The Basics of Dot Product in Unity URP. Vectors, Angles, and Lighting!

I wanted to make my video transcripts publicly available. Note, they might not be one-to-one with the dialogue in the video, but will be pretty close. Thank you!

Hi, I'm Ned and I make games! Today, I want to introduce you to a useful function called the dot product. You might have learned about it in math class, and it has some surprising applications in game programming!

Let's look at an implementation. The dot product takes in two vectors and returns a float. The formula is pretty simple: multiply the matching components of each vector and add everything together. Doesn't seem very useful, right? But, it turns out, this formula also equals the length of each vector times the cosine of the angle between the vectors.

By rearranging, you can get a few useful functions just from this fact, like the length of a vector and the angle between two vectors. When efficiency matters, consider using the dot product of a vector with itself instead of it's length. This avoids a square root operation!

Here's a graph showing how the dot product changes as I move two vectors around. Notice when both vectors are unit vectors, meaning their lengths equal one, their dot product is one when they're pointing in the same direction, zero when they're perpendicular, and negative one when they're parallel but opposite.

OK, so how can you use the dot product? In the shader graph, Unity provides a dot node. There's also the built in dot function in hlsl code. In C#, you can use Vector2.dot, Vector3.dot or the new mathematics package's math.dot. They're all equivalent.

What are some uses of the dot product? In shaders, use it to calculate lighting. Diffuse lighting takes the dot product of the light direction and the surface normal to see how much light affects a pixel. Specular lighting is similar, except it uses the dot product of a reflected light approximation and the view direction. These calculations are the foundation of most lit shaders!

The dot product is also useful for working with physics. For instance, when a projectile reaches max height, the dot product of its velocity direction and acceleration direction will equal zero. Additionally, on collision, take the dot product of its velocity and the impact surface's normal to approximate the amount of damage it might do!

In gameplay, the dot product provides an easy way to test if an object is in a character's field of view. Take the dot product of the character's view direction and a unit vector pointing towards the hiding object. If the result is greater than some threshold, the character sees the object, or at

least you know it’s worth executing a raycast test! This technique also tells you if an object is behind another - the dot product will be negative!

And that just about scratches the surface of the dot product. I hope this introduction has given you some ideas about how to utilize this useful little function. Please tell me about them in the comments!

Thanks so much for watching! If you like game development, consider subscribing - I post weekly content here, including shader tutorials. I’d also really appreciate it if you could like this video - it encourages YouTube to recommend it and really helps me out.

I also want to thank my patrons for their continued support, and give a shout out to Leafenzo, my next-gen patron. Thank you so much! If you’d like early viewing of videos, downloadable project files, or voting power in topic polls, consider joining my Patreon. But, don’t feel pressured, I appreciate you watching at all!

Thanks again for watching, and make games!