[SQUEAKING]

[RUSTLING]

[CLICKING]

BERTHOLD HORN: We're talking about photogrammetry and, in particular, absolute orientation. And we're drilling down into more details for this one than we will for the other aspects of photogrammetry because a lot of it's going to be based on what we're doing here. And we found that rotation is the part that's awkward to handle.

So we talked about different ways of representing rotations. And we picked unit quaternions. And major reason for that is that, given all of the advantages we discussed before, the big thing for us is we can get a closed form solution to the least worst problems. So in some sense, we have an objective way of getting a, quote, "best fit answer," which is more difficult with the other notations.

Then we talked about how to manipulate these things. There are two operations we're particularly interested in. The one is composition of rotations. If you do successive rotations, what happens? And the other one is rotating a vector. So composition of rotations is just multiplication. And we can represent it in various ways. For those of us who grew up with vectors, maybe mapping it into scalars and vectors is the most intuitive. So we have this formula. And it's actually quite efficient in terms of composition of rotations. In terms of computation, it's a lot less work than multiplying two orthonormal 3 by 3 matrices.

But for much of what we do, what's more important is the other operation, which is rotation of a vector. And so for that, we have that formula where we're rotating in four space using q. And then we're unrotating using q conjugate. And somehow, we end up back in a real three-dimensional world. And for that, again, if we want to just think about vectors, one way we can write it is-- so if we don't want to think too much about operations in the quaternion world, we can just do this operation directly using vectors and scalars.

And this one, we have a disadvantage. It takes somewhat more arithmetic operations than multiplying a 3 by 3 matrix by a vector. Although we saw that the ways of rewriting this were we reused the q cross r that reduces the number of operations somewhat. And then we need to connect this to other notations. So we, for example, want to relate it to axis and angle notation. And there, we have the formula of Rodriguez, where omega hat is the unit vector in the direction of the axis. And so that gives us one conversion that we want.

So we can use that to-- q dot-- sorry about the handwriting. And when we identify these two formulas, we find that actually q is cos omega over 2 and omega hat is-- well, let's see. Other way around. q vector is omega hat sine theta over 2. Oh, let me make that theta. Sorry. Our criterion for rotation looks like that. And so we can right away read off the axes, it's going to be parallel to the vector part. And if we want to, we can read off the angle by using a10 on the ratio of the real part and the magnitude of the imaginary part.

So that is one way of converting between two of these different forms. Now, we have eight. So we don't want to go through all eight of them. But the other one that's important for us, since we live in a world where orthonormal matrices rule, we need to be able to convert back and forth between those. And so if we expand that formula using that isomorphism with 4 by 4 orthogonal matrices, then we got that. And we expanded that out.

And when we look at that matrix, we find that it has skew symmetric parts and symmetric parts. And we can use that to help us in the conversion from one form to the other. So first of all, if we're given the quaternion, this allows us to compute the orthonormal matrix very easily. So in this transformation, it's 4 by 4. But we don't care about the first row and the first column because that is a special quaternion that represents a vector where the scalar part is 0. So all the first column and the first row tells us is that if we have a completely imaginary quaternion, then when we perform this transformation, we get back one of those. So we're back in the world of vectors.

So that's the one way of conversion. The other way is a little bit less obvious, partly because we're going to have a 3 by 3 matrix. So we have nine numbers. And our answer only has three degrees of freedom. We only want the axes and the angle. And so you've done some of this. But one part we can do is to look at the trace of that matrix, that 3 by 3 sub matrix, and I think we end up with something like 3q squared minus-- so it's just running down the diagonal.

And of course, q0 we said over there was cosine of half the angle. And then this part, each of these is proportional to sine of half the angle and is a unit vector. So if we take square of these, we should get sine squared theta over 2. So trace equal to that. And then we can manipulate this in various ways. We could add cos squared theta over 2 plus sine squared theta over 2 equals 1 in order to get rid of that sine squared theta. So we can add that in without changing anything. Or we can subtract it.

So if we subtract it, we get that thing that's more symmetrical than cosine and sine. And of course, that's the double angle formula. So that's two cosine of theta. Oh, we need the minus 1. And from that, we can solve for cosine of theta is 1 half trace of R minus 1. So right away, that allows you quick test on whether rotation matrix could even be a rotation matrix. Because if you trace ends up being too large or too small, cosine theta can only be between plus and minus 1. So that limits the value of the trace. So that's a lot cheaper than checking whether it's also normal.

Now, that's a way to get the angle. It's not a good way because the problems in the theta equals 0 because we have that thing. We're up on the curve and a tiny change in the height potentially corresponds to a large change in the theta. And similarly-- so yes, this is true, but that's not the way to compute theta. So what do we do?

Well, we use the off-diagonal elements, right? Because the off-diagonal elements all depend on the sine of theta over 2. And then it depends on whether we're going to use the symmetric part, like q,x, q,y, which would then depend on sine squared of theta over 2 or the asymmetric part, which is q,0, q,z, which is going to depend on sine theta over 2, cosine theta over 2, which is the double angle formula for sine theta.

So anyway, need to get sine theta so that you can then use atan2 and avoid that problem because where this one is bad, sine theta is good. OK, so that's one way to go. But to make it really explicit, let's actually give a full inversion formula.

