

# Planes and Half-Spaces

*Author: Max Wagner, mwagner@digipen.edu*

## Pre-Reqs

Familiarity with terms and concepts from Linear Algebra, including the ideas behind vectors and vector spaces.

## Notation

Vectors AND points in bold, i.e.  $\mathbf{u}$  or  $\mathbf{n}$ . I leave it to the reader to discern from the context whether the object is a point or a vector. Scalars will be weighted normally, that is un-bolded. Often I will refer to a vector  $\mathbf{u}$ , and then shortly afterward refer to its components as  $u_x, u_y, u_z$ , etc.

## What is a Plane?

Intuitively, we all know what a plane is, it's a big flat thing lying in space. It goes on forever, and there's a thing called a "normal" which we all know to be an important cousin of the plane.

A plane can be described as a two-dimensional vector space. A plane allows two degrees of freedom, and two vectors make up a basis of a plane. That is to say, a plane is any linear combination of any two vectors  $\mathbf{u}$  and  $\mathbf{v}$ , where  $\mathbf{u}$  and  $\mathbf{v}$  are linearly independent. Think of the standard x-axis as  $\mathbf{u}$ , and the standard y-axis as  $\mathbf{v}$ ; then, all points in this plane can be described using the ordered pair  $(x, y)$ , which really is just a shorthand for the linear combination  $x\mathbf{u} + y\mathbf{v}$ .

A plane can live in N-dimensional spaces of course, where  $N > 1$ , and the planes that reside in vector spaces for  $N > 2$  will simply be sub-spaces of the larger vector space.  $\mathbf{R}^2$  of course is itself a plane, and planes in  $\mathbf{R}^3$  will be what interests us in this paper.

In  $\mathbf{R}^3$ , the plane normal is defined as the orthogonal complement of the plane. This means that it is another sub-space of  $\mathbf{R}^3$  none of whose elements reside in the plane, and vice versa (none of the elements of the plane's sub-space reside in the sub-space of the plane normal). How's that? The normal is a vector sub-space? Indeed it is, a one dimensional sub-space of  $\mathbf{R}^3$ , or, in other words, all scalar multiples of a 3D vector, where none of these vectors lie in the plane.

Usually, in computer graphics, when we say "normal", we will be talking about a single vector, the outwardly facing plane normal of unit length. This normal can be thought of as "perpendicular" to the plane, if this term is more familiar.

## How do we represent a plane?

One standard way of representing a plane is with the Point-Normal equation, which looks like

$$\mathbf{n} \cdot (\mathbf{x} - \mathbf{p}) = 0 \quad \text{EQU 1}$$

for all points  $\mathbf{x}$  that lie in the plane, where  $\mathbf{n}$  is the plane normal vector and  $\mathbf{p}$  is a (fixed) point known to lie in the plane. Above I described the normal to the plane and the plane itself as *orthogonal complements*. This equation falls out immediately from the definition of orthogonality, which holds that any vectors  $\mathbf{u}$  and  $\mathbf{v}$  such that  $\mathbf{u} \cdot \mathbf{v} = 0$  are said to be orthogonal. In this case, the vectors are  $\mathbf{n}$  and  $(\mathbf{x} - \mathbf{p})$ . This equation also follows naturally from a geometric understanding of the plane and the dot product.

This equation is not always the most natural to use in computer graphics, however. Another, perhaps more common form is

$$\mathcal{A}x + \mathcal{B}y + \mathcal{C}z + \mathcal{D} = 0 \quad \text{EQU 2}$$

for a point  $(x, y, z)$  that lies in the plane. What are  $\mathcal{A}$ ,  $\mathcal{B}$ ,  $\mathcal{C}$ , and  $\mathcal{D}$ ? It turns out that EQU 2 can be derived very naturally from EQU 1, which will lead us to understand these coefficients that were most likely imparted to us way back when somewhere in middle or high school when we couldn't have cared less.

