

# Chapter 1

## Three Dimensional Geometry I

In computer graphics, we are constantly describing both the shapes and positions of three dimensional objects. For example, one simple way to describe a 3D object is to approximate its shape as a mesh of triangles. Each triangle is defined by three vertices, and the positions of each of these vertices will have to be described by three coordinates  $[x, y, z]^t$ .

When modeling a 3D environment, one often needs to apply various transformations to these 3D objects. We may wish to translate or rotate the object to position some object in a scene. Alternatively one may wish to apply a non-rigid transformation to object; this allows us to change its shape. For example, one may wish to create an egg shape by starting with a sphere, and squashing it in one direction.

When one is creating an animation, one very often wants to have the position and orientation of some object to be continually changing over time. In this case, one wants to apply a time-varying rigid transformation.

To create a 2D image of the 3D environment, one places a “virtual camera” at some position in the scene. a projection operation is then applied to turn the 3d scene description into a 2d image. This too needs to be represented as some type of transformation.

It is for these reasons, it is essential that we understand how to manipulate the coordinates that describe 3D objects. We must also understand how to manipulate and apply transformations. We will pay special attention to the role that the order of transformation applications play. When we study things like rotations, we will pay special attention to the role of the coordinate system with respect to which we will be applying the rotation.

In this chapter, we will begin by looking at linear transformations; these can represent rotations, but they can also represent scaling, and shearing. Later we will investigate affine transformations, which add the ability to translate objects. Finally we will look at projective transformations, which are essential in the imaging process.

## 1.1 Geometric data types

Imagine some specific geometric point in the real world. This point can be represented with three real numbers,

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

which we will call a coordinate vector. The numbers specify the position of the point with respect to some agreed upon coordinate system. This agreed upon coordinate system has some agreed upon origin point, and also has three agreed upon directions. If we were to change the agreed upon coordinate system, then we would need a different set of numbers, a different coordinate vector, to describe the same point. So in order to specify the location of an actual point, we need both a coordinate system, as well as a coordinate vector. This suggests that we really need to be careful in distinguishing the following data types. Coordinate systems, coordinate vectors, and geometric points.

We will start with four types of data types each with their own notation.

- A geometric point will be notated as  $\tilde{p}$  with a tilde above the letter.
- A vector will be notated as  $\vec{v}$  with an arrow above the letter. We will discuss in detail the difference between vectors and points in detail soon. The main difference is that points represent places while vectors represent the motion needed to move from point to point.
- A coordinate vector will be represented as  $\mathbf{c}$  with a bold letter.
- A coordinate system will be notated as  $\vec{s}^t$  with a bold letter, arrow, and superscript “t”. There is actually two kinds of coordinate systems. A “basis” is used to describe vectors, while a “frame” is used to describe points. We use the same notation for all coordinate systems and let the context distinguish the type.

## 1.2 Vectors, Coordinate vectors, and bases

Let us start by clearly distinguishing between vectors and coordinate vectors. A vector is an abstract geometric entity that represents motion between two points in the world. A coordinate vector is a set of numbers used to specify a vector once we have agreed upon a coordinate system.

More formally, a vector space  $V$  is some set of elements  $\vec{v}$  that satisfy certain rules. In particular one needs to define an addition operation that takes two vectors and maps it to a third vector. One also needs to define the operation of multiplying a scalar (real number) times a vector to get back another vector. To be a valid vector space, a bunch of other rules must be satisfied. These include things like the addition operation has

to be associative and commutative. One of the important rules is that scalar multiplies must distribute across vector adds

$$\alpha(\vec{v} + \vec{w}) = \alpha\vec{v} + \alpha\vec{w}$$

The set  $V$  can be anything, motion between points, polynomial expressions, farm animals, as long as we have defined addition and scalar multiplication on the set elements, and these operations obey the necessary rules. In particular, in our context we will think of vectors as actual motion between geometric points. We will not think of a vector as a set of 3 numbers.

A coordinate system, or basis, will just be a set of vectors that we can use to get to all of the vectors using the vector operations.

