PROFESSOR: OK, welcome back to 6849. So last lecture, lecture three, we were talking about local foldability and some complicated flat folding, like a flapping bird here. We were looking at a single vertex and locally around that vertex what properties it would have to have. And we saw Kawasaki's theorem which characterized the angles. And without a mountain valley assignment Kawasaki was all you needed. The alternating sum of angles should be 0. And given a mountain valley pattern, locally we characterize things as a sequence, anything you can do by a sequence of crimps. So sort of a similar version to lecture two, which was about 1D flat foldability. There we needed crimps and end folds. Here we needed just crimps, which is easier.

So I'm going to jump into questions. And in particular, this is an opportunity for me to talk about the main thing that I skipped last class, class two, because it relates again to this lecture, which is how do we do this algorithmically? So it's one thing to say, oh, just do crimps and end folds till you can't anymore, and then when you run out of stuff to do, if you're done, you're done, otherwise it's not flat foldable. That's true. But the obvious way to implement that algorithm is to sweep over the crease pattern, look for any crimps or end folds. If you find one, do it, and then repeat and keep sweeping. And that would take quadratic time, because in the worst case every scan you have to look through the whole pattern and at the very end you find what you need to do. So after about n operations-- n is the number of creases in your pattern-- you find what you need to do, you do the operation, and then you repeat. So n plus n plus n, n times, is n squared. But you can do better and it's been alluded to in the lectures, but it wasn't covered. And so I wanted to cover it, because there's actually a really simple way to do it. There's one version in the textbook, but there is a simpler way. So I want to talk about that.

This is a new way we just invented last week? This week? I forget. So I'm going to first talk about it in the 1D scenario, but it's pretty much the same for both. 1D's a little bit easier to think about, though. So 1D mountain valley pattern something like this. So I'm going to follow the same approach, which is sweep left to right until I find either an end fold or a crimp that I can do. So in this case, maybe this would be the

first thing that I find sweeping left to right. That's a crimpable pair-- or crimpable segment, I'm going to call it-- is just one that is locally smallest. It's less than or equal to its two neighbors in length, and it has two different assignments, M/V or V/M. So that's a crimpable pair, because it's shorter than those two guys. So that's the basic algorithm, but what I do after that is going to be a little bit different.

So I want to search left to right for a segment that's either crimpable or end foldable. And there's two situations. If I don't find any operations to do, then we stop. And if there are any creases left we know that the resulting thing is not flat foldable from what we proved in lecture two. If there's no crimp or end fold, you're not flat foldable. But if you do find something, then do the fold. This is so far pretty obvious.

I'm going to draw the picture for a crimp situation. So in the crimp situation-- these are the previous and next creases-- we have these lengths x, y, and z. We know that y is less than or equal to z and is less than or equal to x. And after we do the crimp, it looks like this. So we have x, y, and z here, but we're then going to fuse this material together, because these creases are done with. We don't really care about them. The new length that we get is x minus y plus z.

OK. So that's what I mean by do the fold. And we'll also call this sort of merging the segments, meaning replace-- so normally we have a sequence of lengths-- replace x, y, z with x minus y plus z, remove these two creases which were, say, M and V.

Now we want to continue. And the realization is that we don't have to start over our search in searching left to right for a crimpable thing. If we went all the way through the pattern and then finally found a crimp at the end, should we start way back here? No. There's no point. You might as well start basically where you just were. Not quite. You have to go back one step. So the next step in this algorithm is go back one step, and then continue the search. Back means left. And the search is this line. OK. So that's an algorithm, a little loop there.

So in this example we-- the point of what's going on and the reason this algorithm is correct is we just modified these three segments. And we replaced these three segments with a single segment which looks something like that. And the rest, the

parts to the left and to the right are the same. I still claim we need to back up one segment and look at this one again, because now potentially this one might be crimpable whereas it wasn't before. Because we changed this length, it may have gotten longer potentially. Or shorter. It could have changed. So this pair may be crimpable whereas it wasn't before. So just to make sure, we'll go left one step, and then we'll check, is that segment crimpable? Is this one? Is this one? And just keep going.

