Dan Morris
CS148

Handout #3
Summer 2005

# Essential Mathematics for Computer Graphics

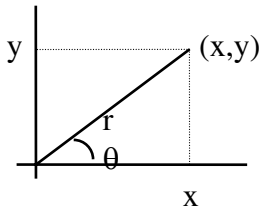
- Trigonometry
- Polar Coordinates
- 3-D Coordinate Systems
- Parametric Representations
- Points and Vectors
- Matrices

---

This handout provides a compendium of the math that we will be using this quarter. It does not provide a complete presentation of the topics, but instead just gives the "facts" that we will need. We will not cover most of this in class, so you may want to keep this nearby as a handy reference.

## Trigonometry

Let θ be an angle with its vertex at the origin and one side lying along the x-axis, and let (x,y) be any point in the Cartesian plane other than the origin, and on the other side of the angle. The six trigonometric functions are as follows:




| | | |
|---|---|---|
| $\sin\theta = y / r$ | $\cos\theta = x / r$ | $\tan\theta = y / x$ |
| $\csc\theta = r / y$ | $\sec\theta = r / x$ | $\cot\theta = x / y$ |

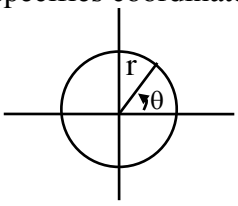
Angles can be specified in degrees or radians. One radian is defined as the measure of the angle subtended by an arc of a circle whose length is equal to the radius of the circle. Since the full circumference of a circle is $2\pi r$, it takes $2\pi$ radians to get all the way around the circle, so $2\pi$ radians is equal to 360°.

$1 \text{ radian} = 180/\pi = 57.296°$ $\qquad$ $1° = \pi/180 \text{ radian} = 0.017453 \text{ radian}$

FYI, OpenGL expects angles to be in degrees.

**Polar Coordinates**

We are used to the standard Cartesian coordinate system of graphing, but there are many other methods. One common method is the polar coordinate method. In this system, the origin is at the center of the plane and one specifies coordinates by giving an angle θ and a radius r:




If you need to translate polar coordinates to Cartesian coordinates, use these formulas:

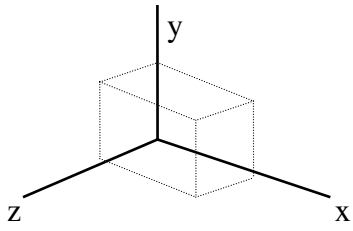
$x = r \cos \theta$ $\qquad$ $y = r \sin \theta$

Going from Cartesian to polar:

$r = \text{sqrt}(x^2 + y^2)$ $\qquad$ $\theta = \tan^{-1}(y/x)$

**3-D Coordinate Systems**

In a 3-d Cartesian system, we specify 3 values to identify a point (x,y,z):




This is a right-handed system, which is most often used in graphics, i.e., the thumb of the right hand points down the z-axis if we imagine grabbing the z-axis with the fingers of the right hand curling from the positive x-axis toward the positive y-axis.

In OpenGL's coordinate system, the *positive* z axis points *out of* the screen.
In OpenGL's coordinate system, the *positive* z axis points *out of* the screen.

I wrote that sentence twice on purpose because it's just that important to remember this.

**Parametric Representations**

There are two ways to visualize a curve: as a line "frozen" in space, or as the path of a particle as it moves in space. The first view leads us to describe the curve according to a function that defines the x- and y-coordinates of the curve; we'll call this the *functional form*. Functional-form equations come in two flavors: *explicit* and *implicit*. An explicit equation specifies the y-coordinate for each value of x as $y = f(x)$ or vice versa. For example, the equation for a line is $y = mx + b$. An implicit equation usually sets some function of all variables equal to a constant. For example, the equation for a circle is $x^2 + y^2 - r^2 = 0$.

A *parametric form* is the other way to view a curve. It suggests the movement of a point through time. The *parameter* is the value t (time) and is used to distinguish one point on the curve from another. The path of the particle traveling along the curve is fixed by two functions: x() and y() and we speak of (x(t), y(t)) as the position of the particle at time t. The curve is all the points visited by the particle over some interval, e.g., from 0 to 1.

