Gradient of cross product

Taking two vectors, we can write every combination of components in a grid: This completed grid is the outer product, which can be separated into the: Dot product, the interactions between similar dimensions (x*x, y*y, z*z) Cross product, the interactions between different dimensions (x*y,y*z, z*x, etc.) The dot product ($\vec{a} \cdot \vec{b}$) measures similarity because it only accumulates interactions in matching dimensions. It's a simple calculation with 3 components. The cross product (written $\vec{a} \times \vec{b}$) has to measure a half-dozen "cross interactions". The calculation looks complex but the concept is simple: accumulate 6 individual differences for the total difference. Instead of thinking "When do I need the cross product?" think "When do I need interactions between different dimensions?". Area, for example, is formed by vectors pointing in different directions (the more orthogonal, the better). Indeed, the cross product measures the area spanned by two 3d vectors (source): (The "cross product" assumes 3d vectors, but the concept extends to higher dimensions.) Did the key intuition click? Let's hop into the details. Defining the Cross Product The dot product represents the similarity between vectors as a single number: For example, we can say that North and East are 0% similar since $(0, 1) \cdot (1, 0) = 0$. Or that North and Northeast are 70% similar ($\cos(45) = .707$, remember that trig functions are percentages.) The similarity shows the amount of one vector that "shows up" in the other. Should the cross product, the difference between vectors, be a single number too? Let's try. Sine is the percentage difference, so we could write: Unfortunately, we're missing some details. Let's say we're looking down the x-axis: both y and z point 100% away from us. A number like "100%" tells us there's a big difference, but we don't know what it is! We need extra information to tell us "the difference between $\vec{x}$ and $\vec{y}$ is this" and "the difference between $\vec{x}$ and $\vec{z}$ is that". So, let's express the cross product as a vector: The size of the cross product is the numeric "amount of difference" (with $\sin(\theta)$ as the percentage). By itself, this doesn't distinguish $\vec{x} \times \vec{y}$ from $\vec{x} \times \vec{z}$. The direction of the cross product is based on both inputs: it's the direction orthogonal to both (i.e., favoring neither). Now $\vec{x} \times \vec{y}$ and $\vec{x} \times \vec{z}$ have different results, each with a magnitude indicating they are "100%" different from $\vec{x}$. (Should the dot product be a vector result too? Well, we're tracking the similarity between $\vec{a}$ and $\vec{b}$. The similarity measures the overlap between the original vector directions, which we already have.) Geometric Interpretation Two vectors determine a plane, and the cross product points in a direction different from both (source): Here's the problem: there's two perpendicular directions. By convention, we assume a "right-handed system" (source): If you hold your first two fingers like the diagram shows, your thumb will point in the direction of the cross product. I make sure the orientation is correct by sweeping my first finger from $\vec{a}$ to $\vec{b}$. With the direction figured out, the magnitude of the cross product is $|a| |b| \sin(\theta)$, which is proportional to the magnitude of each vector and the "difference percentage" (sine). The Cross Product For Orthogonal Vectors To remember the right hand rule, write the xyz order twice: xyzxyz. Next, find the pattern you're looking for: xy => z (x cross y is z) yz => x (y cross z is x; we looped around: y to z to x) zx => y Now, xy and yx have opposite signs because they are forward and backward in our xyzxyz setup. So, without a formula, you should be able to calculate: Again, this is because x cross y is positive z in a right-handed coordinate system. I used unit vectors, but we could scale the terms: Calculating The Cross Product A single vector can be decomposed into its 3 orthogonal parts: When the vectors are crossed, each pair of orthogonal components (like $a_x \times b_y$) casts a vote for where the orthogonal vector should point. 