And I'm guessing this pattern is not is not flat foldable because of these two. We could check. Did I miss one? V, M, M. M is the little thing. So I think this is going to crimp. And then this will get end folded. But still this is a problem because of the two M's. So this will not flat fold. We'll be left with something like this, which is the two M's and then we say, oh. We'll reach the stop case. Can't find a crimp or and end fold and there are still creases left. And so we know that we're not flat foldable.

But the reason I'm doing this fast resume of the search, continuing the search from one step to the left, is because we get a good running time. I'm going to use running time notation of order n. This means some constant times n. We don't really care what the constant is, but the growth is linear in the number of creases that's n. And the reason it's linear is that the number of rightward steps that we make is going to be equal to n plus the number of leftward steps.

Why is that? Because overall the search is going left to right, and if it doesn't find anything it just takes n steps. You look at all the creases or all the segments. It's basically n, n plus 1, whatever. For each one, you check is it crimpable, end foldable. That takes constant time. You're just comparing a couple of numbers. So I'm counting the number of rightward steps. The trouble with the search is that every time I find a fold to do I go back one step and so I'm kind of losing progress because I go backwards. There's actually two reasons why that's OK. One is you're also decreasing n at the same time, because you're replacing 3 things with 1 thing, so n goes down by 2. But also, in order to make a rightward step either it's in the full search or it's because you went back one and you have to go right again. So this is true. And the point is the number of leftward steps is also at most n, because every

time you do a leftward step you did a fold, and there's only n folds to make. So-- it's actually less, because crimps do two at a time. But the point is this is at most 2n. And so the number of rightward steps we make is linear. And so the overall running time is linear. For the algorithms people, this is a very simple amortization argument. We're charging the leftward steps to the folds that we're doing.

So that's an easy way to do 1D flat foldability testing for mountain valley patterns.

Yes?

AUDIENCE: Why do you need to go back a step? So, the segment that you labelled x, its new length is x plus the quantity z minus y, and z is greater than y. So it's x plus a positive number, right? So it's only increasing in length, not decreasing.

PROFESSOR: Only increasing. But that's of interest, right? Because if you had something that wasn't crimpable because this was too short, and then-- OK, here's an example. Good question. I hadn't thoroughly checked. So we scan. This guy's not crimpable because this guy's too short. Then we reach-- suppose then we jump to this guy. I'll do valley mountain to make this definitely not crimpable. We look at this guy. We do the crimp. And now suddenly we have this nice big length. And so now this pair is crimpable. Good. So that's why we need to go back a step. After we crimp this guy, this one becomes crimpable if it wasn't before. Yeah. I could have easily believed that this step wasn't necessary, but it definitely doesn't hurt and it is indeed necessary. The key is that we don't have to back up more than one step, because we're only changing our neighbors, basically.

Other questions about this algorithm?

OK, well once we have this for-- this is really for lecture two material, we can adapt it to lecture three material, which is the circular case. Instead of having a line we have a circle of paper. In that case, we only need to look for crimps, and so we do the same thing for crimps. I don't think I really need to write this down, but algorithm for a single vertex mountain valley pattern. It's basically the same algorithm. Instead of wherever you see left and right you replace it with clockwise and counterclockwise

going around in a circle. There's no obvious starting point you just start at an arbitrary segment, which in this case is an angle of the crease pattern. Maybe I should draw a little crease pattern, just for a picture.

So you start, let's say, at this segment. You see is this pair crimpable. If not continue, let's say, clockwise. Keep going. If you ever find a crimpable pair, like these two guys-- maybe this is mountain and valley and this is a locally smallest angle-- you do the crimp, meaning you replace this angle x y and z with x minus y plus z, just as before. And then you step counterclockwise one step. And the point is the invariant you're maintaining at all times during this algorithm, the interval of the segments from the very first segment you went to, up to but not including the segment you're currently looking at, those are all guaranteed not crimpable at the moment. And so when you do a crimp, that may invalidate this one, and so you have to step backwards because you're not sure whether that one's crimpable anymore. But you maintain that invariant. And so when if you ever get back to the original angle that you were considering, the original segment, then you know that in fact everything is not crimpable, and then you're in trouble. Maybe.