And I do this mostly to vaccinate you against conversion formulas that have been published because they are not good. So in this kind of sense, where, yeah, mathematically you're done. You've got this. But actually, in terms of numerical accuracy, no.

OK, so given the three-by-three matrix, r, we can compute various sums. And let's start with a diagonal. So we've got our three-by-three matrix over there. And if we end up with the diagonal, we get that if we add 1, right? There's that 1.

Then we can combine it in various other ways. So this is just sort of like, you know, try all possible ways of subtracting and adding terms on the diagonal, as if the off-diagonals didn't exist. And ta-da. Now, we just take square roots. So that's one approach.

Of course, the problem is there's a sine ambiguity because, in each case, we can compute these sums and differences and then divide by 4 and take the square root, and we've got one component of the Quaternion. But we don't know whether it's plus or minus.

And yeah, we know that plus q is the same as minus q. But this is more. This allows us 16 different sine choices. So even if you allow for the flipping of the sine of the Quaternion. That leaves us with 8. So that means that we shouldn't rely on this method alone.

But what we can do is compute these, and then pick the largest for numerical accuracy, and solve for it. So suppose this sum is the largest, then we'll use it and arbitrarily use the positive version because we have that minus q is the same as q, so we can pick. We can make one arbitrary choice and solve for that. So let's call that q,i. And then, we need to go to the off-diagonals.

And fortunately, they're symmetric and asymmetric parts that we can pull out. Now, this is a bit more involved than the published methods, but it works. It's resistant. It's robust. It's resistant to numerical problems. So by adding and subtracting corresponding off-diagonal parts, we get six relationships more than we need.

But the way this works is, we've picked one of these and solved from the diagonal. And then we go into three of these and solve for the rest. So for example, if we solved for q,y, this involves q,y, this involves q,y, and this involves q,y. So we picked those three to solve four q,0, q,x, and q,z. So then correspondingly, if we have-- so that's a direct way of going from Quaternion to rotation matrix.

I mean, we can also go indirectly. We can first find axis and angle and then write the rotation matrix in terms of axis and angle. And it's more complicated mostly because we've got nine numbers, and they could be noisy, and we'd like to get the best possible result out of those nine numbers.

OK, so out of the 56 ways of converting between different representations, we've done four. And that should be enough. So we can convert back and forth between the things that we're more familiar with-- maybe not between poorly spin matrices, but we don't use those much, except in quantum mechanics. OK.

And you did some of this in the homework, and I don't want to repeat that. But let me talk about something in particular, scaling. So far, we'd assume that our coordinated system transformations were rotation and translation. And in many cases, that's all there is. If we have actual time-of-flight data to measuring distances, there's usually no question about the scale factor because we know the speed of light with more precision than you can throw a stick at, although there's often an error in offset.

And so how do we get into scale? Well, it's quite easy to get into scale-- for example, if we get the baseline wrong So our plane is flying along taking one picture, and taking another picture. And we know the speed of the plane, and we know the time, so we know how far-- we know the baseline. But how accurately do we really know it?

And so if we are trying to patch together pieces of the terrain that we obtained with successive-- two camera positions, then we should allow for some small difference in the scaling because we know the distance pretty accurately, but not perfectly. And we're looking for very high accuracy in topographic reconstruction.

So how to deal with that? Well, let's assume we've-- this isn't hard. We again find that centroid maps the centroid, even with the introduction of scaling. That's very easy. Just differentiate with respect to translation. Set the result equal to 0.

And then we move the origin to the centroid, and we get these prime coordinates. So now, let's look at this. So here's a possible version of the problem we're trying to solve that, now, we've got rid of translation by moving to centroid. And now, we have an unknown rotation, and we have an unknown scaling. That's a new part.

And so we might want to set up a least squares problem. And of course, the norm is the dot product of this thing with itself. So we can actually multiply it out. We get four terms. And then the sum of those four terms is the sum of four sums. And so we end up with-- so this is very similar to what we've done before.

And how do we find the optimum? Well, of course, we just differentiate with respect to s. This is particularly easy now. It's just a scalar. And we set the result equal to 0. And the first term drops out, and then we get the second term. Oh, here, of course, we use the fact that this the size of this vector is the same as the size of the rotated vector. So rotation doesn't change the size.

OK, this is equal to 0. And so we can solve for s. Well, let's give these things names. Let's call this Sr, so we don't have to write them too many times, and let's call this D, and call this Sl. So we have S equals D over Sl. We just solved this for D. We can forget the 2.

And so we don't, at this point, know the answer because we haven't got the rotation yet. But in the same situation as we were with translation, we can remove the scale factor from consideration now. And then at the end, we go back and use this formula to compute the scale factor. So that keeps everything coupled that we can't do it independently of rotation.

Now, we talked earlier about symmetry. Why are we rotating from left coordinate to right coordinate rather than from right coordinate to left coordinate? And a method should really be symmetrical, in that if you compute the inverse, you should get the inverse. So the rotation matrix you get should be the transpose of the rotation matrix and so on.

So here, if we now go from the other way round, if we look at this problem, then we expect to get a scale factor that's the inverse of that scale factor, right? Because we're going from right coordinated system to left. We were going from left to right before.

And if you actually work it out, you get that. So again, you can do it, but it's not the same answer. It's not the inverse of that. What we would expect is that this prime is 1 over S, and it's not, in general. So that suggests that this is not a good method. And what's going on?

