Chapter 1

Linear Algebra Foundations

Vectors and Basis

What is the location of the point above? You may say simply that it is on the fiirst page of the first chapter of this book or that it's located somewhere above the first paragraph of that page. You would not be incorrect in either of those observations, but if you wanted to be more precise, you would have to begin to specify some sort of coordinates.

For example, you might say that the point is 2 inches from the top of the page and 4 inches from its left edge. Alternatively, you could pick the bottom of the page and its right edge as your reference and measure the distance from there. Maybe you would use centimeters or millimeters instead of inches if you lived in a place that preferred the metric system. You could even be so impractical as to pick a reference point outside of the physical bounds of this book, like the corner of the room that you are in, and measure the distance to the point on the page from there, although you would have to measure the distance in 3 dimensions.

In any case, you will need the same 4 things: a point to measure from, the direction in each dimension to measure in, the unit of measure you are using and the actual measure in each direction.

In game programming we deal a lot with these sorts of spatial coordinates. For example, we might want to describe to our graphics card where exactly we want to draw a game character or some other graphic. Or we might want to calculate the coordinates of a game character and an item and to see if they are close enough for the character to pick it up. And all of this inivolves being able to describe mathematically the same things we would need in the real world: the point to measure from, the directions and units to measure along and the measure itself.

In mathematics, we do this using the concept of vectors. A vector is simply a scalar value that describes a measure in a certain direction.

Consider this example on the number line (Fig. 1).

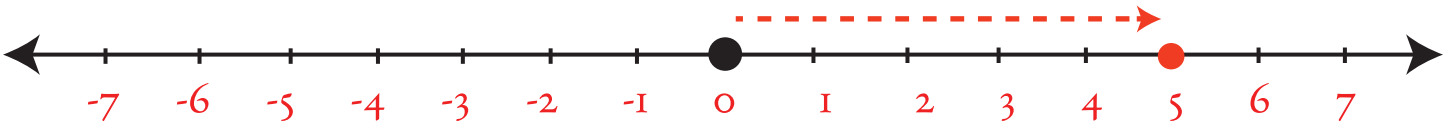


Fig. 1

You can think of the number 5 as a coordinate on the number line, or you can think of it as a vector, measuring 5 units in the positive direction starting from 0, ending up at number 5 on the number line. A vector can be described as having a direction (another vector with magnitude of 1, called a unit vector) and magnitude. In the case above, the vector's magnitude is 5 and its direction is 1. A vector with the same magnitude and a direction of -1 (-5) would end up at -5 on the number line relative to 0.

The "relative to 0" in our example is important, since a vector, being simply a measure of distance, always points to a relative value. The same vector relative to the number 2 on the number line would end up at the number 7 (or -3 if its direction was -1).

We can also think of the number line as a single-dimensional space with the basis vector 1 and our coordinate vector as being multiplied with that basis vector. You can think of the basis vector as defining how many steps we have to take and in what direction for each unit of measure. The default (standard) basis for a single dmension is 1.

So we can say that to get to a certain point in that space we add to an origin point 0, the basis vector b times coodinate vector c, or $o + bc$. (i.e. 0 + 1 * 5 in our example). Just like what we need to find any coordinate in the real world, except the basis vector is doing the job of both direction and unit of measure.

If we change our basis vector, the resulting coordinate of our vector with respect to the standard basis will also change. So for example, with a basis vector 2, the resulting coordinate for coordnate vector 5 relative to 0 will be 10 (0 + 2 * 5).

Vectors can be N-dimensional, containing a scalar value for each dimension. So we can have a two-dimensional vector [5,3] or a three-dimensional vector [5, 3, 2], etc.