6 components, 6 votes, and their total is the cross product. (Similar to the gradient, where each axis casts a vote for the direction of greatest increase.) xy => z and yx => -z (assume $\vec{a}$ is first, so xy means $a_x b_y$) yz => x and zy => -x zx => y and xz => -y xy and yx fight it out in the z direction. If those terms are equal, such as in $(2, 1, 0) \times (2, 1, 1)$, there is no cross product component in the z direction (2 – 2 = 0). The final combination is: where $\vec{n}$ is the unit vector normal to $\vec{a}$ and $\vec{b}$. Don't let this scare you: There's 6 terms, 3 positive and 3 negative Two dimensions vote on the third (so the z term must only have y and x components) The positive/negative order is based on the xyzxyz pattern If you like, there is an algebraic proof, that the formula is both orthogonal and of size $|a| |b| \sin(\theta)$, but I like the "proportional voting" intuition. Example Time Again, we should do simple cross products in our head: Why? We crossed the x and y axes, giving us z (or $\vec{i} \times \vec{j} = \vec{k}$, using those unit vectors). Crossing the other way gives $-\vec{k}$. Here's how I walk through more complex examples: Let's do the last term, the z-component. That's (1)(5) minus (4)(2), or 5 – 8 = -3. I did z first because it uses x and y, the first two terms. Try seeing (1)(5) as "forward" as you scan from the first vector to the second, and (4)(2) as backwards as you move from the second vector to the first. Now the y component: (3)(4) – (6)(1) = 12 – 6 = 6 Now the x component: (2)(6) – (5)(3) = 12 – 15 = -3 So, the total is $(-3, 6, -3)$ which we can verify with Wolfram Alpha. In short: The cross product tracks all the "cross interactions" between dimensions There are 6 interactions (2 in each dimension), with signs based on the xyzxyz order Appendix Connection with the Determinant You can calculate the cross product using the determinant of this matrix: There's a neat connection here, as the determinant ("signed area/volume") tracks the contributions from orthogonal components. There are theoretical reasons why the cross product (as an orthogonal vector) is only available in 0, 1, 3 or 7 dimensions. However, the cross product as a single number is essentially the determinant (a signed area, volume, or hypervolume as a scalar). Connection with Curl Curl measures the twisting force a vector field applies to a point, and is measured with a vector perpendicular to the surface. Whenever you hear "perpendicular vector" start thinking "cross product". We take the "determinant" of this matrix: Instead of multiplication, the interaction is taking a partial derivative. As before, the $\vec{i}$ component of curl is based on the vectors and derivatives in the $\vec{j}$ and $\vec{k}$ directions. Relation to the Pythagorean Theorem The cross and dot product are like the orthogonal sides of a triangle: For unit vectors, where $|a| = |b| = 1 $, we have: I cheated a bit in the grid diagram, as we have to track the squared magnitudes (as done in the Pythagorean Theorem). Advanced Math The cross product & friends get extended in Clifford Algebra and Geometric Algebra. I'm still learning these. Cross Products of Cross Products Sometimes you'll have a scenario like: First, the cross product isn't associative: order matters. Next, remember what the cross product is doing: finding orthogonal vectors. If any two components are parallel ($\vec{a}$ parallel to $\vec{b}$) then there are no dimensions pushing on each other, and the cross product is zero (which carries through to $0 \times \vec{c}$). But it's ok for $\vec{a}$ and $\vec{c}$ to be parallel, since they are never directly involved in a cross product, for example: Whoa! How'd we get back to $\vec{j}$? We asked for a direction perpendicular to both $\vec{i}$ and $\vec{j}$, and made that direction perpendicular to $\vec{i}$ again. Being "doubly perpendicular" means