rigid-body* transforms. These are composed only of translations and rotations, so they have no stretching or shrinking of the objects. Such transforms will have a pure rotation for the a_{ij} above.

6.4 Inverses of Transformation Matrices

While we can always invert a matrix algebraically, we can use geometry if we know what the transform does. For example, the inverse of $\text{scale}(s_x, s_y, s_z)$ is $\text{scale}(1/s_x, 1/s_y, 1/s_z)$. The inverse of a rotation is the same rotation with the opposite sign on the angle. The inverse of a translation is a translation in the opposite direction. If we have a series of matrices $\mathbf{M} = \mathbf{M}_1 \mathbf{M}_2 \cdots \mathbf{M}_n$ then $\mathbf{M}^{-1} = \mathbf{M}_n^{-1} \cdots \mathbf{M}_2^{-1} \mathbf{M}_1^{-1}$.

Also, certain types of transformation matrices are easy to invert. We've already mentioned scales, which are diagonal matrices; the second important example is rotations, which are orthogonal matrices. Recall (Section 5.2.4) that the inverse of an orthogonal matrix is its transpose. This makes it easy to invert rotations and rigid body transformations (see Exercise 6). Also, it's useful to know that a matrix with $[0 \ 0 \ 0 \ 1]$ in the bottom row has an inverse that also has $[0 \ 0 \ 0 \ 1]$ in the bottom row (see Exercise 7).

Interestingly, we can use SVD to invert a matrix as well. Since we know that any matrix can be decomposed into a rotation times a scale times a rotation,



inversion is straightforward. For example, in 3D we have

$$\mathbf{M} = \mathbf{R}_1 \text{scale}(\sigma_1, \sigma_2, \sigma_3) \mathbf{R}_2,$$

and from the rules above it follows easily that

$$\mathbf{M}^{-1} = \mathbf{R}_2^T \text{scale}(1/\sigma_1, 1/\sigma_2, 1/\sigma_3) \mathbf{R}_1^T.$$

6.5 Coordinate Transformations

All of the previous discussion has been in terms of using transformation matrices to move points around. We can also think of them as simply changing the coordinate system in which the point is represented. For example, in Figure 6.19, we see two ways to visualize a movement. In different contexts, either interpretation may be more suitable.

For example, a driving game may have a model of a city and a model of a car. If the player is presented with a view out the windshield, objects inside the car are always drawn in the same place on the screen, while the streets and buildings appear to move backward as the player drives. On each frame, we apply a transformation to these objects that moves them farther back than on the previous frame. One way to think of this operation is simply that it moves the buildings backward; another way to think of it is that the buildings are staying put but the coordinate system in which we want to draw them—which is attached to the car—is moving. In the second interpretation, the transformation is changing

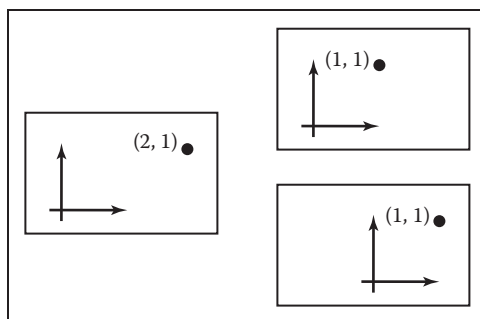


Figure 6.19. The point (2,1) has a transform “translate by (-1,0)” applied to it. On the top right is our mental image if we view this transformation as a physical movement, and on the bottom right is our mental image if we view it as a change of coordinates (a movement of the origin in this case). The artificial boundary is just an artifice, and the relative position of the axes and the point are the same in either case.

the coordinates of the city geometry, expressing them as coordinates in the car's coordinate system. Both ways will lead to exactly the same matrix that is applied to the geometry outside the car.

If the game also supports an overhead view to show where the car is in the city, the buildings and streets need to be drawn in fixed positions while the car needs to move from frame to frame. The same two interpretations apply: we can think of the changing transformation as moving the car from its canonical position to its current location in the world; or we can think of the transformation as simply changing the coordinates of the car's geometry, which is originally expressed in terms of a coordinate system attached to the car, to express them instead in a coordinate system fixed relative to the city. The change-of-coordinates interpretation makes it clear that the matrices used in these two modes (city-to-car coordinate change vs. car-to-city coordinate change) are inverses of one another.

The idea of changing coordinate systems is much like the idea of type conversions in programming. Before we can add a floating-point number to an integer, we need to convert the integer to floating point or the floating-point number to an integer, depending on our needs, so that the types match. And before we can draw the city and the car together, we need to convert the city to car coordinates or the car to city coordinates, depending on our needs, so that the coordinates match.