Well, what's going on is that the least squares method is trying to make the errors as small as possible. And one way to do that is to make the transform coordinates smaller up to a point. So you can kind of cheat by making the scale factor a little bit smaller than it really should be because you're shrinking things down and making things smaller. And then conversely, if you go in the other direction, you shrink down the other coordinate system.

So neither of these is really acceptable. And so we look at another error term, the one that you looked at in the homework problem in the 2D case. And so in this case, we're going to end up with an error that looks like this, where the Sr, Sl, and D as defined over there.

And that's nice because S doesn't show up here in that term that has the rotation in it. We just have these other terms. And so we can just differentiate with respect to S, set the result equal to 0 because derivative of that 0 with respect to S is 0. And so that means that S squared is Sr over Sl.

And now, if I go in the other direction, I'm going to get S bar squared is Sl over Sr. And that is the inverse. So that method is much to be preferred. It also has the property that we don't need correspondences. So it's just like translation, where we are able to map centroid to centroid. We didn't need correspondences for that.

Here, it's very convenient that we don't need the-- the only term that depends on correspondence is that one. We only need these others, which basically say, you've got this cloud of points. How big is it? And then you have the other cloud of points. How big is it? Well, the scale factor is the ratio of those two sizes. It's very intuitive.

OK, so that leaves us with the problem of-- so we can deal with translation and scaling in the correspondence-free way and also free of rotation. We don't need to take into account rotation. Well, we can separate the problems.

So that leaves us with the rotation part. And we spent some time on that, so I won't review that in great detail. Just note that, in the end, we have this to maximize because this was a minimization, but we have the negative sign, so this has to be maximized. And N, of course, is a four-by-four matrix, which we can show is symmetric. It's not obvious that it is.

And so now, we have a calculus problem. We just differentiate with respect to q and set the result equal to 0-- well, not quite. We have that constraint. And that's just as well because, otherwise, the answer is 0 because that differentiate that, and you get 2N,q. How do you do that? Well, I'm sure you remember these formulas and so on-- so all in the appendix.

OK, so if we set that equal to 0, well, there's a very convenient answer, which is that q is 0. Or looked at another way-- that's an extreme one. Well, we're actually trying to maximize, so we'd go the other way. We would just make q grow. We can make q as large as we like. And so there's no maximum-- well, infinity.

So that's obviously not satisfactory. We need to take into account the constraint. And as we saw in the slides last time, one way to do that is using Lagrange multipliers-- a very elegant method, and it's described in the appendix. But this is a special case where we can get away with something simpler called Raleigh quotient.

So Raleigh did all sorts of interesting things, including work in optics. And he ran into this problem of trying to find an extremum. And so he came up with the idea, well, how do I prevent this from running away by being made very large? Sorry, this is transposed there, yeah.

Well, I just divide by the size of q. So there's no advantage to making q very large, right? So if you think about it as directions in a space, this quantity is constant along any ray because wherever I am, I'm going to divide out by the length of q itself. So that creates a different function, the one that doesn't go off to infinity. But that is constant along any ray, and that's exactly what we want. We want to find the ray, the direction of the ray that makes that as extreme as possible. Yeah?

AUDIENCE: What is the matrix, then?

BERTHOLD HORN: Oh, so that's the thing that we got out of where-- it disappeared-- where we took the representation of a Quaternion product with as a four-by-four rotational, orthogonal matrix. So that's sort of the key--

AUDIENCE: Can you call it q, really?

BERTHOLD HORN: No. Where is there space? Well, let's just take this. Just a quick review of that part-- so I'm not going to copy it all out, just at the beginning. So we had this sigma over-- right? This was the thing we were trying to maximize. And then we wrote-- first, we took this over to the other side to make it more symmetrical, so-- oh, sorry.

And then we used that notation where we represented this as-- now, before, we expanded it the other way out. We converted q into a four-by-four matrix and multiplied by this Quaternion. And now, we do it the other way around. And then because this is a dot product, it's that transpose.

And then we get 2 transpose R transpose li, R,ri. And of course, this is the sum of that. And q doesn't depend on i, so we can get rid of that. So it's that matrix. So it's derived from the data, from the correspondences.

OK, so how do we find that? Well, now, it's a pretty straightforward calculus problem. We just differentiate with respect to q and wet the result equal to 0. And it's a ratio, so we use the rule for differentiation of a ratio. And so first, differentiate the numerator. We get 2N,q divided by this. And then we need to add the term involving the denominator. So we get minus 1 over q dot T q,0 squared 2,q, and q,0 N,q.

So this is just the formula for taking the derivative of a ratio, and this 2,q here is the derivative of q transpose q with respect to q. And this is supposed to be 0. So that means that N,q is-- and this, of course, is a scalar, some constant. And so what does this tell us about the q? That we multiply some matrix by q and we get out some scaled version of q-- so eigenvector, right?

So q is an eigenvector. And now, we're trying to maximize this. So we need to know which of the eigenvectors to pick. And it's pretty easy to see that if N is lambda q, then this ratio over here is going to be lambda q transpose q over q transpose q is lambda.

And so to maximize it, we pick the eigenvalue. Pick the largest eigenvalue. We want to maximize this ratio. And the way to do it is to pick q to be the eigenvector corresponding to the largest eigenvalue. And then this whole thing is equal to that eigenvalue. So obviously, you wouldn't want to pick a smaller eigenvalue than the maximum one.