Just like before, we can define a vector space within N dimensions using N-dimensional basis vectors. So a 2 dimensional space might be defined with the standard basis vectors [1, 0] and [0, 1]. To get to our coordinate, we do the same thing as in one dimension, only now we need to multiply each basis vector with the step in each dimension and add them. So for a coordinate vector [5,3] with respect to the standard basis, we multply [1,0] by 5 ([5,0]), multiply [0,1] by 3 ([0, 3]) and add them together to get [5,3]. (Fig. 2)

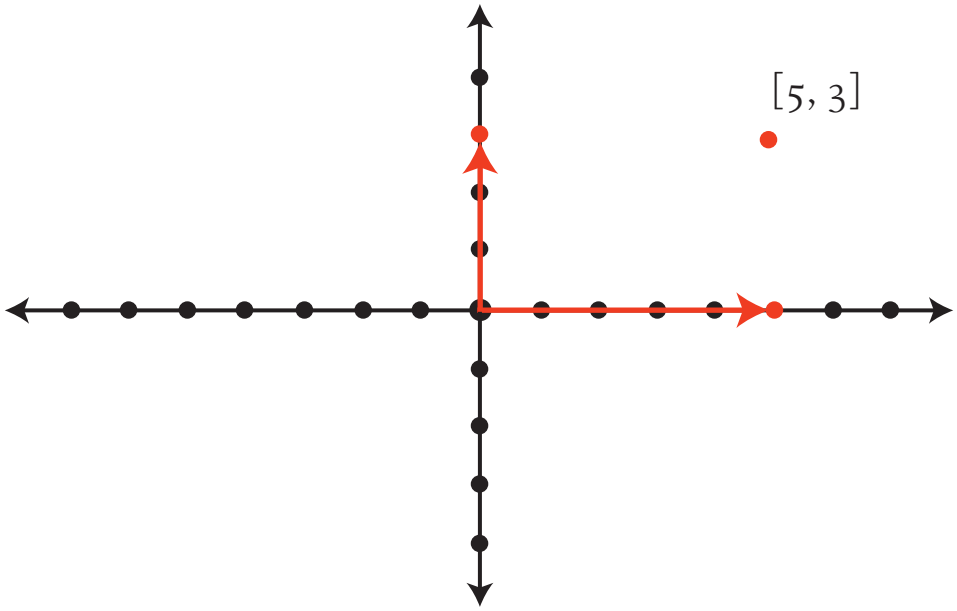


Fig. 2

We can write this neatly as a matrix multiplication, putting our basis vectors as col-

$$\begin{matrix} b_1 & b_2 \\ | & | \end{matrix}$$
$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} 5 \\ 3 \end{bmatrix} = \begin{bmatrix} 1 \times 5 + 0 \times 3 \\ 0 \times 5 + 1 \times 3 \end{bmatrix} = \begin{bmatrix} 5 \\ 3 \end{bmatrix}$$

Fig. 3

umns in a NxN dimensional matrix and multiplying it with our coordinate vector (Fig.3).

Our resulting vector here is the same as our coordinate vector [5,3]. This is because our basis is the same as the standard basis for our 2 dimensional space, resulting in an identity matrix (the same thing as multiplying by 1 in one dimension).

Let's see what happens if we modify our basis vectors.

Linear Transformations

So far we have not changed our basis vectors from the standard basis. But by modifying the basis vectors we can create linear transformations of coordinates with respect to the standard basis.

The simplest linear transformation we can do is scaling the basis vectors by multiplying them with some scalar value. Just like in our single-dimensional example where we changed our basis vector of 1 to 2 and changed the final coordinate, we can scale each of the basis vector in our matrix.

$$\begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix} \times \begin{bmatrix} 5 \\ 3 \end{bmatrix} = \begin{bmatrix} 2 \times 5 + 0 \times 3 \\ 0 \times 5 + 3 \times 3 \end{bmatrix} = \begin{bmatrix} 10 \\ 9 \end{bmatrix}$$

Fig. 4

By scaling our basis vectors by 2 and 3 respectively (Fig. 4), we have scaled the resulting coordinate. This sort of transformation might seem simple, but it is very useful in computer graphics. Consider a simple drawing made from a number of coordinate points (Fig. 5).