Except for this issue which I also forgot about in lecture three-- I mean, I didn't forget about this time, but I forgot about it then-- which is in the base case. You can never actually do everything by crimping, because at the very end your hope is that you have a cone with two creases that are both the same orientation. That is your goal. You know that there's two more mountains or two more valleys and crimps pair them up. So you hope that you end up with the situation. If you do, and these two angles are equal, then you're flat foldable. You have that one last fold to make. Otherwise, if you have anything else, you're not flat foldable. That's what we proved in lecture three, is that crimps are enough to get down to this situation where you have only two creases left, and that's what's foldable when you only have two creases.

OK so this is becoming a cone as soon as you do operations. It's just like the analysis we did. But this algorithm will run in linear time for the same reason. The number of clockwise steps equals at most n plus the number of counterclockwise

steps. And so I guess I should maybe write "at most." And so this is linear time as well. Any questions about those algorithms?

Yeah.

AUDIENCE: For the simple one, could you run into some edge case where the very last crimp you considered makes the very first one, which wasn't crimpable, crimpable and then you have to go around the circle again?

PROFESSOR: Ah. OK. Good question. Right. So we said, OK, this is guaranteed not crimpable, but if you then crimp the very last segment this one may become crimpable, and then that may propagate and force you to go around a second time.

AUDIENCE: So if that happens at each step, then at each step you go all the way around, and it could become quadratic.

PROFESSOR: Ooh, interesting.

AUDIENCE: But I'm not sure if it's possible for that to happen every step.

PROFESSOR: Good. Well, this is the nature of new algorithms. Yeah.

AUDIENCE: The set of possibly crimpable things increases by at most 1 every time-- or at most 2 every time you make a crimp.

PROFESSOR: Yeah. So certainly you will only have to go around at most n times, but a quadratic bound overall is not very exciting.

AUDIENCE: So it stays linear. The interval in which things are possibly crimpable expands by at most 2 every time you make a crimp.

PROFESSOR: I see. OK. So this is a different algorithm, unfortunately. But you maintain the interval of things that are guaranteed not crimpable and you can look at both ends whether you can crimp something. If you can, fine, you do it. You shrink the interval a little bit. So in general your interval is-- I have colored chalk, I should use it-- your interval is an interval. It starts somewhere, ends somewhere. And you're checking is

this guy crimpable. If it's not crimpable you can extend the interval a little bit. If it is crimpable you do the fold and you actually shrink the interval a little bit. But every time you shrink the interval, you did a fold, and that only happens n times. Every time you grow the interval, you grew the interval and you can only grow n times. Good.

So that will clean up this situation. If this ends up being crimpable, you just shrink the interval from the other side. So I guess if I wanted to just tweak this algorithm I would change the notion of first. So I said, oh, this guy is the first one that I visited. But if I end up crimping this guy, I have to advance the first to be the very next interval. Good. Thank you.

All right. I'll correct that in the notes as well. It was almost correct. Think now that should be correct. Thanks for checking. And now we can do it in linear time.

There's a different algorithm in the textbook, which uses more data structures. This I like because it's very simple. It's just, like, storing two pointers and that's it. Other questions?

OK. That does algorithms. That was the new material I wanted to cover. Then there is the other-- the actual algorithm that was briefly described in class is this local foldability algorithm. So you have now not just a single vertex, but you have a whole crease pattern. You'd like to assign a mountain valley assignment to it that is at least locally good, that when you run this algorithm at each vertex it gives the right answer, it says that, yeah, it's locally foldable for each vertex. It doesn't mean the whole thing will fold flat, but it's a start, at least. It's a necessary condition for that.