OK, and so the only slight tipping point is the fact that there's this constraint. But the constraint is so much easier to handle then the constraints we had with the normal matrices. So we can actually do this in a very straightforward way-- Raleigh quotient.

So a couple of things that we usually ask after we've, quote, found the solution, one is, when does it fail? And the other one is, how many points do we need? How many correspondences? And so let's start with that. So in all of these photogrammetric problems, that's an important point.

And so well, we can approach this from the point of view of the properties of this matrix N. But it's a bit of a bear of a problem to worry about. And singular is how many eigenvalues are non-zero and so on? Let's just do it intuitively.

So first of all, we can say, well, how many things are we looking for? We're looking for six. If we're looking for translation or rotation, we're looking for six. If we add scaling, we might be looking for seven. So each measurement correspondence, there's a point in 3D and a point in 3D that we say are the same. So that's worth three correspondent-- three constraints, right?

So we're looking for six things. So hey, we only need two correspondences, right? That's assuming there's no redundancy in that information. Well, let's start off with one correspondence. So here's an object. And then we have a second object. And we want to know how one is rotated relative to the other.

And of course, we know the duality between two objects, one coordinate system, or one object, two coordinate systems. So let's do the two objects, one coordinate system. And obviously, this isn't going to do it because I can move this thing. I can rotate it around that point without changing that correspondence. And also, I can rotate it about this axis. It's just very little constraint-- well, three constraints out of six dimensional possibility. So that's not good enough.

So let's pick two. Suppose that our very rough back-of-the-envelope calculation suggests that two should be enough because each of those correspondents gives us three constraints. It's very powerful to say that this point is on top of that point. And so we get three out of one correspondence, three out of a second correspondence. That makes six. We're looking for six degrees of freedom. Maybe that'll do.

Well, what do you think? I mean, if I tie two objects together at two points, is that enough to rigidly combine them? Or is there some degree of freedom left? So it's hard to see because it's in 2D. Well, draw that axis, and take one of the objects in the rotate it about that axis. So no, that doesn't work.

And the reason is that when we make the second measurement, we don't get three new things. The information's redundant. Why? Well, because there's the distance between the two that is fixed. And so it is true we have six numbers, but there's also one that's not worth anything new. And so we only have five constraints. So yes, we've got just one left, and perfectly makes sense. We only have one degree of freedom left. We can rotate about this axis.

So OK, so it takes three. It takes three correspondences to work. And that method that we deprecated at the beginning used three correspondences. So in that respect, we're not any better. We need exactly the same number of correspondences as that method that we pooh-poohed and said wasn't very good.

OK, so one question is-- we've come across this before in the 2D world-- is there some way of generalizing the transformation so that it matches? Here, with three points, we get nine constraints. And of course, they're highly redundant because the distances between the points are the same. And so with scaling, we're up to seven. So that's already going in the right direction if we add scaling. We need to add two more so that we would fully use and need three correspondences.

So one idea is, well, how about the general linear transformation? And we did this in 2D. And I guess, in 2D, we ended up with six degrees of freedom. What is it in 3D? So well, so we could have no longer constrained to orthonormal matrices. And that actually is a great thing because the orthonormal matrices are the ones that produce these horrendous problems with the constraints that they are orthonormal. And oh, this looks just right. It's got nine numbers in it. So nine constraints from three correspondences matches to that, so it looks like this is, perhaps, the generalization that we need.

Well, unfortunately, not quite because we have translation as well. So the general linear transformation has 12 elements in 3D. We had six in 2D. And so there isn't a nice symmetric argument that, with three correspondences, it would work. We would need four correspondences. And that does work. And it's very elegant and very neat. The least squares comes out beautifully because there are none of these annoying constraints-- like, determinant of the matrix is one.

But that's not the transformation that we're dealing with in the real world where we have two sensor systems looking at the world, and they're typically only dealing with rotation and translation, maybe scaling, but not that full-- this allows skewing of the axes and asymmetric scaling, that the x-axis is scaled different from the y-axis, and that there's a slant, and so on.

So OK, so back to our analysis of this method, we've said something about how many correspondences we need. So one way to fail is to not have enough correspondences. And if you have only two correspondences, that matrix N will be singular, which isn't enough to make the method not work.

But it will it will have more than one eigenvalue that's zero, and that's a way to formalize our hand-waving argument. But it's kind of messy, so I didn't want to do that.

But there is another case that we want to look at that's actually kind of interesting. So how do we find the eigenvalues? Well, we use MATLAB. Or, more seriously, we need to solve this characteristic equation, which is obtained from that matrix N. So that's the equation that says, determinant of N minus lambda i is 0.

So that's the determinant of a four-by-four matrix that has minus lambdas down the diagonal, subtracted out the diagonal. And when we take the determinant, we get up to fourth-order combinations of lambda. So it's going to look like this.

Now, our matrix N is a particular matrix that was obtained this way. And so actually, we can say more about this. I already mentioned this last time. It's not easy to show, but C,3 is the trace of the matrix that happens to be 0.

Well, it's easy to show it's a trace of a matrix. It's not easy to show that it's 0. So anyway, it's 0, which makes this equation easier to solve. In fact, usually, the first step in solving a fourth-order equation is to eliminate the third-order term. Well, it's already happened. So we're ahead of the game.