We can make the drawing larger or smaller relative to the standard basis by changing the basis vectors the coordinates are mapped on. Or even squash and stretch the drawing if we scale the basis vectors in a non-uniform way (Fig. 6).

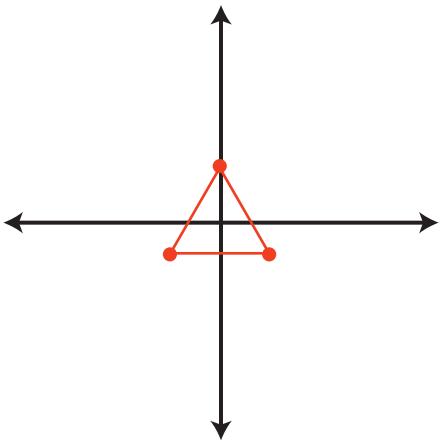


Fig. 5

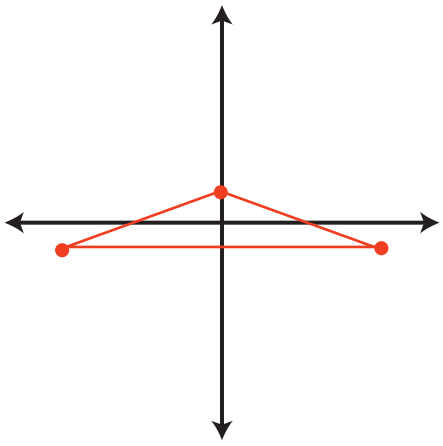
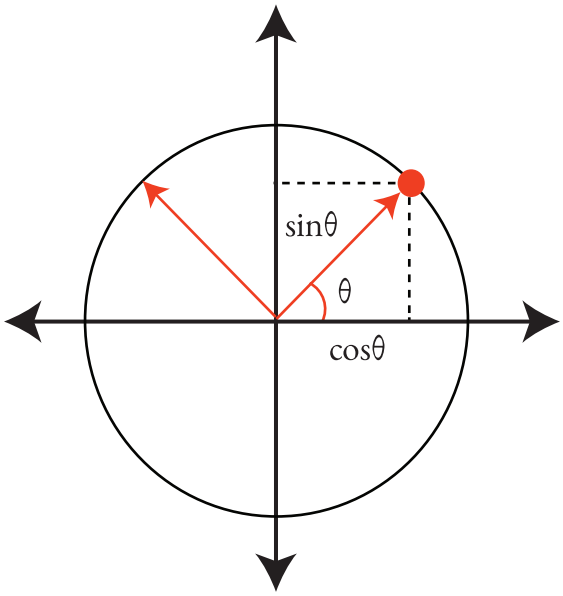


Fig. 6



Let's try another kind of linear transformation: *rotation*.

We can use the sine and cosine of an angle to rotate the basis vectors. Remember that cosine of an angle is the horizontal position of the corresponding point on the unit cricle (the point one unit away from origin in the direction of the angle) and the sine of an angle is the vertical position of that point.

For a given angle θ, the vector [sin θ, cos θ] is thus a unit vector pointing in the direction of that angle. If we use that unit vector as our first basis vector and the vector orthogonal to it as the second basis vector[1], we will have in effect rotated our basis vectors by that angle.

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \cos(\theta) \times x - \sin(\theta) \times y \\ \sin(\theta) \times x + \cos(\theta) \times y \end{bmatrix}$$

Now all coordinate vectors mapped on these basis vectors will be rotated around origin with respect to the standard basis by the angle

In this 2-dimensional example we're rotating the X and Y axes, but in 3 dimensions you can rotate around any of the 3 axes by rotating the Y and Z or X and Z vectors.

$$\begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix}$$

---

[1] To get a vector orthogonal to a 2d vector, we can switch its X and Y components and inverse one of them. Depending on which component we inverse, we get the vector pointing to the right or left of the original vector. In this case, we invert the first component of the resulting vector.