More formally, we say that a set of vectors  $\vec{b}_1 \dots \vec{b}_n$  is linearly dependent if there exists scalars  $\alpha_1 \dots \alpha_n$  such that  $\sum_i \alpha_i \vec{b}_i = \vec{0}$ . If a set of vectors is not linearly dependent, then we call them linearly independent. If  $\vec{b}_1 \dots \vec{b}_n$  are lin. ind. and can generate all of  $V$  using add and scal mult, then the set  $\vec{b}_i$  is called a basis of  $V$ .  $n$  is the dimension of the basis/space, which in our cases will usually be 3.

We can use a basis as a way to “address” all of the vectors in the space. This can be done using a coordinates  $c_i$  as follows.

$$\vec{v} = \sum_i c_i \vec{b}_i$$

We can use vector algebra notation and write this as

$$\vec{v} = \sum_i c_i \vec{b}_i = \begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

And if we are willing to create a bit of notation, we can simplify this as

$$\vec{v} = \vec{\mathbf{b}}^t \mathbf{c}$$

where  $\vec{v}$  is a vector,  $\vec{\mathbf{b}}^t$  is a row of basis vectors and  $\mathbf{c}$  is a coordinate vector.

### 1.3 Linear transformations and 3 by 3 matrices

A linear transformation  $\mathcal{L}$  is just a transformation from  $V$  to  $V$  which satisfies the following two properties.

$$\begin{aligned} \mathcal{L}(\vec{v} + \vec{u}) &= \mathcal{L}(\vec{v}) + \mathcal{L}(\vec{u}) \\ \mathcal{L}(\alpha\vec{v}) &= \alpha\mathcal{L}(\vec{v}) \end{aligned}$$

The class of transformations is exactly the class that can be expressed using matrices and matrix operations. This is because a linear transformation can be exactly specified by telling us its effect on the basis vectors.

Linearity of the transformation implies the following relationship

$$\vec{v} \Rightarrow \mathcal{L}(\vec{v}) = \mathcal{L}\left(\sum_i c_i \vec{b}_i\right) = \sum_i c_i \mathcal{L}(\vec{b}_i)$$

If we express  $\vec{v}$  using a basis and a coordinate vector, then the above relationship becomes

$$\begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} \Rightarrow \begin{bmatrix} \mathcal{L}(\vec{b}_1) & \mathcal{L}(\vec{b}_2) & \mathcal{L}(\vec{b}_3) \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

Since each of the 3 new vectors  $\mathcal{L}(\vec{b}_i)$  are themselves elements of  $V$ , each can ultimately be written as a linear combination of the original basis vectors.

$$\begin{bmatrix} \mathcal{L}(\vec{b}_1) & \mathcal{L}(\vec{b}_2) & \mathcal{L}(\vec{b}_3) \end{bmatrix} = \begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 \end{bmatrix} \begin{bmatrix} M_{1,1} & M_{1,2} & M_{1,3} \\ M_{2,1} & M_{2,2} & M_{2,3} \\ M_{3,1} & M_{3,2} & M_{3,3} \end{bmatrix}$$

Putting this all together, we see that the operation of the linear mapping operating on a vector can be expressed using matrix algebra.

$$\begin{aligned} & \begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} \\ \Rightarrow & \begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 \end{bmatrix} \begin{bmatrix} M_{1,1} & M_{1,2} & M_{1,3} \\ M_{2,1} & M_{2,2} & M_{2,3} \\ M_{3,1} & M_{3,2} & M_{3,3} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} \end{aligned}$$

We will use matrices for three important operations. First, as we described above, we can take a matrix and use to transform one vector to another

$$\vec{\mathbf{b}}^t \mathbf{c} \Rightarrow \vec{\mathbf{b}}^t M \mathbf{c}$$

We can also use a matrix to transform one basis into another basis

$$\vec{\mathbf{b}}^t \Rightarrow \vec{\mathbf{b}}^t M$$

Finally, we can use a matrix to transform one coordinate vector to another

$$\mathbf{c} \Rightarrow M \mathbf{c}$$

### 1.3.1 How to read a matrix expression

In computer graphics, we will often want to apply sequences of operations to our vectors. Such as first translate a point by some amount, and then rotate it in some way about an origin point. In order to create the correct matrix sequence, we must get proficient at reading matrix expressions.