What about the others? So this is the trace of another matrix. So these others aren't 0, so what is this matrix M? Well, the matrix M is where we actually started with. This is what you'd compute when you got the correspondences.

What's that? That's a dyadic product. So we take the victor in the left-hand coordinate system and multiply it by the victor in the right-hand coordinate system. But rather than the transpose of the first times the second, which gives us a dot product, we do this. And you've seen that. So that's a three-by-three matrix.

We just stepped through the data, correspondence by correspondence. And we compute this dyadic product, and add it into a total, and we get that matrix N, which is, of course, three by three. And it turns out that the four-by-four matrix N has terms that can be computed from this matrix.

So that's actually the most efficient way of getting N. You just sift through all of your data and compute M. Then at the end, you compute N from it.

OK, next thing-- suppose that the determinant of M is 0. Suppose that M is singular. Well, that's pretty useful because that means that C,1 0. And then we can all solve this equation without needing to go to any special textbook or something because then, we've got lambda to the fourth plus C,2 lambda squared plus C,0 is 0. In that case, it reduces to that problem. And we use what equation to solve that?

So it seems to have lambda to the fourth in it, which means we should use Ferrari or Cardano. But of course, it's quadratic in lambda squared, right? So we just apply the quadratic. So it's a particularly easy case. And so you might wonder, well, who cares? That's a special case.

Well, it turns out, it's a it's a particular case that's interesting from a geometric point of view. It has to do with the distribution of the points. And so this matrix N can have some 0 eigenvalues, which are potentially problematic. And when does that occur? Well, one is when points are coplanar.

So far, we've talked about clouds of points, like the spikes on a sea urchin. And we're matching two of these. So we're rotating one in alignment with the other. But this applies equally well if all of those points are in a plane. And so what happens then?

Well, if there's a plane, then all of our points in that plane, and we can draw a unit normal to the plane, and then the dot product has to be 0. So let's suppose all the right-hand points are in a plane. Now, with our measurement error that implies that the left-hand points also are in a plane, but we may not need to make that restriction. There might be measurement error, and the others might not be exactly in the plane.

So what I've done here is I've drawn the centroid. So I've already moved everything to the centroid. And therefore, I can write the equation of the plane in this simple form. The dot product-- the component of any one of these vectors in this direction is 0. That's the definition of a plane.

OK, well, then, that's the same as r,i transpose n. And then I can go to M,n hat is-- OK, so if the points are coplanar, then we do have the determinant of M is 0. And then that problem is particularly easy to solve.

Now, I haven't shown it the other way round, which I should, which is that if determinant of M is 0, the points are coplanar. But I think you can see how to do that without too much trouble.

Now, in that case, actually, we can solve the problem more simply. Suppose that things are nice, and both are coplanar, both the left-hand cloud of points and the right-hand cloud of points. So I can draw these two planes. So here's one plane. So all of the points in the left-hand coordinate system are one plane. All of the right-hand ones are in this other plane. Not particularly good drawing.

Well, two planes-- so all of left point cloud, right point cloud. So then, I can decompose the problem into two steps. The one step is rotate the one plane so it flies on top of the other plane, and then an in-plane rotation, and I'm done. So the only two things-- I can decompose the full 3D rotation into two very simple rotations.

Now, to do that, I'd need to find this angle. And I need to find this axis. And the axis is parallel to n,1 cross n,2. If we have two planes that have normals n,1 and n,2, they intersect in a line, which is parallel to n,1 cross n,2. And the angle, of course, I can get very easily also.

I take the dot product. That gives me the cosine. I take the cross product, and the magnitude of that gives me the sine. And then I use atan2. OK, so I've got the access. I've got the angle, so I can construct the Quaternion. And then I rotate all of the points in one of the coordinate systems.

And now, the two planes are on top of each other. And now, I've just got to take the rotation in that plane so that these points align. So that's a least squares 2D problem, like you've solved in homework problems. Of course, it's in a plane that's somewhere in 3D-- but pretty straightforward anyway.

So that's interesting that the special cases where things changed in terms of the difficulty of solution also correspond to some things that are physically relevant. So OK, so let's see. Where to go next? I don't want to go onto relative orientation too quickly. I want this to sink in because this is like a triumph. We've got a closed-form solution. And sadly, so far, there isn't one for relative orientation. But we use a lot of the same tools-- the representation for rotation and so on.

But I want to look at some other aspects. One of them is least squares. Everything we've done in this course is least squares. And if you talk to your more sophisticated friends, they'll say, oh, that's useless because it's not robust. And what does that mean?

Well, it means that if you have nice noise, least squares is the way to go. By nice noise, we mean something that has a reasonable distribution, like Gaussian. And the problem we face in practice sometimes is that a lot of the measurements follow a Gaussian distribution. But every now and then, you get something that's completely wacko. And so what do you do with that? So how do we deal with outliers?

Now, you can formulate a new optimization problem that uses something other than square of error-- for example, absolute value of error. But it no longer has a closed-form solution. And so there's a trade off. Obviously, we don't have a closed-form solution. It might be more work to compute. That's often not an issue.

But it also makes it very hard to say anything in general about it. Like, how does it vary with the parameters, and so on. If the only way you get the answer is to crunch the numbers, then you don't have that pleasant experience of a general formula that says, oh, it's going to vary linearly with frequency or something like that. So that's one disadvantage of departing from this least squares method.

