PROFESSOR: All right. So, lecture 13 was about a lot of things. We had algorithms for the carpenter's rule theorem. We had three of those. Let's see, there's the CDR strain of pseudo-triangulations, which we're going to talk about a bunch today. There's the energy method, which we'll talk about a little bit. Then there was locked trees. I'll give you-- there's one new result in locked trees I'll tell you about. And then we had the last topic, which was-- cheat-- oh, right. Last topic for today is four dimensions, which wasn't covered in class.

And what we did talk about in class is 3D. Essentially no updates there, although open problems are still open. Fairly challenging. But 4D we didn't get to talk about at all. And that's not too hard, how to unfold any 4D chain. So that'll be fun. Also 5D chains we'll get-- and 6D and 7D, but not eight. Any dimension.

First question for today is why do we care so much about expansiveness? And this is mostly-- mostly, I want to reference next lecture, which will use expansiveness. But, indeed, the original point of expansiveness was just a convenient way to ensure self-intersection. Essentially, before the idea of expansiveness, we had this idea of a really complicated chain. *that's* not very complicated but for the carpenter's rule problem, we had this idea that, well, how do we distinguish between a linkage that is locked within Epsilon? However you draw it, if it's not self-touching, it's going to jiggle a little bit. Versus something they can eventually unfold.

And there was no good way to distinguish between just jiggling a little bit and unfolding all the way. Which was frustrating, and expansiveness gave us a way to guarantee that. Not only did it avoid self-intersection, which is nice. If you expand, you don't self-intersect. But it also gave a way that things would actually be rigid if it was impossible to do. And so you couldn't move at all. That reduced it to a rigidity theory problem or, really, a tensegrity theory problem. Which was kind of what this problem needed. That's why it was open for 25 years before we could prove it with tensegrities.

So, that's one answer. But, in fact, now that we know expansive motions exist, they're really handy. They let us prove that the energy method worked, for one. And next class we're going to use them to add thickness to our edges, add polygons or regions attached to every edge. And if you have an expansive motion, it turns out under certain conditions, this thing will still work. Whereas with non-expansive motions like those produced by the energy method it won't work in general. So initially it was convenience, but it turns out to be handy mathematically.

Two, we've seen one example just proving the energy method That's actually our next topic. We will see another, next lecture. But a few people asked about the proof of the energy method, so I wanted to review it. How do we know we don't get stuck in a local minimum? So this is an obvious question to ask if you're familiar with gradient descent.

If not, here's a kind of picture. Imagine you have some energy landscape. In reality, this is a surface drawn over n times d dimensions, but I will assume n times d equals 1 because I can draw it. So here's configurations. And then on the x-axis, or in general all the other axes. And then the y-axis, which will always be one thing, is the energy function. So sometimes the energy is high. At some points it's going to actually shoot off to infinity. That's where you'd self-intersect. Remember, the energy function was sum over all edges. The w sum over all vertices. U of 1 over the distance between u and dw. So when a distance goes to 0, the energy shoots to infinity. Those are the points we want to avoid. So those are bad things.

But if we start at some non self-intersecting configuration, we'll have some finite energy. And the idea is, well, if we followed the negative gradient, that will decrease energy as fast as possible. And from here it's going to basically follow this trajectory. And it would get stuck in this little well. This is called a local minimum. But *this* local minimum might be lower. And so the worry is that it depends where you start. If I start over here, I'll fall down there. But if I start in this kind of well, I'll end up at this kind of accumulation point. And what I really want is maybe some kind of global energy minimum. That's the usual issue with gradient descent. That's why people don't like to use gradient descent sometimes. Because it gets stuck. Hey, Jason.

However, with this particular setting we actually proved that the energy method does not get stuck in a local minimum. And I'll just review that proof because it went by quickly in lecture. So the idea in our setting is we know the carpenter's rule theorem. Carpenter's rule theorem tells us that any configuration where we're not done-- and I'll start just by thinking about a single chain, maybe an open chain or a closed chain. Any chain that is not already straight or convex, not only can be unfolded, but has an expansive motion to unfolding.

Expansive motion tells us that if we look at any distance between a vertex and an edge, this increases. That's not how we define expansive. We define it between every pair of vertices. But it turns out if you have-- this is actually something we needed for non-self-intersection. If you have a vertex and an edge, and you know that-- well, this edge stays fixed length because it's a bar. We know that these distances increase. And this distance increases by expansiveness.