Affine Transformations and Homogeneous Coordinates

We learned how to use a matrix of basis vectors to represent scale and rotational linear transformations, but there is one more type of transformation that we need to be able to manipulate a set of points in any way we want: *translation*.

Translation means simply adding some value on each axis to move the coordinate by that amount. It sounds simple, but the trick is that we want to use the same matrix/coordinate multiplication system to do all of our transformations. Currently with the linear transformation matrix, this is impossible. We can scale and rotate the basis vectors, but no matter what, their transformations will always be relative to center. How do we do it then? Enter homogeneous coordinates.

The idea behind homogeneous coordinates is simple, but sometimes difficult to visualize. Let's go back to a one-dimensional example to illustrate the concept as simply as possible.

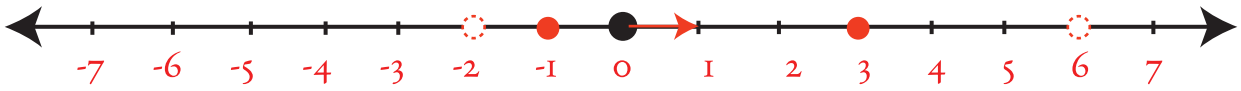


Fig. 8

Let's say we have two one-dimensional coordinates: 3 and -1 mapped to a basis vector of 1. We can't rotate them in one-dimension, but we can scale them by scaling the basis vector. When we scale the basis vector, both coordinates move away from origin, for example 3 moves to 6 and -1 to -2 when scaled by 2.(Fig.8). But what can we do if we wanted to move them both in the same direction and still use multiplication? We can't do it with our basis vector no matter what we do with it.

But maybe we can do it in the next dimension up! Let's move into 2d and add a second basis vector [0, 1], orthogonal to our original basis vector of 1, which now becomes [1, 0] (since it also has to be described in 2 dimensions), to create our standard 2d basis. Now, let's project our points into this new dimension at position 1. We simply set the sec-

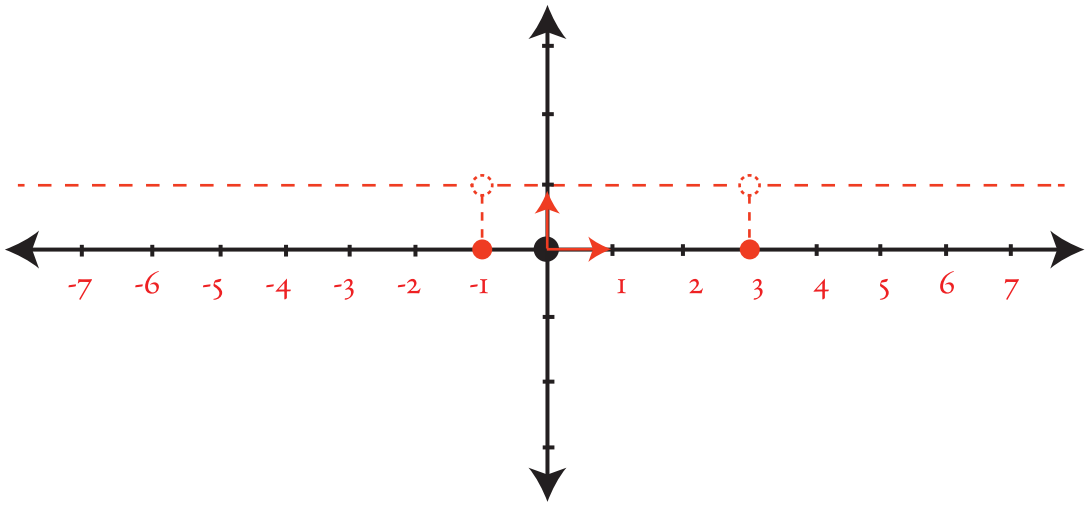


Fig. 9

ond dimensioin value to 1, so our points become [-1, 1] and [3, 1] (Fig. 9).

