Vectors and matrices

From DmWiki

Two of the basic mathematical concepts that are needed for working your way around 3-dimensional space are **vectors** and **matrices**. Both are intuitive concepts, but learning to manipulate them formally can save you a lot of time and trouble in solving the various geometric problems you will encounter as you write your game.

Table of contents

Vectors

In basic geometric terms, a **vector** is an arrow. It has a certain direction and a certain length. That's really all there is to a vector - it's a quantity that has a direction and a length (also called **magnitude**). Some vectors can be thought of as "floating" anywhere in space while maintaining the same direction, while other vectors are bound to a particular point in space.

You can use a vector for a vast number of different purposes. For instance, the location of a point in space can be the vector that points from the origin (0, 0, 0) to the point you want. The velocity of a moving object is a vector, as well as its acceleration. In addition, force, angular velocity, and torque are all physics concepts that can be represented by vectors.

Very often you will want to use vectors to represent things that are directions only - for instance, the direction from a point to a light source. In this case, it is common for the vector to be **normalized** so that its length is 1. A vector that has been normalized is called a **unit vector**. Working with unit vectors allows some calculations to be simplified.

Vectors and Coordinate Systems

You probably already know how to use a coordinate system. In 3-dimensional space you have the three axes, called x, y, and z, and points in space can be identified by three numbers, which are the **coordinates** of the point. Vectors can be represented using coordinates too. You just take the coordinates of the vector's endpoint and subtract the coordinates of its initial point. For example, suppose the vector **a** starts at (1, 2, -2) and ends at (5, -3, 1). Then the coordinates of **a** (technically speaking, the components of **a** with respect to this particular coordinate system) are:

$$\mathbf{a} = \begin{bmatrix} 5 \\ -3 \\ 1 \end{bmatrix} - \begin{bmatrix} 1 \\ 2 \\ -2 \end{bmatrix} = \begin{bmatrix} 4 \\ -5 \\ 3 \end{bmatrix}.$$

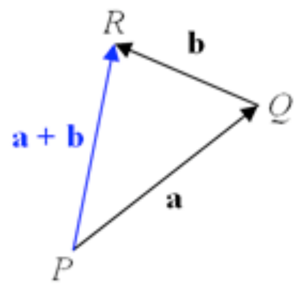
So this vector is 4 units long in the *x* axis, 5 units long in the negative direction on the *y* axis, and 3 units long on the *z* axis.

Note that you are not limited to just one coordinate system. In fact, in a game situation you will be using quite a large number of different coordinate systems. See the article on coordinate systems for more information. The same vector will have different coordinates in each coordinate system, so it is typical to have a standard "world" coordinate system that acts as a base from which all other coordinate systems can be defined.

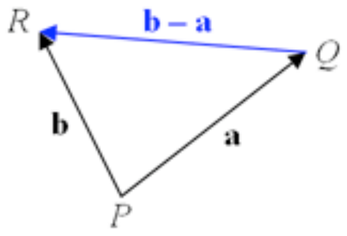
Operations on Vectors

Some basic geometric operations can be performed on vectors.

- You can **add** two vectors by adding their coordinates. Geometrically, thinking of vectors as arrows, this corresponds to taking the tail of one arrow and putting it at the head of the other arrow. So if the vector **a** points from *P* to *Q*, and **b** points from *Q* to *R*, then the combined vector **a + b** points from *P* to *R*.



- You can also **subtract** two vectors, by subtracting their coordinates. Geometrically, this corresponds to moving both vectors so that they start at the same point, and drawing the vector that connects their end points. So if **a** points from *P* to *Q*, and **b** points from *P* to *R*, then **b - a** points from *Q* to *R*.



- If you want to change the length of a vector while keeping its direction the same, you can multiply it by a scalar (that is, a regular number, not a vector). For example, multiplying the vector by 2 will double its length. Multiplying by -2 will double the length and reverse the direction.

You can calculate the length of a vector using the Pythagorean theorem. If we let a_1,a_2,a_3 denote the components of the vector **a**, then the length of **a** is:

$$\|\mathbf{a}\| = \sqrt{a_1^2 + a_2^2 + a_3^2}$$

This means that you can normalize a vector just by multiplying it by one over its length. The square root involved means that normalizing a vector can be a relatively expensive operation, so it is best to avoid it unless absolutely necessary.