Let's expand EQU 1:

$$\begin{aligned} \mathbf{n} \cdot (\mathbf{x} - \mathbf{p}) &= 0 \\ (\mathbf{n} \cdot \mathbf{x}) - (\mathbf{n} \cdot \mathbf{p}) &= 0 \end{aligned} \quad \text{(EQU 1)}$$

Which, component-wise, is:

$$(n_x x_x + n_y x_y + n_z x_z) - (n_x p_x + n_y p_y + n_z p_z) = 0$$

We have now expressed EQU 2 in terms of the names of EQU 1. Fortunately, we understand what the entities in EQU 1 are. The mystery behind the coefficients of EQU 2 is solved:

$$\begin{aligned} \mathcal{A} &= n_x \\ \mathcal{B} &= n_y \\ \mathcal{C} &= n_z \\ \mathcal{D} &= -(n_x p_x + n_y p_y + n_z p_z) = -(\mathbf{n} \cdot \mathbf{p}) \end{aligned}$$

These values are constant for a plane, and require knowledge only of the plane normal and a point (any point) in the plane. Given the normal and a point in the plane, we can compute  $\mathcal{D}$ , and then keep only these four values (the components of the normal  $\mathbf{n}$  and  $\mathcal{D}$ ) to describe the plane. For the rest of this paper, I will be assuming this representation (as a 4D vector of sorts).

## Who cares?

So whooppy-do, we have a plane described by four numbers and we know how to get these numbers. What do we do with this? Of course there are many answers. Here is a short list off the top of my head:

- Ray Casting
- Polygon Collisions/Intersections
- Reflections
- Clipping
- Frustum Culling
- BSP Trees

Planes are truly a fundamental component of physical simulations, and, I assume that if you're reading this you care about physical simulations. Otherwise you can go away and stick your head in a bucket or something.

## Applications: Line/Plane Intersection

How do we compute the intersection of a line and a plane? This is of course relevant to ray-casting and derivative applications.

First, let's formulate our line parametrically as all points  $\ell(t)$ , such that

$$\ell(t) = \mathbf{r} + t\mathbf{d}$$

for origin  $\mathbf{r}$ , direction  $\mathbf{d}$ , and all scalars  $t$ . If we wanted to limit our line to a ray, we would say for all  $t \geq 0$ .

Using the Point-Normal equation from above (EQU 1), we can plug in our equation for a line into the unknown variable  $\mathbf{x}$  and solve for the new unknown variable:  $t$ .

$$\begin{aligned} \mathbf{n} \cdot (\mathbf{x} - \mathbf{p}) &= 0 && \text{(EQU 1)} \\ \mathbf{n} \cdot (\ell(t) - \mathbf{p}) &= 0 \\ \mathbf{n} \cdot (\mathbf{r} + t\mathbf{d} - \mathbf{p}) &= 0 \\ \mathbf{n} \cdot \mathbf{r} + \mathbf{n} \cdot t\mathbf{d} - \mathbf{n} \cdot \mathbf{p} &= 0 \\ \mathbf{n} \cdot \mathbf{r} + t(\mathbf{n} \cdot \mathbf{d}) - \mathbf{n} \cdot \mathbf{p} &= 0 \\ t = (\mathbf{n} \cdot \mathbf{p} - \mathbf{n} \cdot \mathbf{r}) / (\mathbf{n} \cdot \mathbf{d}) \end{aligned}$$

Above I said that all we'd need were our 4 plane equation coefficients  $\mathcal{A}$ ,  $\mathcal{B}$ ,  $\mathcal{C}$ , and  $\mathcal{D}$ , and that we'd be able to throw out the point-on-the-plane that we've been calling  $\mathbf{p}$ . Was I lying? No. We know that  $\mathbf{n} = (\mathcal{A}, \mathcal{B}, \mathcal{C})$ , and we know that  $\mathcal{D} = -(\mathbf{n} \cdot \mathbf{p})$ . Using these substitutions, we can rewrite the above as

$$t = -(\mathbf{n} \cdot \mathbf{r} + \mathcal{D}) / (\mathbf{n} \cdot \mathbf{d})$$

$$t = -(Ar_x + Br_y + Cr_z + \mathcal{D}) / (Ad_x + Bd_y + Cd_z) \quad \text{EQU 3}$$