So I just want to give you a few examples of this algorithm, because it is confusing. I didn't do any examples in lecture. It's always been the most confusing part to me, and especially this notion of merging cycles and paths. So for starters, these are the examples in the textbook. They're pretty simple, but at least they'll get us warmed up. So the crease pattern is the bold black lines. It's two of them. And in this case it's the generic case, so there's a unique pairing here. There's only one crimpable pair, which are these two guys. This is the only locally smallest angle. So those two

have to be crimped first. One's going to be a mountain, one's going to be a valley. So we would write not equals in this blue thing to represent that constraint. This will be all that's left. And so these two guys have to be equal. It's symmetric, so it looks the same all the way around. And so when you see, OK, these two guys have to be not equal, but also these two guys have to be not equal, but also these two guys have to be not equal, this is what we call a cycle of constraints.

In general, you get paths. Like, this one starts and ends at infinity, so these two guys have to be equal, these two guys have to be equal. They could both be mountain, both be valley, doesn't matter. These are the only constraints. But this cycle-- cycles can be problems, and here because it's an odd cycle of not equals there's no way to assign it. If you say mountain, valley, mountain, then these two guys are both mountains, which violates that constraint. In general, the number of not equals should be even in each cycle. If it's ever an odd number of not equals, you're in trouble.

Here's an example where the cycle is even, but we end up with an equals here, and so the number of not equals is still odd, so this is also bad. I mean, it's just replacing this segment with two equal creases. Still can't assign a mountain or valley. But these are kind of simple examples. These are cycles and everything was uniquely determined here. You had to crimp this guy first. You had to crimp this guy first. There was no choice. And the tricky part of the algorithm is when you have choice, when you have multiple equal angles, you don't know what to crimp first. The algorithm just crimps one of them first, but then it might have to fix things. So I came up with a simple example where you have to do that. I think it will help clarify how this merging really happens.

So the example is-- it's hard to draw an equilateral triangle with accurate angles, but I'll do my best-- and then these are supposed to be right angles. OK. So that's my crease pattern. And some of this is forced. This guy-- at this vertex, this is the only locally smallest angle. It's only 60 degrees, these are 90. This is bigger than 90. So this has to be crimped first, and it's not equal. That means these guys are equal. We don't really care about those. And it's symmetric, so not equal, equal. OK. But

here we have a choice. There's two 60 degree angles. This does not look very flat foldable. I think I'd better add some more creases there. Got to have to even parity at every vertex.

So now we have a choice. Do we crimp this one first or do we crimp the other one first? At this point, it's symmetric, so I'll do this one first. This'll be my pairing. And there are two possibilities here. Either I crimp this one first or crimp this one first. The algorithm doesn't care. So let's suppose it does this one, because this is the bad one. OK. Now we have a cycle with an odd number not equals. So this is not possible to satisfy. The algorithm doesn't stop there. It says, OK, I have these paths. There's a cycle here that's a problem, then I also have a path up here. How does it go? There's a path that goes here, here, here, here. And there's a path that goes here. But really it looks at the vertices that had choice. It says, look, at this moment I had a choice between whether to fold this angle or this angle. In general, it might have been a bunch of equal angles. We see here that there's a cycle that the other choice would've involved a path.

And so we merge. In general, if there's two different things-- one could be a cycle, one could be a path, they could both be cycles, they could both be paths, whatever-- if I ever have the opportunity to join those two parts together I'll do it. So in other words, I do the other crease first. Let me draw that. What that means will become clear once we actually do one. So I want to do this first. The rest is the same. OK. So in that situation, what do my paths and cycle look like? Well, there was this path that started over here. It used to go like this, but now this vertex has changed. So it's going to do something different at that vertex. Now it goes over this way. I'm just following the constraints. Now those two guys are constrained to be different. And these are constrained to be equal. So lo and behold we merged a path and the cycle and we got a single path. The other paths remain the same. Just these two guys got interchanged. What we're doing is basically turning here and turning here instead of going that way and going that way. And whenever you have two pieces like this, and one of which is a cycle, you will do a merge. Merging can only help us, because they'll get bigger and bigger and bigger. The bigger these sets of constraints are, essentially, the better chance that you'll get the parity right.

