Vectors Part 3

By Alexei Novikov

A long time ago in a galaxy, far far away...

Here's the last episode of the "Vectors" trilogy. This one will be about enlightening and turning to the light side. Er... I mean lighting calculation. This is quite a feat (an amazing feat, a death-defying feat, well, not so death defying, really, a dangerous feat, no, not so dangerous at all, an easy feat, but exciting!), yet it doesn't go beyond the vector math we discussed before. Have you got a helmet? And, um.. you've played "The Secret of Monkey Island", haven't you?

So here we go. Let's analyze the task. What do we need to do to calculate an amount of light delivered to a given point by a give light source?

First, we need some formula for finding the lighting at a given position relative to the light assuming the light is not blocked. Theoretically there can be many types of lights, for which this process would be different - point light, directional light, spotlights, etc. But since JK's way of storing light values per vertex does not allow sharp and precise shadows, it really isn't worth it to have many types of lighting. So JED uses only non-directional point lights, with very simplistic way of calculating the lighting. Which is: if the given point is outside light's range, it isn't lit. If it's inside light's range, the lighting is:
Intensity=Light's Intensity*square((Light's range-distance)/Light's range)
I.e. - quadratic decay.

Second, we need to check if the light is visible from this point. And that's the interesting part. You might say that it's all about seeing the light.

What does that mean in geometrical terms? A given surface will block a given light from a give point if the vector point->light intersects the surface's plane within the surface. See the picture at the right. So this basically boils down to two tasks:

- 1) Finding an
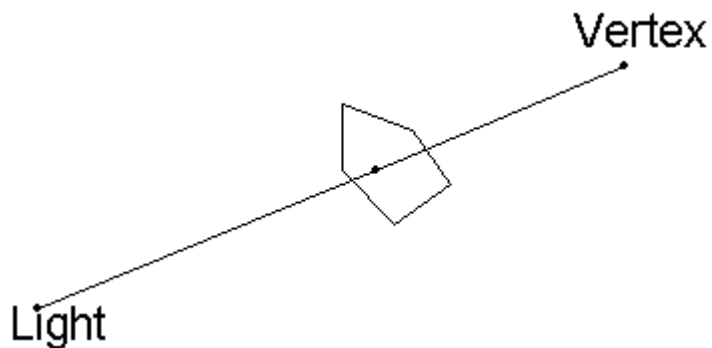
- intersection of a line segment and a plane.
- 2) Check that intersection lies within the surface

So, intersection of a surface and a line segment. How would we approach this? For the input data we have a surface as a set of vertices actually we'll need just one so far) and its normal and two points that define line segment. Let's call them S0-SN (vertices of the surface), N (surface's normal), L1 and L2 (beginning and ending points of the line segment). Remember that earlier we found that dot product of plane's normal and and a difference vector from a given point to any point on the plane is the distance from that point to the plane? And, importantly, a signed distance - it's positive on one side of the plane and negative on another. Does that give you any ideas? Let's find the dsitance from both line ends to the plane:
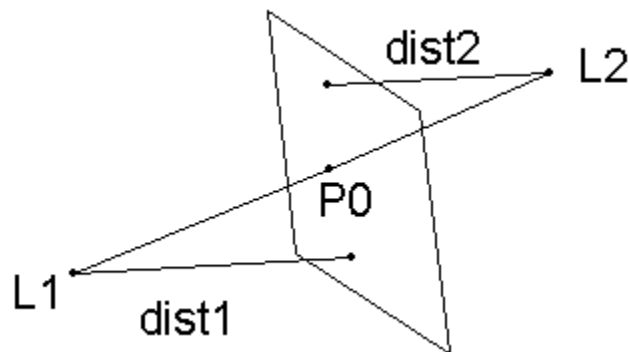
dist1=N*(L1-S0)
dist2=N*(L2-S0)

Obviously if the line segment intersects the plane, one distance is negative, the other is positive (well, there are also marginal cases where one or both of them is 0). This quick check can be done to eliminate obviously non-intersecting pairs.