We start by noting that matrix multiplication is associative. This gives us two ways to read the expression

$$\vec{\mathbf{b}}^t \mathbf{c} \Rightarrow \vec{\mathbf{b}}^t \mathbf{M} \mathbf{c}$$

If we place the parenthesis on the left part of the expression, we get

$$\begin{aligned} & \left[ \begin{array}{ccc} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 \end{array} \right] \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} \\ \Rightarrow & \left( \left[ \begin{array}{ccc} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 \end{array} \right] \begin{bmatrix} M_{1,1} & M_{1,2} & M_{1,3} \\ M_{2,1} & M_{2,2} & M_{2,3} \\ M_{3,1} & M_{3,2} & M_{3,3} \end{bmatrix} \right) \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} \\ = & \left[ \begin{array}{ccc} \vec{b}'_1 & \vec{b}'_2 & \vec{b}'_3 \end{array} \right] \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} \end{aligned}$$

We can interpret the mapping as doing its job by changing the basis from  $\vec{\mathbf{b}}^t$  to  $\vec{\mathbf{b}}'^t$ ,

$$\vec{\mathbf{b}}^t \mathbf{c} \Rightarrow \vec{\mathbf{b}}'^t \mathbf{c}$$

where

$$\vec{\mathbf{b}}'^t = \vec{\mathbf{b}}^t \mathbf{M}$$

and using the new basis with the original coordinate vector  $\mathbf{c}$ .

If we place the parenthesis around the right side of the expression we get a second way to read the transformation

$$\begin{aligned} & \left[ \begin{array}{ccc} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 \end{array} \right] \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} \\ \Rightarrow & \left[ \begin{array}{ccc} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 \end{array} \right] \left( \begin{bmatrix} M_{1,1} & M_{1,2} & M_{1,3} \\ M_{2,1} & M_{2,2} & M_{2,3} \\ M_{3,1} & M_{3,2} & M_{3,3} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} \right) \\ = & \left[ \begin{array}{ccc} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 \end{array} \right] \begin{bmatrix} c'_1 \\ c'_2 \\ c'_3 \end{bmatrix} \end{aligned}$$

In this reading, we interpret the matrix working by changing the coordinate vector

$$\vec{b}^t \mathbf{c} \Rightarrow \vec{b}^t \mathbf{c}'$$

where

$$\mathbf{c}' = \mathbf{M}\mathbf{c}$$

while keeping the basis fixed.

### 1.3.2 The basis is important

If i give you a vector and tell you to transform it using a matrix, i have actually not told you enough to specify the actual mapping. I also must tell you what basis we are using.

Here is a simple example showing this. Suppose i start with some vector  $\vec{v}$  and the matrix

$$\mathbf{S} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

If we look at how this matrix operates, we can see that this is the matrix for applying a non-uniform scaling to the vector.

Now lets fix a basis  $\vec{b}^t$ . Using this basis, the vector can be expressed using the appropriate coordinate vector  $\vec{v} = \vec{b}^t \mathbf{c}$ . If we now use the matrix to transform the vector as described above, we get  $\vec{b}^t \mathbf{c} \Rightarrow \vec{b}^t \mathbf{S}\mathbf{c}$ . In this case the effect of the matrix is to transform the vector by stretching it by a factor of two in direction of the first vector of the basis  $\vec{b}_1$ ; we often call this the  $x$  direction.

Suppose i instead pick some other basis  $\vec{a}^t$ . and suppose that tis basis is related to the original one by the matrix equation  $\vec{a}^t = \vec{b}^t \mathbf{M}$ . We can express the original point in the new basis with a new coordinate vector  $v v v = \vec{b}^t \mathbf{c} = \vec{a}^t \mathbf{d}$ , where  $\mathbf{d} = \mathbf{M}^{-1} \mathbf{c}$ .

Now if we use  $S$  to perform a transformation on the vector represented in  $\vec{b}^t$ , we get  $\vec{a}^t \mathbf{d} \Rightarrow \vec{a}^t \mathbf{S}\mathbf{d}$ . In this case we have scaled the same vector  $\vec{v}$  about the first vector of the basis  $\vec{a}^t$ . This is a different transformation.