Right now, our second basis vector [0,1] is pointing straight up, so for every unit step in that direction, we go 0 units on the X axis and 1 unit on the Y axis. But what if we skewed our second basis vector by changing its X axis component? (Fig. 10).

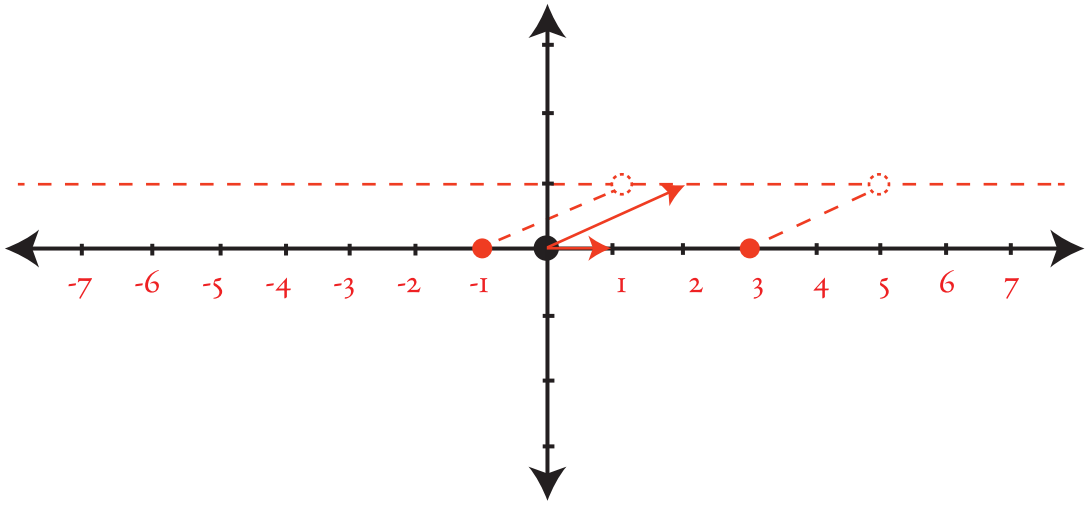


Fig. 10

Let's say we changed the X axis component our our second basis vector to 2. Now for every unit on that second basis vector, we go 2 units on the X axis!

$$\begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \times x + 2 \times 1 \\ 0 \times x + 1 \times 1 \end{bmatrix} = \begin{bmatrix} x + 2 \\ 1 \end{bmatrix}$$

Now all of our coordinates that have been multplied with this skewed basis will be translated by 2 units. All we have to do now is project the points back down into our original dimension (simply ignore the second component of 1)!

The same process works in any number of dimensions. For any n-dimensional space, you can project its coordinates into the n+1 dimension and skew that new basis vector in any of the n dimensions. For example, points on a 2D plane can be projected on a 3rd basis vector into 3D, then that vector can be skewed in the first two dimensions to translate the coordinates on the 2D plane (Fig.11).

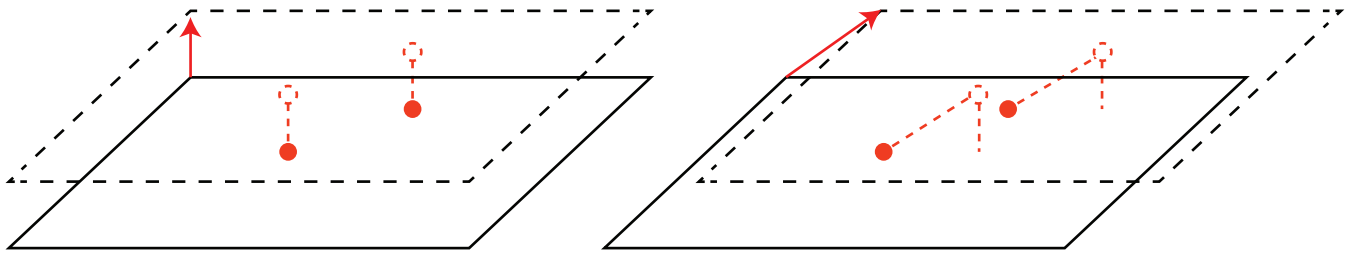


Fig. 11

It's harder to visualize this in dimensions higher than 2, but mathematically the process is exactly the same.