I can never-- and this is argued in the notes, but few-- it never hurts to merge something, is the point. And if you're lucky-- if you, say, merge two odd cycles-- they will become even. If you merge with a path, you'll become a path. And so you're golden. Paths are always good. And so this thing becomes flat foldable, or at least locally flat foldable. And we can mark in a crease pattern, I guess. You could make this-- that's going to be hard to see, I think-- mountain, valley, mountain, valley, mountain-- oh, sorry-- equals, not equals, this is not equals, not equals, not equals, equals. These guys are free. So you can make one of them mountain, one of--- you can make them all mountains, I guess. Or not. OK. This is a locally valid flat folding. And it's hard to tell whether it actually works except by folding it. So here I made one. That's the top side, I think. Ideally I got the same crease pattern as here. The reds are mountains. Looks the same. And then--

It's clear that this pattern is flat foldable, right? You just simple fold here, and then you've got a single vertex and that single vertex is flat foldable. But this does not do that. It does a kind of twist. It's kind of a fun mountain valley assignment for it. So in this case it works. In general, you might get some weird mountain valley assignment that doesn't work. But something simple like this pattern, which has four vertices, always will.

Any more questions about local foldability? That gives you at least an idea of what the merges look like. It's hard to draw a huge example, but-- Yeah.

AUDIENCE: So the generic case, you don't really have a choice at any--

PROFESSOR: In the generic case you have no choices, and so you've got to-- you just check, does it work. And if it--

AUDIENCE: If you have an odd number of faces, odd number of sides, then it's not going to work?

PROFESSOR: If you have a face with an odd number of sides, that might be fine. It depends on these assignments, whether they're not equal or equals. Like if this were an equal

sign, then you'd be happy. And that would happen, for example, if you move these creases to be very small. I'd have to also make this one proportionally big, which is possible if I move this vertex way over here. I'll have a triangle. This will have a big angle. Then I can have a small angle here and these two guys will be made not equal and these two will be made equal, and then the parity's fine. So if you have a cycle with an odd number of not equals, then you're screwed. In the generic case there's nothing you can do.

When you have equal angles, when you had a choice, you go back and check whether the choices would do merges. If you do, you do them. Overall, this turns out to take a linear time if you're careful, because you can only merge so many times. You have at most n parts and each merge-- with some care. You need fancy data structures to get this to work. You need find stuff, but then you'll get linear time over all. Other questions?

AUDIENCE: So what's the definition of merging, again?

PROFESSOR: So the definition of merging is you look at every time you had a choice in running this algorithm-- which you have to not just run this algorithm but you have to maintain all the choices that you had made. So whenever you had two equal angles-- in general, the algorithm is you look at a sequence of equal angles. The algorithm maybe chooses the first one, but you could have chosen any. You see for each of the other possibilities, would that end up combining two of the components. So the constraints sort of join together either into cycles or into paths, like this guy. And we just check, if I do this other change, does it end up combining two of those components? Before I had 1, 2, 3, 4 components. Now I only have 3 components. So just see what happens. If that decreases the number of components I do it. And you can guarantee this never hurts you. Keep merging components until you can't anymore. And then either it works and you've got no parity problems or no cycles, or it doesn't work. If it doesn't work, there is no local foldable assignment.

Yeah.

AUDIENCE: When you say it's linear, are you counting vertices or edges?

PROFESSOR: Let's say number of vertices plus edges. That's the safe way to define n and then linear in that. Or if you count the endpoints here it doesn't matter whether you just count vertices or edges. The sum of the two is always safe. So n usually just means the size of the input, and those vertices and edges, so why not just count them both. Other questions? Cool.

So that's local foldability. I just have a few more little things and then you can ask more questions. Oh, sorry. I have one more example. I forgot.