When managing multiple coordinate systems, it's easy to get confused and wind up with objects in the wrong coordinates, causing them to show up in unexpected places. But with systematic thinking about transformations between coordinate systems, you can reliably get the transformations right.

Geometrically, a coordinate system, or coordinate *frame*, consists of an origin and a basis—a set of three vectors. Orthonormal bases are so convenient that we'll normally assume frames are orthonormal unless otherwise specified. In a frame with origin \mathbf{p} and basis $\{\mathbf{u}, \mathbf{v}, \mathbf{w}\}$, the coordinates (u, v, w) describe the point

$$\mathbf{p} + u\mathbf{u} + v\mathbf{v} + w\mathbf{w}.$$

When we store these vectors in the computer, they need to be represented in terms of some coordinate system. To get things started, we have to designate some canonical coordinate system, often called “global” or “world” coordinates, which is used to describe all other systems. In the city example, we might adopt the street grid and use the convention that the x -axis points along Main Street, the y -axis points up, and the z -axis points along Central Avenue. Then when we write the origin and basis of the car frame in terms of these coordinates it is clear what we mean.

In 2D our convention is to use the point \mathbf{o} for the origin, and \mathbf{x} and \mathbf{y} for the right-handed orthonormal basis vectors \mathbf{x} and \mathbf{y} (Figure 6.20).

In 2D, of course, there are two basis vectors.

In 2D, right-handed means \mathbf{y} is counterclockwise from \mathbf{x} .

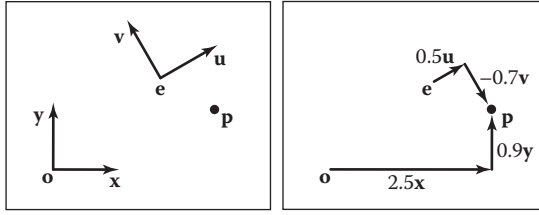


Figure 6.20. The point \mathbf{p} can be represented in terms of either coordinate system.

Another coordinate system might have an origin \mathbf{e} and right-handed orthonormal basis vectors \mathbf{u} and \mathbf{v} . Note that typically the canonical data \mathbf{o} , \mathbf{x} , and \mathbf{y} are never stored explicitly. They are the frame-of-reference for all other coordinate systems. In that coordinate system, we often write down the location of \mathbf{p} as an ordered pair, which is shorthand for a full vector expression:

$$\mathbf{p} = (x_p, y_p) \equiv \mathbf{o} + x_p \mathbf{x} + y_p \mathbf{y}.$$

For example, in Figure 6.20, $(x_p, y_p) = (2.5, 0.9)$. Note that the pair (x_p, y_p) implicitly assumes the origin \mathbf{o} . Similarly, we can express \mathbf{p} in terms of another equation:

$$\mathbf{p} = (u_p, v_p) \equiv \mathbf{e} + u_p \mathbf{u} + v_p \mathbf{v}.$$

In Figure 6.20, this has $(u_p, v_p) = (0.5, -0.7)$. Again, the origin \mathbf{e} is left as an implicit part of the coordinate system associated with \mathbf{u} and \mathbf{v} .

We can express this same relationship using matrix machinery, like this:

$$\begin{bmatrix} x_p \\ y_p \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_e \\ 0 & 1 & y_e \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_u & x_v & 0 \\ y_u & y_v & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_p \\ v_p \\ 1 \end{bmatrix} = \begin{bmatrix} x_u & x_v & x_e \\ y_u & y_v & y_e \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_p \\ v_p \\ 1 \end{bmatrix}.$$

Note that this assumes we have the point \mathbf{e} and vectors \mathbf{u} and \mathbf{v} stored in canonical coordinates; the (x, y) -coordinate system is the first among equals. In terms of the basic types of transformations we've discussed in this chapter, this is a rotation (involving \mathbf{u} and \mathbf{v}) followed by a translation (involving \mathbf{e}). Looking at the matrix for the rotation and translation together, you can see it's very easy to write down: we just put \mathbf{u} , \mathbf{v} , and \mathbf{e} into the columns of a matrix, with the usual $[0 \ 0 \ 1]$ in the third row. To make this even clearer we can write the matrix like this:

$$\mathbf{p}_{xy} = \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{e} \\ 0 & 0 & 1 \end{bmatrix} \mathbf{p}_{uv}.$$

We call this matrix the *frame-to-canonical* matrix for the (u, v) frame. It takes points expressed in the (u, v) frame and converts them to the same points expressed in the canonical frame.

The name “frame-to-canonical” is based on thinking about changing the coordinates of a vector from one system to another. Thinking in terms of moving vectors around, the frame-to-canonical matrix maps the canonical frame to the (u, v) frame.