So suppose that we have a 2D problem, and we're just doing a line fit. So there's our least squares line. But now, suppose that there's some data point that's just completely wacko. Well, that has a huge square of error. And so it's going to pull your solution dramatically in that direction.

So that's the intrusion. And it'd be slightly better if it was absolute value of error because, then, it wouldn't pull quite as hard. But it would be even better if you said, oh, that point's just crazy. Forget it. And there are different approaches to this. So there's a question of robustly dealing with outliers.

Now, it actually goes further than that, that you can show that there are certain problems where, if the noise is Gaussian, then least squares gives you the best possible answer. It's not just hand waving, but maximum likelihood, all those wonderful things. So how do we deal with this problem?

Well, there are various approaches. The one I'm going to talk about is called RANSAC, Random Sample-- I don't know-- Agreement. And Fischler and Bolles invented it at SRI, Stanford Research Institute, which is connected to Stanford in some way, although because they do classified work, it's not as intimately connected as a C cell. It's more like Lincoln Labs.

Anyway, they had problems like satellite navigation and making measurements of, I don't know, star positions, gyroscopes. And occasionally, they'd have some massive error, which would completely mess up their result, even though, most of the time, the errors were well behaved. They followed some nice, smooth distribution with not too much spread.

So what's the idea? Step one, random sample. So if I take a random sample of these points and there are not too many outliers, there's a good chance that I won't hit the outlier, and I will get a good result. And how do I know that I picked the random sample and how large? So there's several questions. How large a random sample?

Well, their recommendation was the minimum needed to fix the transformation. So in our case of absolute orientation, there would be three. Now, we know if we get 1,000 correspondences, we're going to get much bette results than with three. It goes as 1 over the square root of n. So that's limiting the accuracy, so some other people recommend that you use more.

And then what do you do? So now, you take the random sample, and you do a best fit, typically least squares because it's efficient and easy to understand. So OK, and then check how much of the data.

And so what you do is if that's your fit, then you add some sort of band. And you count how many of them fall inside. And notice that this involves some kind of threshold.

So in our case here, let's say that all but one point fall in that band, and we say we're done. If not, we take another random sample, and we do this operation again. And there are all sorts of interesting issues.

One of them is you have to know what the ratio of inliers to outliers is. So you have to have some model of your measurement process because that's going to control how you set the threshold. So you need to know something about the noise to set that threshold.

And then you need to know how many. So there are two thresholds. There's a threshold here of the band that determines whether these are inliers or not. And then there's the second threshold here that says-- oops, scared her with some equations.

So we got arbitrary numbers in there, which assume that you know something about the data. And this works pretty well. I mean, I guess, you have to have some limit on not doing this an infinite number of times. So you probably have another exit clause which says, well, if you've tried 100 random samples, then let's stop, then, probably, your assumptions about the model are wrong, and the thresholds aren't right. So you need to give up.

And there are variations of this. So that's a good point. I didn't make that point yet. But we're talking specifically about absolute orientation.

But this trick is to be applied, if necessary, if there are outliers, to any of the other least squares method. So the least squares approach is very appealing because you get a closed-form solution many times, and you get a measure of how large the error is-- standard deviation-- and all these good things. But it's always subject to the problem of bad data. And so this is definitely a method to consider in the presence of real data, which sometimes has outliers. Yes, OK.

And there are different variations of this. In fact, the name of it, consensus, kind of suggests a different approach, more like a Hough transform. So before I read the paper, what I thought they were doing was the following. You take a random subset. You do a fit, and that gives you some answer, like slope and intercept of a straight line, say, in this case.

And you do it again. Do it a certain number of times. And then you plot them in some parameter space. And some of them will cluster tightly, and some of them that included the outlier will be way off.

And so you can imagine a Hough transform method where you have an accumulator array in this parameter space. And whenever you get an answer, you increment that cell in the parameter space. And after you've done it often enough, you look for the cell that has the most hits. And that's somehow hybridizing Fischler and Bolles' RANSAC method with a Hough transform method. And all of these are kind of ad hoc, but they are very useful in practice.

OK, well, one of the things we're going to do a little later is to find representations for objects, particularly objects that are not simple polyhedra, but more complicated, and, particularly, objects that are near convex that might not have a lot of holes. And that representation is very useful for detecting things and finding their orientation. And we run into the same orientation problems as we have over here.

And often, there's not a closed-form solution. So one approach is to sample the space of rotations. And how do you do this?

So since we're talking about-- so let me first look at sampling a sphere in 3D. So one way we can do it is to use some discretization of latitude and longitude. You know, I just find there's some people who-- I forget what the terminology is. But where these lines cross, that's like a point of convergence. And if you go there, something magical happens to you, particularly if you take along crystals. And sometimes, these are hard-to-access places, like in the middle of a military reservation.

Anyway, we sample them. And what's wrong with that? It gives us a way of sampling the whole space. But obviously, it's not a uniform sampling. Up here near the poles, we're sampling points that are very close together. And so that means it's inefficient, that if we had used some of those points near the equator, we could have done a better job, more even job of sampling.

So there are lots of interesting questions that come out of this. One of them is, well, can we think of a better coordinate system so that, if you sample uniformly in those two coordinates, you will get a nice spacing on the sphere? Well, good luck with that. You can show that it's not possible.