We can combine this homogeneous coordinate transformation with the linear transformations we discussed earlier since the linear transformatoin matrix is a subset of this larger matrix and encode rotation, scale and translation in a single matrix.

This kind of transformation that combines a purely linear transformation with a translation is called an *affiine transformation*[1] and the matrix in n+1 dimensions of our coordinate system that combines a linear transformation with the translation using homogeneous coordinate is called an *affine transformation matrix*.

Since we will be working primarily with 3D coordinates, let's familiarize ourselves with the 3D affine transformation matrices we will be dealing with. Let's start with the identity 3D affine transformation matrix and a 3D coordinate in homogeneous coordinates. (Fig. 12)

linear part

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

3D homogeneous coordinate with 1 as its w component

homogeneous coordinate basis

[1] Every linear transformation is affine, but not every affine transformation is linear, since the origin point is not always preserved in an affine transformation.

Let's review the different kinds of transformations we discussed and look at their corresponding 3D affine transformation matrices.

$$\begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scale by [S_x, S_y, S_z]

$$\begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Translate by [T_x, T_y, T_z]

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotate around Z-axis by angle θ

$$\begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotate around Y-axis by angle θ

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotate around X-axis by angle θ

We now know how to create matrices for each individual transformation, but what if we wanted to combine them into a single matrix? We can premultiply the scale and rotation basis vector in the linear part and combine it with the translation vector in the final matrix.

Another option is to create a matrix for each transformation we want to do and multiply them together to create the final matrix that combines all of them together.

$$M = M_t * M_s * M_r$$

However, we have to be careful when multiplying our matrices as transformation matrix multiplication is non-commutative. That is, the order of operations matters and we can get completely different results by changing the order in which the matrices are multiplied. Which brings us to our next topic.

Reference frames and matrix multiplication

So we mentioned that multiplying two matrices combines their transformations, but that the order of the multiplication matters. Let's look at why that is the case.

When you multiply two transformation matrices together, you are applying the transformation of the first matrix to the basis vectors of the second matrix the same way that you would apply it to a coordinate vector. So, you are mapping the basis vectors of the second matrix on the basis vectors of the first matrix.

Consider two 2d affine transformation matrices, one that scales by [2,3] and one that translates by [4,5]. If we multiply the translation matrix with the scale matrix, we combine the scale of the scale matrix with the translation of the translation matrix, but the translation is not affected by the scale.

$$\begin{bmatrix} 1 & 0 & 4 \\ 0 & 1 & 5 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1\times 2+0\times 0+4\times 0 & 1\times 0+0\times 3+4\times 0 & 1\times 0+0\times 0+4\times 1 \\ 0\times 2+1\times 0+5\times 0 & 0\times 0+1\times 3+5\times 0 & 0\times 0+1\times 0+5\times 1 \\ 0\times 2+0\times 0+1\times 0 & 0\times 0+0\times 3+1\times 0 & 0\times 0+0\times 0+1\times 1 \end{bmatrix}$$

$$= \begin{bmatrix} 2 & 0 & 4 \\ 0 & 3 & 5 \\ 0 & 0 & 1 \end{bmatrix}$$

However, if we reverse the order of the multplication, we are mapping the basis vectors of the translation onto the scaled vectors of the first matrix, therefore also scaling the X and Y components of the third basis vector, thus scaling our translation. So the resulting matrix contains both the scale and translation like before, but now the translation is scaled as well.