you're back on the original axis. Dot Product of Cross Products Now if we take what happens? We're forced to do $\vec{a} \times \vec{b}$ first, because $\vec{b} \cdot \vec{c}$ returns a scalar (single number) which can't be used in a cross product. If $\vec{a}$ and $\vec{c}$ are parallel, what happens? Well, $\vec{a} \times \vec{b}$ is perpendicular to $\vec{a}$, which means it's perpendicular to $\vec{c}$, so the dot product with $\vec{c}$ will be zero. I never really memorized these rules, I have to think through the interactions. Other Coordinate Systems The Unity game engine is left-handed, OpenGL (and most math/physics tools) are right-handed. Why? In a computer game, x goes horizontal, y goes vertical, and z goes "into the screen". This results in a left-handed system. (Try it: using your right hand, you can see x cross y should point out of the screen). Applications of the Cross Product Find the direction perpendicular to two given vectors. Find the signed area spanned by two vectors. Determine if two vectors are orthogonal (checking for a dot product of 0 is likely faster though). "Multiply" two vectors when only perpendicular cross-terms make a contribution (such as finding torque). With the quaternions (4d complex numbers), the cross product performs the work of rotating one vector around another (another article in the works!). Happy math. Other Posts In This Series Given three 3D points p1, p2 and p3 which define a triangle with the intial area A0. I now define a constraint function such that the initial area should be preserved: C(p1, p2, p3) = 0.5 * |(p2 - p1) x (p3 - p1)| - A0 How do I compute the gradient (Or more precisly I need dC/dp1, dC/dp2 and dC/dp3) of this function? How do I generally approach such problems? Are there identies to solve this directly on the "vector" level or do I need to transform the problem into a scalar field? Can I do this with Maple, Matlap or any other tools? Cheers, -Dirk dC/dpx isn't really a well-defined quantity - you can't differentiate with respect to a vector. However, there's nothing stopping you from calculating the directional derivative of C in the direction of (pn - pm)/|pn - pm|.RegardsAdmiral I tried to derive some identities and came up with the following:1) Given a vector function f(r) = r. Building the gradient yields df/dr = E where is E the identity matrix2) Given a function f(r) = |r|. Building the gradient yields df/dr = r/|r| what is actually the unit direction3) Given a function f(a, b) = a x b where a and b are 3D vectors. Building the gradient df/da = -b* and df/db = a* where b* and a* are 3x3 matrices with the property a x v = a* * v (the skew symmetric "cross" matrix)Using this I get for my initial problem:C(p1, p2, p3) = = 0.5 * |(p2 - p1) x (p3 - p1)| - A0 I define e1 = p2 - p1 and e2 = p3 - p1 what gives meC(e1, e2) = = 0.5 * |e1 x e2| - A0 dC/de1 = -0.5 * (e1 x e2) / |e1 x e2| * E2dC/de2 = 0.5 * (e1 x e2) / |e1 x e2| * E1here E_i are the respective cross matrices and i = 1,2Can somebody confirm this or is this total nonsense what I derived here? Any comments are appreciated. How do I resubstitute for e1 and e2 in order to find dC/dp1, dC/dp2 and dC/dp3?Cheers,-Dirk Thanks for the reply. For my understanding in my context I would define this as follows:Given a vector function f(n). I define df/dn as the matrix df1/dn1 df1/dn2 df1/dn3 df2/dn1 df2/dn2 df2/dn3df3/dn1 df3/dn2 df3/dn3 You can use the fact that the triangle area remains infinitesimally the same when you move p1, say, in directions either parallel to p2-p3 or perpendicular to the whole triangle (parallel to (p2-p1)×(p3-p1)), so grad A(p1) is a vector lying in the plane that's perpendicular to e3:=e2-e1 and has length |e3|, so it's (depending on your sign convention, maybe -1*) |e3|*([e3*e3]e1-[e1*e3]e3)/|[e3*e3]e1-[e1*e3]e3|=([e3*e3]e1-[e1*e3]e3)/sqrt([e1*e1][e3*e3]-[e1*e3][e1*e3]).edit: and of course the other two components can be calculated in the same way, and when you add all of them together, you get the grad A(p1,p2,p3) vector. If you really want the behaviour of 0.5 * (p2 - p1) x (p3 - p1), then your second post is the way to go, just don't throw matrices from right to left while you're doing it.