Now some terminology:

A *line segment* is a straight path between two points, and it can be defined by its two endpoints.

A *ray* is just like a line segment, only it extends infinitely in one direction. It can be defined by its endpoint and one other point on the ray, or by its endpoint and its direction (a vector).

A *line* is just like a ray, only it extends infinitely in both directions. It can be defined by two points or by any point on the line and its direction (a vector).

Back to parametric representations…

Lines, line segments and rays all share the same parametric representation. All points on a line segment from a = (ax, ay) to b = (bx, by) are represented by the parametric form:

$$x(t) = ax + (bx - ax)t$$
$$y(t) = ay + (by - ay)t$$

Usually we let t vary from 0 to 1. When t = 0, the point (x(t), y(t)) is at point a. As t increases toward 1, the point moves in a straight line toward b. It is midway between the two points when t = 1/2, and in general, it is a fraction f of the way from a to b at t = f.

The ray that starts at a and passes through b is also defined by these equations but t is allowed to take on any nonnegative value. The ray passes through b at t = 1 but then continues forever along the same path. The line defined by points a and b is also defined by these equations but now all real values of t are permitted. Thus, line segments, rays and lines differ parametrically only in the values of t that are relevant:

line segment: $0 \le t \le k$ (where k is often 1)
ray: $0 \le t < \infty$

line: $-\infty < t < \infty$

A circle with radius R centered at the origin of the Cartesian plane has the following parametric representation:

$$x(t) = R * \cos(2\pi t)$$
$$y(t) = R * \sin(2\pi t)$$

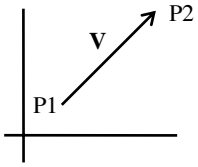
for t between 0 and 1. If you try graphing this with equidistant values of t, you'll end up with a circle.

## Points and Vectors

Basics

A *point* is a position specified with coordinate values in some reference frame. Points are only meaningful if I give you a reference frame, aka an origin and a set of coordinate axes. This is really important for CS148. Remember in Indiana Jones and the Last Crusade when the Joneses had detailed instructions on how to find the Holy Grail, but they didn't know where the instructions started? When they found out that the instructions started at Alexandretta, that was like defining a reference frame. That was a loosely-related tangent. That movie was totally sweet.

A *vector* is the difference between two points:




$$\mathbf{V} = P2 - P1$$
$$= (x2 - x1, y2 - y1)$$
$$= (Vx, Vy)$$

(Vx, Vy) are called the Cartesian *components* or *elements*, and are the projections of **V** onto the x and y axes. We can describe a vector as a directed line segment (starting at the *initial point* and ending with the *terminal point*) that has two properties: magnitude (length) and direction. To calculate magnitude, we use a variation of the Pythagorean Theorem:

$$|\mathbf{V}| = \text{sqrt}(Vx^2 + Vy^2)$$

The direction is given in terms of the angular displacement with the x axis:

$$\alpha = \tan^{-1} (Vy / Vx)$$

In a 3-d Cartesian space, magnitude is defined by:

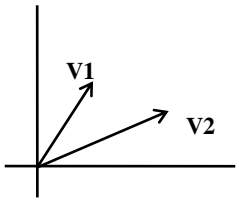
$$| \mathbf{V} | = \text{sqrt}(Vx^2 + Vy^2 + Vz^2)$$

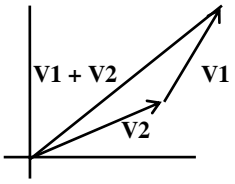
Vector direction in 3-D is given with direction angles, $\alpha$ (x-axis), $\beta$ (y-axis), and $\gamma$ (z-axis):

$$\cos \alpha = Vx / | \mathbf{V} |$$
$$\cos \beta = Vy / | \mathbf{V} |$$
$$\cos \gamma = Vz / | \mathbf{V} |$$

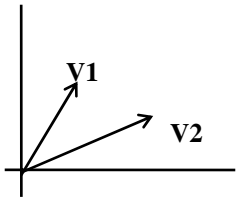
Addition and Subtraction of Vectors

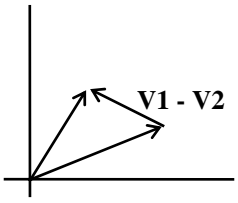
Sum of two vectors: **V1** + **V2** = (V1x + V2x, V1y + V2y, V1z + V2z). Geometrically, the sum of two vectors means placing the start position of one vector (V1) at the tip of the other (V2) and then drawing the vector from the initial point of V1 to the terminal point of V2.