So what else can we do? Well, we could generate random theta and phi using some distribution. So let's see, latitude goes from minus pi over 2 to pi over 2 and longitude minus pi too. So we could divide this up.

We could have a random number generator that goes between minus pi and plus pi and one that goes between minus pi over 2 and plus pi over 2 uniform. And of course, we'd have exactly the same problem because we're going to sample the polar region more strongly than the other region. So that's not very good. So how to solve that?

Well, the trouble is, we got this curved surface. If we had a Cartesian world, it would be easy to sample. If I have a cube, I can just have a uniform distribution in x, a uniform distribution in y, uniform distribution in z. And I call a random number generator three times. I have a point in that space. And if I do this a lot, I don't expect there to be aggregation, systematic clumping, as we have in the case of the sphere.

So how to go from this to that? Well, one thing to do is to inscribe a sphere in the cube and then somehow map the cube onto the sphere. So one way to do that would be to imagine the origins at the very center of this thing. And any point on the sphere, we draw a line, and then we find where it intersects the sphere. At any point in the cube, we sample-- and that point could be inside the sphere. So if it's inside the sphere, we draw that line, and we project it out to the sphere in this case.

So to be clear, again, we just have uniform, like most languages give you the ability to generate uniform random numbers between 0 and 1. You just map them onto that interval in x, and then another one interval in y, another one interval in z. There is a point in 3D. And you have to reduce it to the surface of a sphere while you just normalize. So you have a three vector. You just take the unit vector version of that, and you're done.

So does that give me a uniform distribution on the surface of the sphere? No, OK, so we have some head shaking. And that's because, well, the cube has these corners. And so there will be a higher density of points in the directions of the corners then there will be where the sphere is tangent to the cube. It's better, though. This one's only off by-- I forget-- a factor of 4 over pi, I think, or pi over 3-- no-- 4 over pi. So this isn't too bad.

So how can you do better? How can you take this idea and fix it so that you don't have the corner problem?

AUDIENCE: Tessellate the shapes.

BERTHOLD HORN: Tessellate the shapes-- yeah, we could tessellate the surface of the sphere using regular solids, the magic things that Plato organized his society about. That's certainly a good approach. Is there some way of discarding some of these points that we are generating here?

OK, so counteracting this aggregation in the other corners, we could introduce some weight. And for example, if we could give the points when projected a weight, we could just make it inversely proportional to the radius. So if we're far out, then-- so that's a good idea.

Another idea is to say, well, we could just throw away all the points that are outside the sphere. And then, we have a random distribution of points in a sphere. So we, again, we generate points in the cube.

But then, we check whether their radius is bigger than 1. And if it's bigger than 1, we chuck it out. And it has a disadvantage that you're not generating points at a fixed rate, that it costs you more, sometimes, because you have to go back and regenerate it, and maybe regenerate it again. So there's a disadvantage.

But what happens then? Well, then we have a uniform sampling of the inside of the sphere. And that's almost good enough. Now, we just need to project all of those points onto the surface of the sphere. And if we want to avoid a nasty numerical problem, we might want to also throw away all points that are close to the origin because, then, we might have numbers you're dividing by normalizing the vector to make it a unit vector.

You might be dividing by some small number, which is going to introduce some error and biases. So you generate points uniform in the cube. You throw away all points above some certain radius. And you throw away all points that are below a small radius-- I've drawn it a little big there-- and then normalize them. And so there's your point sampling of the surface of the sphere.

So we may not care about the sphere. But I can't draw this in 4D. But it works exactly the same way, right? If we want to have a nice, uniform sampling of 4D space, one way-- a random sampling-- one way of doing it is exactly this. We do a four-dimensional equivalent of the cube. So we generate four random numbers for x, y, z, and, say, w. And then, we check whether our sum of squares is larger than 1. If so, try again.

If not, then we turn them into a unit Quaternion. And that gives us a very simple way of getting a uniform sampling of that space. Now, there was mention of regular polyhedra. So that's an important thing to think about. So what are they? Well, they're tetrahedra, they're hexahedra, octahedra, dodecahedra, and icosohedra.

So these are the so-called regular solids, regular polyhedra. That means that all of their faces are the same. And those aren't the only objects of interest, but let's start with those. So if we want to sample the surface of the sphere, one other way is to project out-- well, these obviously aren't spheres.

But you can imagine them placed at the origin and then projected out onto the sphere. And we get great circles wherever there's an edge on one of these objects. And so that's a way of dividing up the sphere in a perfectly regular way.

And what's the problem? Well, the only thing is that there aren't too many facets. So tetrahedra, 4, 6, 8, 12, 20. So we're not dividing the sphere up very finely. And as Plato discovered, that's it. There aren't anymore. So that means we have to subdivide.

And you might say, well, wait. What happened to the soccer ball? How many faces on a soccer ball? Ugh, how can people play games and not even know how many-- anyway, 32. So it's a mix of the dodecahedron and icosahedron. And it's not in this group because it's a semi-regular.

So I guess, Plato gets credit for these. And Archimedes gets credit for the semi-regular one. And there's a whole bunch of, I don't know, 14-- or it depends on how you count. So these objects are all their own mirror images. But when you get to the semi-regular object, there's one where the mirror image is actually different. In other words, you can't rotate it to bring it into alignment with itself.