Vector Products

Vectors can be added and subtracted just like ordinary numbers can, and they can also be multiplied (and divided) by scalars. But vector multiplication is not so straightforward. There are actually two completely different ways to multiply vectors, and they have completely different uses.

The Dot Product

The most common way to multiply two vectors is to take their **dot product**, also called the **inner product** or **scalar product**. If a_1,a_2,a_3 are the components of **a** and b_1,b_2,b_3 are the components of **b**, then we define their dot product as follows:

$$\mathbf{a} \cdot \mathbf{b} = a_1 b_1 + a_2 b_2 + a_3 b_3$$

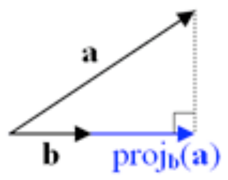
Note that the dot product is a scalar, not a vector!

This equation shows how to calculate the dot product, but it doesn't tell us anything about what the dot product can actually be used for. It can be proven, mathematically, that the dot product is also:

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

That is, the dot product equals the lengths of the two vectors multiplied together with the cosine of the angle between them. This leads to the three main uses of the dot product:

- You can tell if two vectors are perpendicular. If they are perpendicular, their dot product equals zero. Furthermore, if the angle between them is acute (< 90 degrees) the dot product will be positive; if the angle is obtuse (> 90 degrees) the dot product will be negative.
- You can find the exact angle between two unit vectors, by taking the arccosine of their dot product.
- You can use the dot product to find the projection of one vector onto another. This works as follows: the projection of **a** onto **b** is
  $$\mathrm{proj}_{\mathbf{b}}(\mathbf{a}) = \mathbf{b}\left(\frac{\mathbf{a} \cdot \mathbf{b}}{\mathbf{b} \cdot \mathbf{b}}\right)$$
  Note that the denominator can be left off if **b** is a unit vector, since a vector's dot product with itself equals the square of its magnitude, which is in this case 1. The diagram below illustrates projection:



The Cross Product

The other way to multiply vectors is to take their **cross product**, also called the **outer product** or **vector product**. When you take the cross product of two vectors, the result is again a vector (unlike with the dot product). However, the cross product is defined only for 3-dimensional vectors; it cannot be used with 2-dimensional or 4-dimensional ones. The cross product is defined like this:

$$\mathbf{a} \times \mathbf{b} = \begin{bmatrix} a_2b_3 - a_3b_2 \\ a_3b_1 - a_1b_3 \\ a_1b_2 - a_2b_1 \end{bmatrix}$$

Geometrically speaking, the cross product of **a** and **b** gives you a vector that is perpendicular to both **a** and **b**. This is the most common use of the cross product. The cross product can be used to find normal vectors, axes for rotation, and in other situations where perpendicular vectors are needed.

Note that the cross product is anticommutative. That is,

$$\mathbf{a} \times \mathbf{b} = -(\mathbf{b} \times \mathbf{a})$$

This means that the order of multiplication is important. Geometrically, the cross product obeys the right-handed rule; that is, if you make a thumbs-up with your right hand, your thumb points in the direction of **a** × **b** when your fingers are curling in the direction from **a** towards **b**. If you turn your hand upside down, so that your fingers curl from **b** towards **a**, then the thumb points in the opposite direction.

Additionally, the length of the cross product is:

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

That is, the length of the cross product is the lengths of the individual vectors, multiplied together with the sine of the angle between them. This means you can use the cross product to tell when two vectors are parallel, because if they are parallel their cross product will be zero.

Matrices

Vectors are the basic geometric entities that allow you to talk about locations and directions in 3D space. Now we will introduce **matrices**, which are the entities that let you "do things" to vectors. For example, matrices can be used to apply geometric transformations, such as translations, rotations, and scales, to vectors. Matrices can also be used to express vectors in different coordinate systems.

Detailed information on how to use matrices to accomplish these tasks can be found in the other articles:

- Transformation matrices
- Coordinate systems

This article will explain the basic matrix operations, such as multiplication, that are needed to move to more advanced matrix topics.

First of all, we must define a matrix. It is simply a two-dimensional array of numbers. For example,

$$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{bmatrix}$$

is an example of a 3×3 matrix. Note that when talking about the size of a matrix, the number of rows is stated first, followed by the number of columns. This would be a 2×4 matrix:

$$\begin{bmatrix}1 & 2 & 3 & 4\\5 & 6 & 7 & 8\end{bmatrix}$$

Although matrices can come in any size, the ones used in 3D graphics are usually 3×3 and 4×4.