Subtraction (V1-V2) is defined as placing the initial point of both vectors at the origin and drawing a vector from the terminal point of V2 to the terminal point of V1.




I don't need to know the reference frame that the points live in to perform a vector addition or subtraction, but I probably need to know the reference frame to do anything useful with the result.

Scalar Multiplication

Scalar multiplication of a vector and a constant: a$\mathbf{V}$ = (aVx, aVy, aVz). For example, if a = 2, each component is doubled. If we scale a vector with a negative scalar, we reverse its direction.

It is often convenient to scale a vector so that the result has a length equal to 1. This is called *normalizing* the vector and the result is known as the *unit vector*. To form the normalized versions $\mathbf{u_a}$ of a vector **a**, we scale the vector **a** with the value 1 / |**a**|:

$$\mathbf{u_a} = \mathbf{a} / |\mathbf{a}| \qquad \text{giving} \qquad |\,\mathbf{u_a}\,| = 1$$

We sometimes refer to a unit vector as a *direction* since its magnitude = 1. Note that any vector can be written as its magnitude times its direction, thus if $\mathbf{u_a}$ is a normalized version of **a** then: $\mathbf{a} = |\mathbf{a}|\,\mathbf{u_a}$

Standard Unit Vectors

The special vectors **i**, **j** (and **k** in 3-d) are called *standard unit vectors* and are defined as follows:

$$\mathbf{i} = (1,0,0) \qquad \mathbf{j} = (0,1,0) \qquad \mathbf{k} = (0,0,1)$$

These vectors allow us to obtain an alternate way of denoting vectors in 2-D or 3-D. In 2-D:

if **a** = (a1, a2) then **a** = (a1,0) + (0, a2) = a1(1,0) + a2(0,1) = a1**i** + a2**j**

This final sum is called a *linear combination* of **i** and **j**.

Linear Combinations

Now that we know how to add and scale vectors, it is useful to define more about linear combinations of m vectors $\mathbf{v_1}$, $\mathbf{v_2}$, ..., $\mathbf{v_m}$ which is a vector of the form:

$$\mathbf{w} = a_1\mathbf{v_1} + a_2\mathbf{v_2} + ... + a_m\mathbf{v_m}$$

where $a_1$, $a_1$, ..., $a_m$ are scalars. For example, the linear combination of 2(3,4) + 3/2(2,4) gives the vector (9,14). Thus, all we are doing is scaling or "weighting" the input vectors by some values and then adding the weighted versions.

A special class of linear combinations has an important place in graphics: *the convex combinations*. These are linear combinations for which the coefficients are nonnegative and add up to one. $\mathbf{w} = a_1\mathbf{v_1} + a_2\mathbf{v_2} + ... + a_m\mathbf{v_m}$ will be a convex combination if all the nonnegative scalars $a_i$ add up to 1, and each $a_i >= 0$. These coefficients are said to make a partition of unity meaning that a unit amount of material is partitioned into pieces.

Multiplication of Vectors: Dot Product

There are two ways to perform multiplication of vectors, one resulting in another vector, the other resulting in a scalar value. The *scalar* or *dot* or *inner product* of two vectors gives us a scalar, and it's defined as:

$$\mathbf{V}1 \bullet \mathbf{V}2 = V1xV2x + V1yV2y + V1zV2z$$

Basically we just multiply each component in one vector by the corresponding component in the other vector, and add up the results. The dot product has the fantastic property that:

$$\mathbf{V}1 \bullet \mathbf{V}2 = |\,\mathbf{V}1\,||\,\mathbf{V}2\,| \cos\theta \quad \text{where } 0 \le \theta \le \pi \text{ and } \theta \text{ is the angle between the two vectors}$$