And so if you count that, it's 14. If you don't count that, it's 13. And the soccer ball is somewhere in here, the icosadodecahedron. So what's the difference? Well, the semi-regular ones still have facets that are regular polygons-- so triangles, squares, dodecahedrons, hexagons, et cetera.

But you're now allowed to have more than one type. So over here, once you decided on triangles for the facets-- they all have to be triangles-- over here, you can have a mix. So the soccer ball is a mix of hexagons and five-sided figures. So that gives you a larger variety.

But think about the fact that they all have to have the same edge length. So for example, if you had an object where you're mixing triangles and squares, since they have matching edge lengths, the facet area is different. And there's obviously some formula for that, the area of regular polygon in terms of the number of edges and the edge length.

OK, so that means that over here, we had equal cells. And so it made it very easy to have a uniform sampling. Over there, it's a little bit more tricky because, now, we've got to deal with the fact that the facets are not equal in area. The good part is that there are more of them.

So that's on the sphere in 3D. We're talking about a sphere in the 4D. And so Plato and Archimedes didn't think too much about those. So we can't resort to them. But those figures come into play again because in 4D, we're talking about rotations. And we we're looking for regular patterns of rotations in this space.

And so one way we can think about that is in terms of rotations of these objects. So let's just do one and see how that goes. So the aim here is, again, to find some methods for uniformly solving the space of rotations so that if, say, we do a search-- so we've got this object in the sensor, and we have a model in the library, and we're trying to figure out, is there some way of rotating this to bring it into alignment with that?

And that could be part of recognition, that we have multiple objects in the library. Which of them can we match? But of course, we can't match them directly. We first have to get them rotated.

And then, how do we pick the rotations? Well, to be efficient. We don't want to be sampling parts of the rotation space more densely than other parts. That'd be the equivalent of spending all the time searching around the poles. And so we're trying to find a uniform way of sampling this slightly hard to imagine space.

But let's start simply with the hexahedron and think about its rotations. And well, first, there's the identity. So what do I mean by rotations? I mean by rotations that you take the object, then you rotate it, and then the faces line up with the faces, and the edges line up with the edges, and the vertices line up with the vertices. OK, so the identity-- so in terms of Quaternions, that looks like that. It has a 0 vector part and a scalar part of 1.

Then we can think about a rotation, let's say, about x through, let's say, pi. And so that's going to be, let's see, cosine pi over 2 is 0. Sine pi over 2 is 1. So that's that Quaternion. So that just turns it 180 degrees about the x-axis. And naturally, I can do the same thing about y.

So I've already got four distinct rotations that line that object up with itself. Well, I don't have to rotate 180 degrees. Oh, well, how about minus 180 degrees? Well, of course, that's the same, right? So that doesn't add anything. But I could think about rotating pi over 2. So let's try that.

So pi over 2, that's going to be cosine of pi over 4, which is 1 over square root of 2. So I get that. And now, here, I could talk about minus pi over 2 because these two aren't the same. So this one is not the negative of that because the real part hasn't changed sign, whereas with these guys, if I flip the sign of x, I get a different Quaternion, but remember, minus q is the same rotation as plus q. So I flip the sign of that to get a different Quaternion, but it's the same rotation.

OK, so I can do this for x, y, and then I can do it for z. So I get six of those. I've got four there. So we have 10 rotations, 10 samples of the space that we're interested in. And so the question is, are there more? And there are two different ways to proceed. The one is geometrically. Look at the diagram. How many other ways can I rotate it?

And the other one is to just think of this as an exercise in Quaternion multiplication. So let's just do one of those. So for example, I could pull out of that table. Now, I can try and generate the whole group just by multiplication, take what I've got, take two pairs. Let's take-- because it's not very interesting taking the identity, but so let's take something else.

OK, according to our rules for multiplication, I get 0 minus x dot y, which is 0. And then we get 0 times x plus 0 times y plus x cross y. And so that's 0 and minus z. Hmm-- no. Well, that's already in the table, right? So that doesn't add to the table.

So let's take something else. Let's take some of these. 1 over the square root of 2, 1, x, 1 of the square root of 2, 1, y. OK, that's going to be more interesting. 1 times 1 is 1 minus x that y is 0. And the 1 over square of 2 gives us 1/2. And then the vector part is 1 times y plus 1 times x plus x cross y. So that's 1/2, 1, and plus x hat, y hat, c hat. So that's a new one.

So the axis of rotation is 1, 1, 1. And if you want to make it a unit vector, it's 1 over the square root of 3. And that's not in this table. And the angle is cosine theta of a 2 is a 1/2, so let's see. That means theta over 2 is pi over 3. So theta is 2 pi over 3. So that's an interesting new rotation.

If you look at the cube, that corresponds to one of its corners. So we're taking the origin and connecting it to a corner, so that's the axis. And we're rotating by 120 degrees. And you can see that that's a rotation of the cube that does bring us back into alignment with ourselves. If we rotate 120 degrees about that axis, everything falls back into place.

And so we're just about out of time. But we can proceed two ways. The one is look at that example and some others. And we get a total of 24 rotations. Or if we're more mathematically inclined, we can go this way. Or it's fun to write a little program that just implements Quaternion multiplication and then builds this table by taking pair-wise products and seeing whether you get something new. And eventually, you'll run out, and you have 24 of them at that point. OK, next time, we'll start talking about relative orientation, which is more relevant to binocular vision.