Parts of a Matrix

The individual numbers inside a matrix are called **elements**. In mathematical notation, if we have a matrix **A**, its elements will be denoted by a_{ij}, where i is the row number and j is the column number. For example, if **A** is the 3×3 matrix above, then $a_{11} = 1$ and $a_{23} = 6$.

If the matrix is square (the number of rows and the number of columns are equal), then its **diagonal** is the elements for which $i = j$; that is, the elements from the top left to the bottom right. In the 3×3 matrix above, the elements 1, 5, and 9 are on the diagonal.

Basic Operations

- Matrices of the same size can be added and subtracted by adding or subtracting their corresponding components. For example,
$$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix} + \begin{bmatrix}1 & 0\\0 & -1\end{bmatrix} = \begin{bmatrix}2 & 2\\3 & 3\end{bmatrix}$$
Addition and subtraction of matrices is not useful for much of anything.
- Matrices can be multiplied by a scalar, the same was as vectors can. Again, multiplying a matrix by a scalar isn't terribly useful.
- Matrices can be **transposed**. This means "flipping" the matrix about its diagonal. so that rows become columns and vice versa. Transposing is denoted in mathematical notation by a superscript T. For example, if
$$\mathbf{A} = \begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{bmatrix},$$
then
$$\mathbf{A}^T = \begin{bmatrix}1 & 4 & 7\\2 & 5 & 8\\3 & 6 & 9\end{bmatrix}.$$

Matrix Multiplication

The most important operation that can be done with matrices is multiplying them. As with vectors, multiplication of matrices is not done the most straightforward way; there is a special way to multiply matrices.

The first thing to know about matrix multiplication is that there is a restriction on the size of the matrices that can be multiplied. Suppose you have an $m \times n$ matrix. Then it can only be multiplied by an $n \times r$ matrix. That is, the number of columns in the first matrix must equal the number of rows in the second matrix. When you multiply the two, the result is an $m \times r$ matrix.

For example:

- You can multiply two 4×4 matrices and the result is a 4×4 matrix.
- If you mutliply a 4×3 matrix with a 3×3 matrix, the result is a 4×3 matrix.
- You **cannot** multiply a 3×3 matrix with a 4×3 matrix.
- A 2×3 matrix multiplied by a 3×4 matrix gives a 2×4 matrix.

It should be clear by now that matrix multiplication is not commutative. That is, if **A** and **B** are matrices, then **AB** and **BA** are probably not the same. In fact, depending on the sizes, if you can multiply **AB** you might not even be able to multiply **BA**. So the order in which matrices are multiplied makes a difference.

Now we can learn how matrix multiplication actually works. It depends on thinking of rows and columns of matrices as being vectors. For example, we could take the 1st column of a 4×4 matrix and think of it as a 4-dimensional vector.

Let's say that we have matrices **A** and **B** and they fulfill the size requirement stated above. Then we can calculate their product **AB** like this: the *ij*th element of **AB** is the dot product of the *i*th row of **A** and the *j*th column of **B**.

For example, the following bit of code will multiply two 4×4 matrices:

```
// Multiply two 4x4 matrices A and B, storing the result in C
void multiply4x4 (const float A[][], const float B[][], float C[][])
{
    // Loop through each element of C
    for (int i = 0; i < 4; ++i)
    {
        for (int j = 0; j < 4; ++j)
        {
            C[i][j] = 0.0f;
            // Calculate the dot product of ith row of A and jth column of B
            for (int k = 0; k < 4; ++k)
                C[i][j] += A[i][k] * B[k][j];
        }
    }
}
```

This can also be expressed in standard mathematical notation as follows:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

where n is the number of columns of the first matrix, which must equal the number of rows of the second matrix.

Multiplying Matrices with Vectors

Another kind of multiplication you can do is to multiply a matrix by a vector. This is a very important operation, and in fact one that GPUs are specially constructed to perform.