In other words, we can use the dot product of two vectors to determine the angle between those vectors, by computing:

$$\theta = \cos^{-1}\left( (\mathbf{V}1 \bullet \mathbf{V}2) / ( |\,\mathbf{V}1\,||\,\mathbf{V}2\,| ) \right)$$

The dot product is often said to measure the "match" or "similarity" between two vectors. Imagine two vectors moving in space like the hands of a clock. If we hold their lengths constant, we see by the equation above that the dot product is proportional to the cosine of the angle between them. For example, if the angle = 0, the cosine is at its max and the dot product is also at its max. As the two vectors move farther apart, the cosine decreases and the dot product decreases. It is its most negative at 180° when the vectors point in opposite directions. Thus, the closer the vectors are, the larger the dot product; the more they point in the opposite direction, the more negative their dot product. This can be summarized as follows:

The angle between two vectors is:

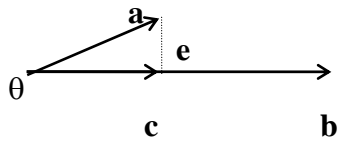
a) less than 90° if $\mathbf{a} \bullet \mathbf{b} > 0$
b) exactly 90° if $\mathbf{a} \bullet \mathbf{b} = 0$
c) greater than 90° **if** $\mathbf{a} \bullet \mathbf{b} < 0$

Dot products have the following properties:

a) symmetry: $\mathbf{a} \bullet \mathbf{b} = \mathbf{b} \bullet \mathbf{a}$
b) linearity: $(\mathbf{a} + \mathbf{c}) \bullet \mathbf{b} = \mathbf{a} \bullet \mathbf{b} + \mathbf{c} \bullet \mathbf{b}$
c) homogeneity: $(s\mathbf{a}) \bullet \mathbf{b} = s(\mathbf{a} \bullet \mathbf{b})$ where s is a scalar.
d) $|\mathbf{b}|^2 = \mathbf{b} \bullet \mathbf{b}$

A useful application of the dot product arises when we discuss *decomposing* a vector into two components: one component in the direction of a second given vector, and the other that is perpendicular to the second vector.

In the real world, I might know how to get somewhere by walking in a straight line, but maybe I have to climb over scary fences to do that, so when I give you directions, I might tell you to walk four blocks east, then three blocks north. I have *decomposed* the original vector into a vector parallel to compass north, and a vector perpendicular to compass north.




Vector **a** is shown here resolved into a first component **c** along the vector **b**, and a second component **e** that is perpendicular to **b**. By vector addition, we see that **e** = **a** - **c** which is basically what the idea of resolving a vector is all about. The vector **c** is called *the projection* of **a** onto **b**; it has the same direction as **b**, but a different length.

Now how do we compute these projections…

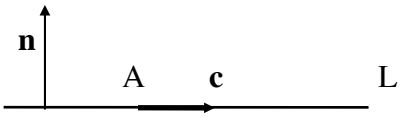
We know that $|\mathbf{c}| = |\mathbf{a}| \cos\theta$ from basic trig. Using this equation and the definition for dot product $\mathbf{a} \bullet \mathbf{b} = |\mathbf{a}|\,|\mathbf{b}| \cos\theta$, we get:

$$\begin{aligned} |\mathbf{c}| &= |\mathbf{a}| \left[ (\mathbf{a} \bullet \mathbf{b}) / (|\mathbf{a}|\,|\mathbf{b}|) \right] \\ &= (\mathbf{a} \bullet \mathbf{b}) / |\mathbf{b}| \end{aligned}$$

Note that the length of **c** depends on the length of **a** (which we would expect), but not the length of **b** (as **b** gets longer, both the numerator and denominator in the above equation get longer, so the change in **b**'s length cancels out and doesn't affect **c**'s length).

Point-Normal Form

Consider a 2D line L which passes through point A = (Ax, Ay) in direction **c** = (cx, cy).




Sometimes it is useful to find the *normal* vector for the line L (**n** = (nx, ny)), which specifies the direction perpendicular (orthogonal) to the line. We can represent a line in 2D as a point on the line and a vector that's normal (perpendicular) to the line.

**Convince yourself that a point and a normal vector define a line in 2D**. (Does this work in 3D?)

So how can we find a nice equation for a line, given a normal vector for the line and a point on the line?