Now let's expand on this a bit. Our line is defined by points L1 and L2, or point L1 and vector L2-L1. The equasion of this line will be:

P=L1+A*(L2-L1)

When A=0, we get point L1, when A=1 we get point L2. So the point we want to find corresponds to A somewhere in 0..1. Let's call it A0. The meaning of this A parameter is:

A=Length(P-L1)/Length(L2-L1)

Now check this out. As A goes from 0..1, distance from the plane goes from dist1 to dist2. We are looking for the point where the distance is 0. You can easily see that dependence of dist from A is linear (can you?). Hence, we can write down:

dist=P*A+Q

Since at A=0, dist=dist1

dist1=P*0+Q
Q=dist1

and at A=1, dist=dist2

dist2=P*1+dist1
P=(dist2-dist1)

So the equasion is:

dist=(dist2-dist1)*A+dist1

Remember that we're looking for point with A=A0 where dist=0. So:

0=(dist2-dist1)*A0+dist1
A0=dist1/(dist1-dist2)

Now remember our equasion for the line:

P=L1+A*(L2-L1)

So, puttin A0 that we found, we now have the solution:

P0=L1+(dist1/(dist1-dist2))*(L2-L1)

One down, one more to go.

The next task is checking whether the intersection lies on the surface. This can be a rather hairy task for a general type of surface, but it's greatly simplified by the fact that all surfaces in JK must be convex. What convex means in geometrical sense is relative to

every edge of the surface all other vectices of the surface are "on the right side". In 3 dimensional case "to the right" would be defined as "the cross product of the edge and vector from a given point to the beginning of the edge points in the same direction as the normal". Look back to the first vector article for "right hand rule" for finding cross-product to see it clearer.

So we can claim this: if the above isn't true, the point is outside the surface. A quick look at this allows us also to claim that if it's true for all edges, then the point is within surface. Check the picture. Here the two examples. Left - the poin P0 is inside the surface, and the pair of vectors (P0-S3) and (S4-S3) are in "right" relation, while on the right the point P0 is outside the surface and those vectors are in the "left" relation.
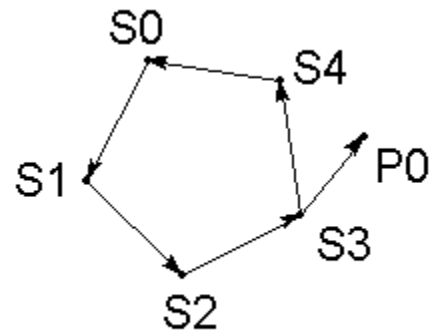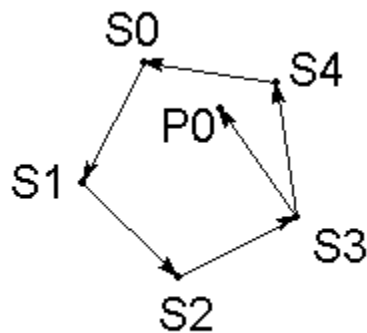
Ah, and actually one more minor thing: how would you check if two vectors (the normal and the cross product of the edge and point-beginnning of the edge vector) point in the same or opposite directions? That's trivial. Simply check the sign of their dot product. Let say we have vectors V1 and V2, since:

$$V1*V2=V1.X*V2.X+V1.Y*V2.Y+V1.Z*V2.Z$$

if vectors point in the same direction, the dot product is ALWAYS positive. Why? because if they are, their respective components will have the same sign. If they are positive, the result of their multiplication is positive. If they are both negative, the result of their multiplication is still positive. The sume of 3 positive numbers is a positive number. Likewise, if they point in opposite direction, their respective components have opposite sign and we have a sum of three negative numbers, which is always negative.

That about sums it up. Of course there are some details with the algorithm, marginal cases (if you ignore them, all the lights will be "blocked" by adjacent surfaces. That's what I got when I first implemented this thing), room for optimization (I sped up lighting calculation about 7+ times from the initial state), etc. But that's the gist.

I hope those vector articles have bee nuseful to you and helped you understand some of this stuff and see how it applies to some real tasks and you had fun doing that. I know I did.

So, with the world saved and secrets of DaVinci protected, Eddie finally had his coffee.

Till the next time,
Alex.
PS. While there are certainly many more cool applications of vector math, this is the end of the "trilogy". However, there can always be a prequel :-).