All you have to do is think of a vector as a matrix with one column. For example, a 3-dimensional vector would be a 3×1 matrix, and a 4-dimensional vector would be a 4×1 matrix. Then you perform matrix multiplication as usual. The result of multiplying a matrix and a vector is another vector. The following piece of code multiplies a 4x4 matrix with a 4-dimensional vector.

```
// Multiply a 4x4 matrix A by a 4-vector x, storing the result in y
void multiply4x4vec (const float A[][], const float x[], float y[])
{
    // Loop through the elements of y
    for (int i = 0; i < 4; ++i)
    {
        y[i] = 0.0f;
        // Calculate the dot product of the ith row of A with x
        for (int j = 0; j < 4; ++j)
            y[i] += A[i][j] * x[j];
    }
}
```

Multiplying a matrix by a vector corresponds to applying a geometric transformation of some kind to the vector. For more information, see transformation matrices and coordinate systems.

The Identity Matrix

In ordinary number multiplication, when you multiply a number by 1 you get the same number back. Correspondingly, in matrix multiplication there is a special matrix called the **identity matrix** that gives you the same matrix or vector back whenever you multiply it by something.

The *n*×*n* identity matrix is:

$$\mathbf{I}_n = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

That is, it is the matrix that has 1s on the diagonal and 0s everywhere else. Identity matrices must be square, obviously, but they can be any square size.

Inverting Matrices

In ordinary algebra, one can use division to undo the effects of multiplication. There is no matrix division, but some matrices can be inverted. This corresponds to taking the reciprocal of an ordinary number. The **inverse** of a matrix is another matrix that "undoes" the effect of the original matrix.

Only square matrices can be invertible (but not all square matrices are). The inverse of a matrix **A** is denoted by $\mathbf{A}^{-1}$ and satisfies the properties:

$$\mathbf{A}^{-1}\mathbf{A} = \mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$$

That is, multiplying a matrix by its inverse gives the identity matrix. Also, the inverse of $\mathbf{A}^{-1}$ is **A** itself. One can deduce, therefore, that for any vector **x**,

$$\mathbf{A}^{-1}(\mathbf{A}\mathbf{x}) = \mathbf{x}$$

so that multiplying by $\mathbf{A}^{-1}$ undoes the effect of **A** and gets you back the original vector **x**.

Unfortunately, giving a full explanation of how to calculate matrix inverses is out of the scope of this document. More information can be found at Wikipedia: Matrix inversion (*http://en.wikipedia.org/wiki/Matrix_inversion*). If you like, you can use the code below to do matrix inversion.

```
// 4x4 matrix inverse using Gauss-Jordan algorithm with row pivoting
// originally written by Nathan Reed, now released into the public domain.
const mat4 mat4::inverse () const
{
        mat4 a(*this);          // The matrix whose inverse we want to take
        mat4 b(identity);       // initialize to 4x4 identity matrix
        int i, j, pivot;

        // loop through columns
        for (j = 0; j < 4; ++j)
        {
                // select pivot element: maximum magnitude in this column below row j
                for (pivot = j, i = j; i < 4; ++i)
                        if (fabs(a[i][j]) > fabs(a[pivot][j]))
                                pivot = i;
                if (fabs(a[pivot][j]) < epsilon)
                        throw std::domain_error("Matrix inverse does not exist!");

                // interchange rows to put pivot element on the diagonal
                if (pivot != j)                                         // skip if already on diagonal
                {
                        vec4 temp = a.row(j);
                        a.row(j) = a.row(pivot);
                        a.row(pivot) = temp;
                        temp = b.row(j);
                        b.row(j) = b.row(pivot);
                        b.row(pivot) = temp;
                }

                // divide row by pivot element
                if (a[j][j] != 1.0f)                                    // skip if already equal to 1
                {
                        float temp = a[j][j];
                        a.row(j) /= temp;
                        b.row(j) /= temp;
                        // now the pivot element is 1
                }

                // subtract this row from others to make the rest of column j zero
                for (i = 0; i < 4; ++i)
                {
                        if ((i != j) && (fabs(a[i][j]) > epsilon))       // skip rows that are already
                        {
                                float scale = -a[i][j];
                                a.row(i) += a.row(j) * scale;
                                b.row(i) += b.row(j) * scale;
                        }
                }
        }
        // When these operations have been completed, a should have been transformed to the identity ma
        // and b should have been transformed into the inverse of the original a
        return b;
}
```

Retrieved from "http://www.devmaster.net/wiki/Vectors_and_matrices"

Categories: Math and Physics

- This page was last modified 17:53, 7 Aug 2008.