We know that the dot product of two vectors perpendicular to one another = 0, so **c** • **n** = 0. To obtain an equation for line L, consider some other arbitrary point R = (x,y) on L. The vector R - A (the difference of two points is a vector) must be perpendicular to **n,** so **n** • (R - A) = 0 (remember we learned above that perpendicular vectors have a dot product of 0).

To transform this equation we need to find a way of representing R and A as vectors not points. Try this: replace R with the vector **r** whose initial point is at the origin; replace A with the vector **a** whose initial point is the origin also. This means that the equation we end up with will be dependent on our origin. This is a fact of life when working with lines.

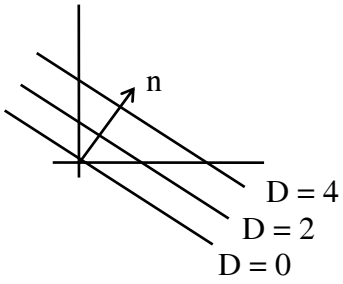
Okay, so now I can take what I know about the line:

**n** • (R - A) = 0

…and re-write it as:

**n** • (**r** - **a**) = 0
**n** • **r** = **n** • **a**
**n** • **r** = D, where D = **n** • **a** = nx * Ax + ny * Ay

This is the *point-normal form* for a line. This equation shows that all points on a straight line, i.e., all vectors from the origin to those points on the line, share the same dot product value with the normal direction, i.e., all these vectors share the same projection onto the normal **n**. The value D reports the "position" of the line, meaning D is altered by shifting the line to a position parallel to itself.




In this diagram, a set of lines **n** • R = D having the same normal (**n** = (1,2)) is shown. As D increases, the lines shift in parallel in the direction of **n**.

This derivation is good to understand, but the take-home message here is:

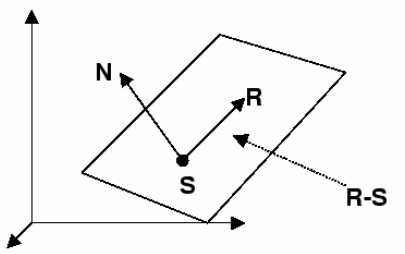
*We can represent a 2d line as* **n** • **r** = D*, where n is a the normal to the line and r is a point on the line. D tells us where along this normal the line lives.*

Point-Normal Form for a Plane

The *real* reason we took you through that was to make it easier to understand the point-normal form for a plane, since most of CS148 is about 3D graphics.

Planes can also be represented in point normal form. A plane is completely specified by giving a single point S = (sx, sy, sz) that lies within it, and the normal direction to the plane. Just as the normal vector orients a line in 2-D, the normal to a plane orients it in space. This normal, **n** = (nx, ny, nz), is perpendicular to any line lying in the plane. For any other arbitrary point in the plane R = (x, y, z) form the vector from R to S which must be perpendicular to **n**: **n** • (R - S) = 0. Replace R and S with **r** and **s** as we did above (these are vectors from the origin to R and S respectively) and – using the same math we used for the 2d case above – you get **n** • **r** = D, where D = **n** • **s**.

This is the point-normal form for a plane. It's the same as the one we defined for a line but it operates on 3-D vectors not 2-D. The basic premise, however, is the same: All points in a plane have the same dot product with the normal, i.e., they all have the same projection onto **n**.




Recall that the equation of a plane is traditionally written as Ax + By + Cz = D where the coefficients A, B, C, D distinguish one plane from another. By calculating out the dot product using the components of the vectors in the equation **n** • **r** = D, we see that the point normal form is actually nx*x + ny*y + nz*z = D. Thus, (A,B,C) of the traditional equation gives the normal direction to the plane.