This is an example where everything works fine. You don't get any cycles. But it's kind of a fun example. The crane. I'd never analyzed it before, so I spent the time to draw one of these pictures. So first I just put a bunch of circles down, and I look for things that are forced, just because that is more interesting. I tried to make it as bad as possible. It turns out I couldn't make it that bad. But this guy's forced, because it's the only locally smallest angle. And then it's symmetrical around. So this guy's forced. Four of them should be forced. If I advance, yeah. This one, this one. Those are all forced to be not equal. The rest are equal. The other guys have ambiguity. And I tried. I thought this would be a great place to find the cycle and then we could resolve the cycle, because I know this should work in the end, but I couldn't make a cycle. It's not possible, because there's sort of this cut point here. And you once you go to one side of the pattern, you can't come back to the other side.

I could make a really long path though. So I chose-- there's a lot of choices here, but the algorithm just chooses one. I tried to make the longest path I could. It's kind of a fun puzzle. Not that hard to solve. It gets a little hard to draw these pairings. Here these two guys are paired together as not equal. We crimp that first. Then let's say we crimp these two. Then we've got two angles here, each of which I think is 90 degrees after you do the crimps. And so these two guys have to be equal. That's what this notation means. So here there were, I think four different possibilities. And if you trace all the paths, you get these guys. So there's some simple paths out here, but then there's this purple path in the middle, which starts here, goes up here, over here, over here, along this crease, over here, here, here, here, up there,

here, here, there, back, forth, and then it escapes in the corner. So you get a mountain valley assignment out of this. I didn't test whether it was possible. Probably not. Anyway, you get something, and it might be possible. Exercise for you to try at home. All right. That took some time to draw. Cool. That's local foldability.

Now with the Kawasaki condition, I mentioned briefly in lecture three that-- We were talking about convex cones, where the amount of material is less than or equal to 360. Someone pointed out and said, oh, out of paper you can't make something more than 360. And it's true if you start from one sheet of paper. But if you start with multiple sheets of material and join them together, like sew them up, like in a T-shirt, you can get non-convex cones, meaning more than 360 degrees of material here. This is, let's say, 270, let's call it? So you double that, because there's a front side and the back side of the T-shirt. Easier to see in a 3D one. You've got more than 360 degrees of material in the armpits. So, yes, it's true.

And what happens, as I mentioned briefly, is there's a new case. There's sort of two situations when you have a kind of a mess of material like this, more than 360 degrees of material. There are two possibilities. Either you end up folding it to lie in less than a full circle, in terms of the boundary, and then it's just like here. The alternating sum should be equal to 0. Or you end up folding it so that it encompasses an entire circle, and then you end up with this alternating sum of angles being plus or minus 360. And it's in the textbook. It's not so easy to prove that that's all that happens, but you-- Basically, you can't twist multiple times, because then you'd end up with a crossing. So it's not that much harder to analyze these kinds of situations. You can do it.

But this is a great excuse for me to show cool ways to fold T-shirts. How many people seen this video? most. It's like six years old. Have you ever tried it?

[LAUGHTER]

I brought an extra little T-shirt here. This is totally for fun. You pinch here and here. Bring this side over here. You pinch. You do a nice flourish, and then you get your perfectly folded-- I didn't do it perfectly-- T-shirt in one motion. As they say, in two

seconds, and it works for us. So it's a great T-shirts.

And for fun, here's the-- this is like the high tech way to do it. There's actually a whole bunch of T-shirt folding machines. You push the button. It's a bunch of simple folds, actually. So it's a nice little simple folding machine.

[LAUGHTER]

It's kind of fun to watch. Seems like a lot of set up time. You have set up time with the fast method also. So It's kind of fun. The same machine. you can also wrap them directly into bags. Question?

AUDIENCE: Have you in clothing stores they have non-mechanical versions of those that are just plastic boards.

PROFESSOR: And you just--

AUDIENCE: That you can flop over so that all your T-shirts are folded the same.