As is always the case in a programming environment, when we divide, we must ensure that we are not dividing by zero. Provided the denominator of EQU 3 is not zero, then the point of intersection between a point and a plane will be given as  $\ell(t)$ , where  $t$  is determined using EQU 3. If the denominator is zero, then the line and the plane are either coincident (the line lies entirely in the plane) or “parallel” (the line never intersects the plane). Usually we lump these two cases into one and say that the line does not intersect the plane if the denominator is zero. If you cared to separate these two cases, then you could simply take any point on the line and determine if it is in the plane or not in the plane.

If we are testing for the intersection between a ray and a plane, then we must additionally check to see that  $t \geq 0$ .

## Applications: Half-Spaces and Signed Distance to a Plane

There comes a time in every simulation programmer’s life when you must decide which side of the plane a point is on (or indeed, whether a point lies in a plane). We say that a plane divides 3D space into two “half-spaces”. Some people refer to these as the “front” and the “back” half-spaces; I prefer the terminology of “positive” and “negative” half-spaces, as this relates more clearly to the geometry.

Intuitively, this idea of half-spaces is very clear; you can visualize a plane, and you can visualize the space above it and the space below it. But if I were to give you an arbitrary point in 3D space and 4 plane coefficients, how would you classify the point with respect to the plane?

### Example: Building the Plane Equation from three triangle vertices

Let’s look at an example, starting with a triangle whose vertices we will call  $\mathbf{v0}$ ,  $\mathbf{v1}$ , and  $\mathbf{v2}$  (these are just 3D points). This triangle defines a plane, but we must do a little work to extract our preferred plane representation.

First we’ll get the plane normal  $\mathbf{n}$ . Let’s call the vectors between the triangle vertices the triangle walls, and denote them as  $\mathbf{w0} = \mathbf{v1} - \mathbf{v0}$ ,  $\mathbf{w1} = \mathbf{v2} - \mathbf{v1}$ , and  $\mathbf{w2} = \mathbf{v0} - \mathbf{v2}$ . Now we have three vectors which are known to lie in the same plane, so we can construct the plane normal using the cross product of any two of these. Recall that the cross product is non-associative, which means that depending on which vectors you use and in which order you perform their cross-product, the direction of the normal will be “up” or “down”. One or the other is not necessarily correct, you usually just want to be consistent and know which is expected in your programming environment. In a right-

handed coordinate system, where triangles are commonly ordered counter-clockwise, an “outwardly-facing” normal would be defined as  $\mathbf{n} = \mathbf{w0} \times \mathbf{w1}$ .

So we have our normal, a.k.a.  $\mathcal{A}$ ,  $\mathcal{B}$ , and  $\mathcal{C}$ . Recall that to get  $\mathcal{D}$ , we need the normal and a point known to lie in the plane. So we can take any of our three original triangle vertices (let’s just use  $\mathbf{v0}$ ), yielding  $\mathcal{D} = -(\mathbf{n} \cdot \mathbf{v0})$ . And voila, we have our plane.

### Deriving the Signed Distance of a Point to a Plane

So now that we have our plane, how can we perform the “Half-Space Test”? First we must define the concept of a point’s signed distance to a plane.

When we talk about the distance of a point to a plane, we are usually talking about the distance of the shortest path between the point and the plane (or for a line; the following applies to both cases). This shortest path is found by traveling along a line which runs orthogonally to the plane and intersects the point in question; in other words, by traveling from the point in the direction of the normal vector (or in the negative direction of the normal vector). Alternatively, this measurement can be thought of as the distance of the vector formed by the point and the point’s orthogonal projection onto the plane.

But how can we compute this quantity? And why do we call it the “signed” distance?