**Convince yourself that a normal vector and a point in 3D uniquely define a plane.**

Multiplication of vectors: Cross Product

The other way of multiplying vectors gives another vector as a result, this is the *vector product* or *cross product* of two vectors. In 2D, we write this as:

$$\mathbf{V}1 \times \mathbf{V}2 = (V1yV2z - V1zV2y, V1zV2x - V1xV2z, V1xV2y - V1yV2x)$$

There is another nice property associated with the cross-product:

$$\mathbf{V}1 \times \mathbf{V}2 = \mathbf{u} \,|\, \mathbf{V}1 \,|\, |\, \mathbf{V}2 \,|\, \sin\theta \qquad \text{where } 0 \le \theta \le \pi \text{ and } \theta \text{ is the angle between the two vectors}$$

**u** is a *unit vector* of magnitude 1 that is perpendicular to both **V**1 and **V**2.

**The cross product of two vectors is perpendicular to both vectors.** The magnitude of the cross product is equal to the area of the parallelogram defined by the two vectors, but we usually use the cross product in graphics to find a vector that's perpendicular to two known vectors.

Finding the Normal to a Plane

Given what we just learned, if we only know that the plane passes through three specific points, how can we compute the normal **n** to the plane, and the complete plane equation **n** • **r** = D?

We know from geometry that three points specify a plane as long as they do not lie in a straight line (i.e., they are *noncollinear*). To find the normal vector, build two vectors **a** = P2 - P1 and **b** = P3 - P1. Their cross product **a** × **b** must be normal to every line in the plane and is thus the desired vector **n**. Now we can use one of the three points P1, P2 or P3 as the **r** element in **n** • **r** = D to compute D. This then gives the point-normal form for the plane.

**Matrices**

Definitions

A *matrix* is a rectangular array of quantities, called the *elements* of the matrix. We identify matrices according to the number of rows × number of columns.

$2 \times 3$ matrix $\begin{bmatrix} 1 & 3 & 8 \\ 4 & 2 & 9 \end{bmatrix}$

When number of rows = number of columns, we have a *square* matrix. A matrix with a single row or column represents a vector, and a larger matrix can be viewed as a collection of row or column vectors.

Transpose and Inverse

The *transpose* of a matrix $\mathbf{A}^T$ (sometimes written **A'**) is obtained by interchanging rows and columns.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \quad \text{transposes to} \quad \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

The matrix B is the *inverse* of a matrix A if AB = I where I is the identity matrix, being all 0's except for 1's on the diagonal. Only square matrices have inverses, and not all square matrices have inverses. Computing a matrix inverse is generally a pain; there is no simple, general-purpose formula for doing this.

Multiplication

To multiply a matrix **A** by a scalar value s, we just multiply each element of the matrix by s. Matrix addition is defined only for matrices with the same number of rows and columns; two matrices are added by simply adding corresponding elements.

We can multiply an m × n matrix **A** by a p × q matrix **B** only if n = p. We obtain an m × q matrix **C** whose elements are calculated as in the following example:

$$\begin{bmatrix} 2 & 3 \\ 4 & 5 \\ 1 & 4 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 3 \end{bmatrix} = \begin{bmatrix} 2*7+3*9 & 2*8+3*3 \\ 4*7+5*9 & 4*8+5*3 \\ 1*7+4*9 & 1*8+4*3 \end{bmatrix} = \begin{bmatrix} 26 & 25 \\ 73 & 40 \\ 43 & 20 \end{bmatrix}$$

We multiplied a $3 \times 2$ matrix by a $2 \times 2$ matrix and ended up with a $3 \times 2$ matrix.

More formally, the element at (i,j) in the resulting matrix is the result of taking the dot product between the $i^{th}$ row in **A** and the $j^{th}$ column in **B**. Ponder that sentence some and make sure it makes sense; if you see how it relates to the above example, you're good to go.

If we use matrix notation to describe two vectors, multiplication produces the same result as the dot product, as long as we assume that the first vector is a row vector and the second is a column vector. In other words, if I have two vectors A = (1,2,3) and B = (4,5,6), I can write their dot product in matrix notation as:

$$\begin{bmatrix}1 & 2 & 3\end{bmatrix}\begin{bmatrix}4\\5\\6\end{bmatrix} = 1*4 + 2*5 + 3*6 = 32$$

We multiplied a $1 \times 3$ matrix by a $3 \times 1$ matrix, and we got a $1 \times 1$ matrix, so all the rules about matrix multiplication hold, and we can see that this is the same as taking the dot product between these vectors.

*These notes were adapted from notes by Sean Walker, which were adapted from notes by Maggie Johnson. Ah, how the circle of life spins ever onward.*