[Edited by - Darkstrike on October 10, 2006 12:23:55 PM] I find it easier to use tensor notation to compute the derivative formulas for vector-valued expressions. For example, F(X) = Cross(X-A,B-A) is a vector-valued function of the vector X. Vectors A and B are constant vectors. In terms of tensor notation, the indexed components are F_i = e_{ijk}(X_j - A_j)(B_k - A_k), where e_{ijk} is the permutation tensor (or Levi-Civita tensor): e_{123} = e_{312} = e_{231} = 1, e_{132} = e_{213} = e_{321} = -1, and all other e_* terms are 0. The repeated j index means "sum over all j" and the repeated k index means "sum over all k". The index i is the free variable, so F_i represents a vector. Now differentiate with respect to the X_m component. In tensor notation, you list indices for derivatives after the component indices but use a comma to separate them. The derivative of F_i with respect to X_m is denoted F_{i,m}. The right-hand side for F_i can be differentiated using the product rule to obtainF_{i,m} = e_{ijk}(X_{j,m})(B_k - A_k) = e_{ijk}d_{jm}(B_k - A_k)where d_{jm}is the Kronecker delta for which d_{11} = d_{22} = d_{33} = 1 and all other d_* terms are zero (the delta represents the identity matrix). It has the property that T_j d_{jm} = T_m (you get to replace the j by m in the tensor it multiplies). In the previous formula,F_{i,m} = e_{imk}(B_k - A_k)The k is repeated, so you sum over all k. The indices i and m are free variables, so F_{i,m} is a second-order tensor (a matrix as it were). DespiteTheAdmiral's claim, you can think of this as a derivative with respect to a vector: dF/dX = M, where M is the second-order tensor e_{imk}(B_k - A_k).In your problem, define the scalar-valued function G(X) = 4*[(C(A,X,B)+A0)]^2 where A represents the constant vector P1, B represents the constant vector P3, and X represents the variable vector P2. ThenG(X) = Dot(Cross(X-A,B-A),Cross(X-A,B-A)) = F_i F_iThe derivative with respect to X_j is G_{,j} = 2 F_i F_{i,j}. In indexless notation, dG/dX = 2*Transpose(F)*M, where dF/dX = M (see above). Also,dG/dX = 8*(C(A,X,B)+A0)*dC/dXand you can solve for dC/dX. Hey Dirk.The following should help you.Given two vectors p and v:d(|p x v|2)------------ = -2p x (p x v) dvThis can be derived by using the following:d(vTAv)--------- = Av + ATv dvwhere, A is a matrix. Quote:Original post by DonDickieDThanks for the reply. For my understanding in my context I would define this as follows:Given a vector function f(n). I define df/dn as the matrix df1/dn1 df1/dn2 df1/dn3 df2/dn1 df2/dn2 df2/dn3df3/dn1 df3/dn2 df3/dn3yeah, thats all there is to it basically. this is usually written as grad(f), and is indeed a (symmetric) tensor. Quote:Original post by Eelcoyeah, thats all there is to it basically. this is usually written as grad(f), and is indeed a (symmetric) tensor.The matrix of derivatives is not usually written as grad(f). This matrix is a tensor but it is not necessarily symmetric. Thanks to everybody for their help. Actually I came up with another solution, which I find very practical. Basically I use the fact that dC/dt = dC/dp * dp/dt = grad(C) * v (chain rule).I will also use following identities;C(n) = |n| -> dC/dn = n / |n| (Gradient)C(a, b) = a x b -> dC/dt = da/dt x b + a x db/dt (Time Derivative)Given:C(p1, p2, p3) = 0.5 * |(p2 - p1) x (p3 - p1)| - A0 dC/dt = 0.5 * (p2 - p1) x (p3 - p1) / |(p2 - p1) x (p3 - p1)| * [d/dt(p2 - p1) x (p3 - p1) + (p2 - p1) x d/dt(p3 - p1)]Define n = (p2 - p1) x (p3 - p1) / |(p2 - p1) x (p3 - p1)| dC/dt = 0.5 * n * [(v2 - v1) x (p3 - p1) + (p2 - p1) x (v3 - v1)]Now use cross product identies and solve for the form:dC/dt = dC/p1 * v1 + dC/dp2 * v2 + dC/dp3 * v3 = grad(C) * vThis is what Erin Catto calls "the Jacobian is found by inspection". BTW: What would be the advantage in a simulation to the normalized form of the area preserving constraint:C(p1, p2, p3) = [0.5 * |(p2 - p1) x (p3 - p1)| - A0] / A0Cheers,-Dirk