Let us consider the case of a plane that intersects the origin of the world. Notice that all points on the plane are, when treated as vectors, orthogonal to the plane normal. Recall that in the general case, a point (when treated as a vector) is not by itself orthogonal to the plane normal; rather, the vector yielded by the *subtraction* of *two* points in the plane would be orthogonal to the plane normal. That is, for plane normal  $\mathbf{n}$ , world origin  $\mathbf{o} = (0,0,0)$ , and all points in the plane  $\mathbf{x}$ ,  $\mathbf{n} \cdot (\mathbf{p} - \mathbf{o}) = 0$ ; but this is just the same as  $\mathbf{n} \cdot \mathbf{p} = 0$ . An immediate consequence of this fact is that the quantity  $\mathcal{D}$  in the plane equation for all planes through the origin is just 0 (recall that  $\mathcal{D} = -(\mathbf{n} \cdot \mathbf{p})$ ).

Now consider a point  $\mathbf{x}$  in the world, not on the plane. We want to compute the distance from this point to its orthogonal projection onto the plane. But because the plane intersects the origin, we can treat this point as a vector by subtracting the origin, and then projecting this vector onto the plane normal. Recall that the projection of a vector  $\mathbf{u}$  onto a vector  $\mathbf{v}$  is given by  $[(\mathbf{u} \cdot \mathbf{v})/(\mathbf{v} \cdot \mathbf{v})] \mathbf{v}$ . For unit vector  $\mathbf{v}$ , this becomes just  $(\mathbf{u} \cdot \mathbf{v}) \mathbf{v}$ , and hence the length of this projected vector is just  $(\mathbf{u} \cdot \mathbf{v})$ . Using the notation above for a plane through the origin with *unit* normal  $\mathbf{n}$  and a vector  $\mathbf{x}$  which intersects the plane at the origin, the orthogonal distance is given by the quantity  $(\mathbf{n} \cdot \mathbf{x})$ . This quantity is signed: in the case that the angle between the two vectors is in the range 0 to 90 (or -90 to 0) degrees, it will be positive, and negative if the angle is in the range 90 to 180 (or -180 to -90). This is the reason for the terminology of “signed” distance, and why I prefer to use the terms “positive” and “negative” half-spaces; it refers directly to the direction of the normal vector and the location of the point with respect to this normal vector.

How can we extend the above derivation to be fully general for all planes, instead of just those intersecting the origin? Essentially, the problem is the same, in that we want to project a vector to the plane normal and use its length. But we can't just use the point (converted to a vector) by itself anymore, as the point (as a vector) no longer intersects the plane at the origin. Instead, we need to "clip" the point (as a vector) where it intersects the plane, and then project this clipped vector to the plane normal. But that sounds like a lot of work.

Instead, let's try using exactly the same approach as we did for the plane through the origin. That is, take our point  $\mathbf{x}$  and subtract the origin (making it a vector), and then project this vector to the plane normal, and call this new projected vector  $\mathbf{x}'$ . Now, for argument's sake, imagine that we had computed the intersection of the vector  $\mathbf{x}$  and the plane. We could then project this point (as a vector) to the plane normal, call it  $\mathbf{v}$ , and subtract  $\mathbf{v}$ 's length from  $\mathbf{x}'$ . But wait! The intersection of a line with a plane is of course in the plane; and the projection of this point (again, as a vector), would be given by  $\mathbf{n} \cdot \mathbf{p}$ . But that's just the quantity  $-\mathcal{D}$  (a constant for the plane)! In other words, we can compute  $\mathbf{x}'$  as before (in the case of a plane through the origin), and then just add  $\mathcal{D}$  to compensate for the shift of the plane away from the origin. This sheds some light onto the nature of the magic number  $\mathcal{D}$ ;  $\mathcal{D}$  represents the negative length of the projection of any point in the plane onto the plane's normal vector.

Thus, we can express the signed distance  $s$  of a point  $\mathbf{x}$  to a plane  $(\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D})$  as

$$s = \mathcal{A}x_x + \mathcal{B}x_y + \mathcal{C}x_z + \mathcal{D} \quad \text{EQU 4}$$

### Normalizing the Plane Vector

It is important to note that throughout the above derivation, we have been using a unit normal vector. Is this required? If you only want to know which half-space a point resides in, the answer is no; the sign of the distance will be unaffected, and will still correctly indicate when a point is in the plane (distance equal to 0). However, if you want a true and accurate distance in world units, the plane vector must first be normalized.