$$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 4 \\ 0 & 1 & 5 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 2\times 1+0\times 0+0\times 0 & 2\times 0+0\times 1+0\times 0 & 2\times 4+0\times 5+0\times 1 \\ 0\times 1+3\times 0+0\times 0 & 0\times 0+3\times 1+0\times 0 & 0\times 4+3\times 5+0\times 1 \\ 0\times 1+0\times 0+1\times 0 & 0\times 0+0\times 1+1\times 0 & 0\times 4+0\times 5+1\times 1 \end{bmatrix}$$

$$= \begin{bmatrix} 2 & 0 & 8 \\ 0 & 3 & 15 \\ 0 & 0 & 1 \end{bmatrix}$$

Another way of thinking about this is that all matrix transformations in a chain of matrix multiplicatoin are relative to the coordinate system of the previous matrix. For example, if we have a matrix containing a rotation and a matrix containing a translation and we multiply the rotation matrix with the translation matrix in that order, we are translating along the rotated basis vectors of the fiirst matrix (Fig. 12).

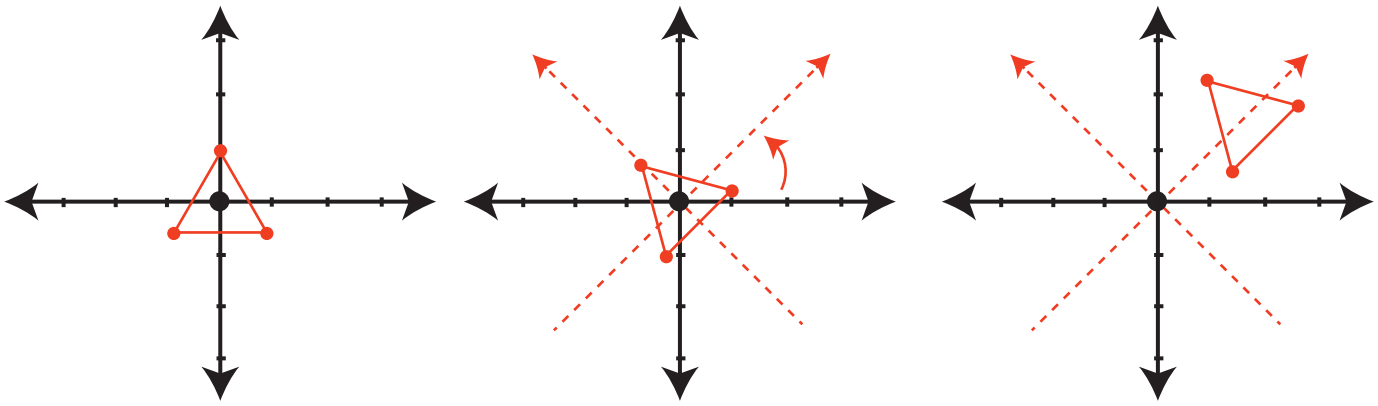


Fig. 12

Whereas if we multiply them with the translation matrix being fiirst, we are rotating relative to the translated coordinate system (Fig. 13). As you can see, it results in a completely different position of our coordinates.