PROFESSOR: It's not automated. Yes. You can actually buy-- I've only seen them in stores rarely. They're usually in the back room. But you can actually buy this folding machine. It's like a giant piece of plastic with exactly-- maybe only three creases, I think typically. And then you just do them in some order manually with simple folds, and it does make really nicely folded T-shirts. Nicer than I could fold by hand, anyway. All right. That was just for fun. While we're on the topic of T-shirts.

Next question is about higher dimensions. So I mentioned briefly, yeah, you can do higher dimensional origami. Not much is known about it. So a natural thing to ask about is flat foldability for higher dimensions. And there are exactly two papers about this. Well, maybe even just one. The old one is this paper by Kawasaki-- same guy as Kawasaki's condition-- and this is in this book. This the first-- this is a hard book to get a copy of, *First International Meeting of Origami Science and Technology*. These days it's called *Origami Science, Math, and Education,* OSME. And this T-shirt is from the latest one, which was in Singapore two years ago, I think, 2010. Next one is in a year or two in Japan. So this is the very first one. This

is before my time. And there's this paper. It's a translated Japanese paper, so it has a few typos. We also have the Japanese original if you're interested in reading this some time.

You see, for example, here is folding a regular piece of paper in half. Here's folding a 3D solid of paper in half. A little harder to imagine, but there it is. And let's see, what's in this paper? There's a definition, although I would really call it a necessary condition. I think this is not a good definition of flat origami in 3D. But there it is. This is the definition. What it says is basically, locally everything works out. When you do a fold, it's like a reflection. If you do it instantaneously, this is like reflecting this piece through the line for flat folding. This is like reflecting this piece through the plane. And so one condition you have is that if you-- it's easier to look at one of the examples in the paper.

So you have this crease pattern, which you can draw by a bunch of planes, let's say. If you kind of walk around and say, OK, I go through this crease, which means I reflect through that plane, then I reflect through this plane, this plane, this plane, you can take any sort of path, any cycle through this world. In the end I should end up back where I started. Otherwise I'm ripping. And what that condition says is that if you take the sequence of reflections and you compose them, you end up with no reflection at all, that nothing moved. And so that's a necessary condition for flat foldability. I wouldn't call it a definition, although this paper did. Natural definition is more like what we draw in 2D for flat folding, where we add another layer, another dimension, and then guarantee no collisions in that dimension.

So same thing in 3D. It's just harder to imagine stacking up copies of 3D. Question?

AUDIENCE: Doesn't the reflection analogy only apply to simple folds, where the plane goes all the way?

PROFESSOR: OK. Good question. The reflection analogy definitely applies to simple folds. It also applies to non-simple folds. But it does require straight folds, which here mean flat folds, like planar folds. For curve creases, you're not really reflecting, because you can't fold a curve crease all the way. You can't fold it flat. But whenever you fold

something all the way-- so even in something like this, which is not a simple fold in the end-- you can check. Essentially, this is what we-- on the circle, when we said, OK, you walk this way and then you just change direction. That's essentially the reflection that's happening on the circle. If you live on the circle you're reflecting in that you're immediately bouncing back at that crease.

It turns out to hold in 2D as well. So if you look at the effect of this crease, it is that you're reflecting this part over that crease. I guess you can see it in the folding too. I mean, you go over this crease, then you hit this line, and then you immediately bounce back at the line. That is actually a reflection through the line. Good question. It's obvious for simple folds, but reflection actually works here as well. We'll talk more about that in the next lecture, I believe.

OK. I was looking up reference, any papers that cited that one, and there's basically one paper, which is this kind of a graphics/art/math paper talking about four-dimensional origami. It's basically some simulations and some examples. Here's the not folding all the way picture. A little harder to imagine. Here they're doing, I think, the 4D analog of a rabbit ear fold. Little hard to tell, but first they do a simple fold along a bisector. That I can understand. And then they do an inside reverse fold. And then you end up with 1/4 of the tetrahedron. So if you divide, this is the centroid of the tetrahedron. It's really hard to draw these pictures. In the paper, they actually showed two images, and if you align them with your eyes then you'll see a 3D image, which is of course a projection of the 4D thing.