Fortunately, this is no more complicated than normalizing a vector, though with a subtle twist. To normalize the plane vector, you must divide all four coefficients  $(\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D})$  by the length of the normal vector. Alternatively, when creating the plane vector from a normal vector and a point on the plane, you can normalize the normal vector before taking its dot product with the point on the plane.

Normalizing the plane equation can become important when you are performing frustum culling. Though normalizing a vector is considered expensive (the square root involved),

it is usually considered as sort of a “pre-process” for a plane, as it is performed only once for duration of the plane’s existence.

## Applications: Transforming Normals and Plane Vectors

Lighting calculations require surface normals and/or vertex normals. These normals will usually be specified in object space, but will be need to be transformed to world space for the actual lighting equations (as the lights live in the world space). It might come as a surprise to some, and seem un-intuitive, that these normals should not be transformed in the same way a vertex’s position should. Specifically, the matrix that transforms your positions into the world is not necessarily the same matrix you should use to transform your normal vectors.

Let’s assume we have two vectors, the normal vector  $\mathbf{n}$  and a vector known to lie in the plane,  $\mathbf{v}$ . These vectors are, by definition, orthogonal. We will say that these vectors are specified in some arbitrary object space.

Now let us say we want to transform the vector  $\mathbf{v}$  by some matrix  $\mathcal{M}$ , yielding  $\mathbf{v}' = \mathcal{M}\mathbf{v}$ . We can consider this as a transformation on the plane. Naturally, this transformed plane will have a new normal vector, which, by definition, is still orthogonal to  $\mathbf{v}'$ . Let’s call the normal’s transformation  $\mathcal{T}$ . Thus we have that  $\mathbf{v}' \cdot \mathcal{T}\mathbf{n} = 0$ , which is equivalent to  $\mathcal{T}\mathbf{n} \cdot \mathcal{M}\mathbf{v} = 0$ . Thus we need to find a matrix  $\mathcal{T}$  that will make this statement true.

The trick here is to treat the vectors as column matrices, and then continue with standard matrix arithmetic. But how do we convert the dot product operator to matrix arithmetic? By transposing the left operand and performing regular matrix multiplication. The original condition of orthogonality becomes:

$$\mathbf{n} \cdot \mathbf{v} = \mathbf{n}^T \mathbf{v} = 0$$

Then, after transformation:

$$\begin{aligned} (\mathcal{T}\mathbf{n})^T \mathcal{M}\mathbf{v} &= 0 \\ \mathbf{n}^T \mathcal{T}^T \mathcal{M}\mathbf{v} &= 0 \end{aligned}$$

It is up to us to make this equation true. So let’s require that  $\mathcal{T}^T \mathcal{M} = I$  (where  $I$  is the identity matrix), yielding

$$\begin{aligned} \mathbf{n}^T I \mathbf{v} &= 0 \\ \mathbf{n}^T \mathbf{v} &= 0 \end{aligned}$$

which we know to be true, as it was our starting definition.

Now we have

$$\begin{aligned}T^T M &= I \\T^T &= M^{-1} \\T &= (M^{-1})^T\end{aligned}$$

Thus we have derived the correct matrix to transform surface normals. The nice part is that, as long as our original transformation matrix  $M$  is a pure rotation matrix, then  $(M^{-1})^T = M$ , as the inverse of a rotation matrix is the transpose. Thus many game engines impose the restriction of allowing only rotational and translational transformations for the majority of the world objects, for obvious performance reasons.

It turns out that transforming plane vectors works identically, though the plane vector is treated as a 4D vector instead of a 3D vector. The derivation is identical to the above, but instead of using the plane normal and a vector in the plane, you use the 4D plane vector and a point in the plane (as the result of a dot product with a plane vector and a 4D point in the plane should also be zero). Transforming plane vectors is useful for frustum culling (see my notes on Frustum Culling).