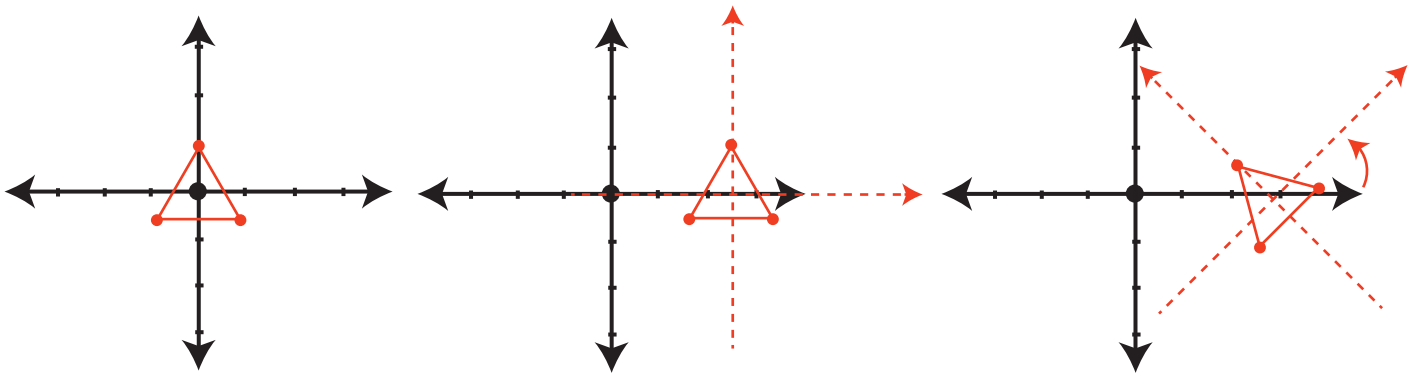


Fig. 13

So, when creating a combined transformation matrix by multiplyng translation, scale and rotation matrices, the order should always have the translation matrix fiirst, so that the translation is not scaled or rotated by the other matrices.

We can also create herarchies of transformations by multiplying chains of transformation matrices in order from the root parent out to each child in the hierarchy. Consider this example of a simple solar system simulation.

A planet is orbiting around a sun and a moon is orbiting around the planet. We have to calculate the position of the moon relative to the sun, which is at the center of our coordinate

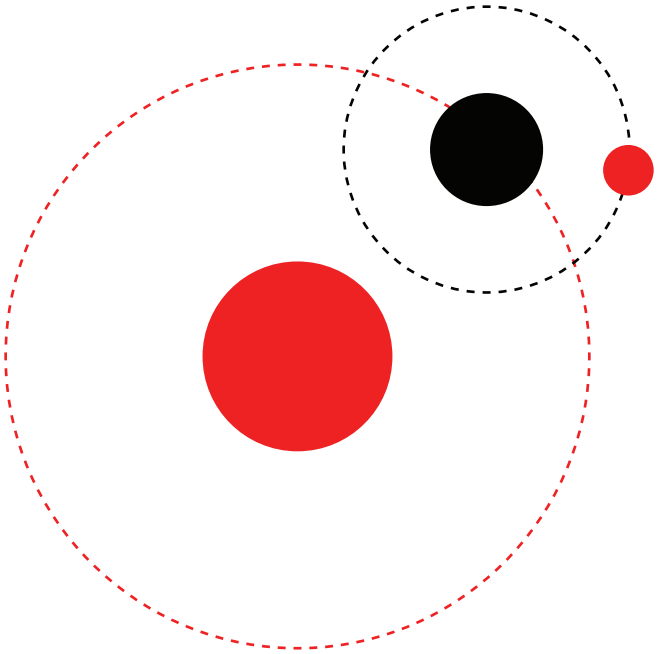


Fig. 14

system so that we can draw it in its proper place (Fig. 14).

We can easily calculate the transformation matrix of the planet based on its rotation around the center of the sun. Let's just say it's a transformation matrix using the sine and cosine of the angle of its current orbit (remember the unit circle!).

But the moon is a bit more complicated. We need to figure out its transformation matrix relative to the sun, which is the center of the standard coordinate system in our theoretical example.

But remember that all transformations are relative, so we can simply calculate the position of the moon relative to the planet the same way we calculated the planet's position relative to the sun and multiply the planet's transformation matrix with the moon's transformation matrix. The resulting matrix describes the moons position relative to the center of the coordinate.

If we had some other object oribiting the moon, we could calculate its transformation matrix relative to the moon, then multiply the moon's matrix with that matrix and get the position of that object relative to the sun, etc.

This is the basic principle behind transformation hierarchies. In games, when we attach objects to other objects, like a sword to a character's hand, we simply multiply an object's transformation matrix by its parent object's matrix.