They also have this one. Anyone have red-blue glasses with them? Then you'll see this in 3D. Here you can watch that later. This is their 4D analog of a flapping bird, I guess. It's pretty hard to see these pictures, though But this is the state of the art in 4D origami as far as I know. It's kind of neat to find. This is a pretty recent paper.

Questions about higher dimensions? Lots of things are open. For example, single vertex flat foldability could be a neat problem. I don't think-- It may be that Kawasaki's condition from that paper is enough. I don't know. Maybe not.

OK. Last question, just kind of a nice place to end is, why are we spending all this

time on flat foldability. Flat origami's kind of boring. And so why do we spend all this time with it? It's good to check why are we doing this. One answer of course is that there's interesting mathematics here. It's kind of a natural question. Why not? That's the math cop-out answer. But there are actually a lot of good answers as well. One answer is that flat origami actually is pretty cool. There's a whole world of flat tessellations, where you fold a repeating pattern in your sheet. This is from a rectangle of paper. What you're seeing here is a shadow pattern. It's held up to a window, and so you see this shadow pattern, which looks an awful lot like a 6 by 6 by 6 Rubik's cube. That's the design here. You could make it even larger if you want.

Another answer is, well, maybe I need to store stuff. And while I can unfold things, it's kind of big when it's unfolded. If I could fold it all the way flat-- maybe I care about the 3D shape, but if I could fold it all the way flat, then it would be easier to store. It's going to be smaller, more compact, I can roll it up, I can do lots of things. Airbag folding is one example. Here you actually start with a 3D shape. There's no sort of unfold. It's been sewn into a 3D shape. And then you want to collapse it into some nice flat shape for storage in your steering wheel, or the side of your car, or whatever. And this is an example of folding using flat origami designs. This crease pattern is based on the tree method, and other stuff we haven't covered yet, fold and cut. So that's another. That's sort of the practical answer.

And then there's more mathematical answers, deeper things. Even if you don't care about flat folding by itself, it turns out to relate to 3D folding as well. This is a paper that's very cool. We have a guest lecture by Tomohiro Tachi coming up. I haven't scheduled exactly when it will be, because it's in the in the box. It's on tape. So we'll be watching that at some point. He was actually just visiting a couple weeks ago. And he has this cool theorem which says that if you have a plane or quad mesh-- so a particular kind of crease pattern, every vertex has four incident increases, every face is a quadrilateral-- and it's flat foldable-- so you may not care about flat foldability, but you need this Kawasaki condition to guarantee that this will work-- then it has a rigid folding motion, if and only if it has a 3D state.

Let me show you some examples. So this a kind of a classic origami called miura-ori. And it has this cool property that even if these panels are made up out of rigid material like sheet metal or plastic or whatever, if these are hinges you can still fold it. And this is an animation of it folding. And indeed it folds flat. And what the theorem is saying is if you can build a 3D picture like this, any one of these, that guarantees that there's this folding motion-- not necessarily all the way to flat, but at least part way to flat. And often you can get it all the way to flat.

And so Tomohiro's developed software that lets you start with something you know works-- so this is a miura-ori that's been folded partly-- and then just start pulling on the vertices and messing it up and just changing the shapes. So this is not a folding. You're changing how the paper fits together. You're changing the crease pattern. As you do that, he keeps track of the crease pattern and he adds constraints to make sure that it stays flat foldable. The result is you end up with a 3D embedding, a 3D folding, and you have a flat foldable crease pattern that goes with it. And that guarantees that you get a motion. Those two things. So this technique of having flat foldability lets him design crazy crease patterns that fold into whatever shape he wants with a nice rigid motion where all these panels stay rigid. So that's another motivation for flat foldability. It's probably used elsewhere as well, but this is kind of the coolest, newest example I know, where it's just a condition in the theorem, and in order to check that condition you need to understand flat foldability.

Other questions? That's all I have for slides. Cool. That's the end of class.