The important thing to notice here is that the vector is transformed (non uniform scaling in this case) with respect to the the basis that appears immediately to the left of transformation matrix in the expression. This rule is so important that we will call it the “left of” rule.

## 1.4 Linear transformation example: Rotations

One of the most useful linear transformations to apply to a vector is a rotation. For simplicity, we will begin with describing a vector in the plane which is done with the

two dimensional coordinate vector

$$\begin{bmatrix} x \\ y \end{bmatrix}$$

Suppose i wish to rotate this point by  $\theta$  degrees counter clockwise about the origin, the coordinates  $[x', y']^t$  of the rotated can be computed as

$$\begin{aligned} x' &= x \cos \theta - y \sin \theta \\ y' &= x \sin \theta + y \cos \theta \end{aligned}$$

This can be written as the following matrix equation

$$\begin{aligned} &\begin{bmatrix} \vec{b}_1 & \vec{b}_2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \\ \Rightarrow &\begin{bmatrix} \vec{b}_1 & \vec{b}_2 \end{bmatrix} \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \end{aligned}$$

In 2D, the forward direction of rotation was counterclockwise. In 3D we describe the forward rotation direction as “right handed”. First of all, we will be using a right handed coordinate system. This means that if you were to grab onto the  $z$  axis with your right hand, such that if when you hand is open, your pinky lines up with the  $x$  axis, when you close you hand, by rotating you fingers ninety degrees towards the closed position, your pinky then lines up with the  $y$  axis. For this setting a forward rotation around the  $z$  axis is the direction of your finger tips as you close your hand.

Rotations in 3d are a bit trickier. Lets start with some easy examples. To rotate a point by  $\theta$  degrees around the  $z$  axis of the basis we can just apply:

$$\begin{aligned} &\begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \\ \Rightarrow &\begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 \end{bmatrix} \begin{bmatrix} c & -s & 0 \\ s & c & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \end{aligned}$$

where for brevity, we have used the notation  $c \equiv \cos \theta$ , and  $s \equiv \sin \theta$ . As expected, this transformation leaves points on the  $z$  axis just where they were. Each fixed  $z = k$  plane for some constant  $k$ , acts just like the 2D rotation describe above.

Just like we saw with the non uniform scaling example, the basis chosen is important. This matrix will rotatate about the  $z$  axis of the chosen basis.

A forward rotation about the  $x$  axis of a basis can be computed as

$$\begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c & -s \\ 0 & s & c \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Again, the forward direction can be visualized by grabbing the  $x$  axis with your right hand, with the heel of your hand against the  $x = 0$  plane; the forward direction is followed by your fingertips as you close your hand.

A forward rotation around the  $y$  axis is done using the matrix

$$\begin{bmatrix} c & 0 & s \\ 0 & 1 & 0 \\ -s & 0 & c \end{bmatrix}$$

In some sense, this is all you need to get any 3D rotation. It can be shown that one can achieve any arbitrary rotations by applying one  $x$ , one  $y$ , and one  $z$  rotation. The amount of the three rotations are called the  $xyz$  Euler angles. In fact, one can achieve all rotations using only two rotation axes. For example, one can achieve any arbitrary rotations by applying one  $x$ , one  $y$ , and another  $x$  rotation, using  $xyx$  Euler angles. Euler angles can be visualized by thinking of a set of gimbles, with three movable axes, with three settings to determine the achieved rotation.

A more direct way to represent an arbitrary rotation, is to pick a unit vector  $[k_x, k_y, k_z]^t$  as the axis of rotation, and directly apply a rotation of  $\theta$  about that axis in the forward right handed direction. This can be achieved using the following matrix

$$\begin{bmatrix} k_x^2 v + c & k_x k_y v - k_z s & k_x k_z v + k_y s \\ k_y k_x v + k_z s & k_y^2 v + c & k_y k_z v - k_x s \\ k_z k_x v - k_y s & k_z k_y v + k_x s & k_z^2 v + c \end{bmatrix}$$

where for shorthand we have introduced the symbol  $v \equiv 1 - c$

Later in this book, we will introduce the quaternion representation for rotations, which will be useful for creating smooth interpolations between object orientations.