Then, in fact, this point must go away from this bar. You can draw an ellipse. If you held these fixed, it would move along an ellipse. Or held the sum fixed. But you've got to go outside that ellipse. This guy's got to get farther away from-- every vertex, it has to get farther away from every bar if the pairwise vertex-vertex distances increase.

So if you take this expansive motion, you know that it is an energy-decreasing motion. So what that tells you is that any configuration that's not already straight or convex has an energy-decreasing motion. So this tells you, you never get stuck in a local minimum. Because what's a local minimum in one dimension? It's a place where you cannot go down from here. And those are the places we're worried about. And the claim is the only places where you don't have a downhill motion are places where you don't have expansive motions, and therefore the configuration must already be straight or convex. And then you're done.

So you don't have to worry about getting stuck with the energy method. There is one tricky part which is, if you have a linkage that consists of multiple pieces--

maybe you have some open chains, some closed chains, you can have some guy stuck in here. They won't necessarily straighten out, but-- there's a couple kinds of things you could do. You could unfold this chain, you could unfold *this* chain, or you could just let them fly away from each other.

All of those things will decrease energy. And so the worry here would be that these two linkages fly apart and never really unfold locally. That would be bad for us because we're always decreasing energy. There's always a decreasing flow, but we may not actually finish. We might not finally make this. Our goal in this case, remember, is to make it outer convex. Meaning that all the outermost chains are straight. All that outermost polygons become convex.

So we want to argue this. And I'm just going to sketch the proof here. Essentially, when you have multiple components, once they're very, very, very far away, their contribution to energy, just by all the pairwise distances across between two components, will be very, very tiny. And the claim is actually the energy method won't get you all the way to an outer-convex configuration, but it will get you within some Epsilon of outer convex. You get to choose whatever Epsilon you want.

And so when these guys are far enough away, their contribution to energy is much less than Epsilon. Once they're, like, n squared divided by Epsilon away or something, the contribution between them will be far less than Epsilon. So, what really matters in energy is all the distances inside the linkages, inside the components. And we're following the gradient motion, which decreases energy as fast as possible. And so you can prove that it's better-- if you want to decrease energy as fast as possible, it's better at some point to unfold the pieces. Not to just let them continue flying apart.

So waving my hands a little bit there. But if you know that you're at least Epsilon away from being done, then the energy will be fairly high within the components. And therefore it's worth unfolding the components. But it's especially clear for one component. All right. Any questions about that.? That's why the energy method works.

The next set of topics is related to pointed pseudo-triangulations, which are indeed very cool. There's dozens of papers about them. They've spawned their whole little sub-field of computational geometry. We touched on them a little bit in lecture, but there's a lot more to say about it. In particular, why they even work for the carpenter's rule problem. I didn't make clear, but I would like to. So a couple things about them. They were actually originally invented in 1994 or the first used, I guess, in 1994. At this point, they were called geodesic triangulations for a particular reason.

And so for the carpenter's rule problem, we had a polygon and then we pseudo-triangulated the inside and the outside of the polygon. Here we're just pseudo-triangulating the inside. In that setting, they're call geodesic triangulations. And it's for a particular data structures problem, actually. This is sometimes covered in 6851, which is Advanced Data Structures. You can check out online.

The goal is to decompose this polygon in a kind of balanced way. So first you choose a pseudo-triangle that's kind of most in the middle of the polygon. Then you recurse on the sides. Actually number one is this one. And if you do this, it turns out if you want to walk from any point in this polygon to any other point in the polygon, you only visit log n pseudo-triangles. Just a nice, small number.

And this is a particular problem called ray shooting, where you imagine you have a laser to shoot in some direction. The polygon-- you want to know when it exits the polygon, when it hits the boundary and where it hits. One way to do that is to walk through the pieces here, the pseudo-triangles. And if you only have to walk through log n of them, it turns out walking through a single triangle only takes log n time. Total amount of time is log squared n.

So this is a pretty good way to do ray shooting in polygons. Which is nice. So that's where they originally come from but they weren't used very much until later. So let me tell you some nice properties about them. Pseudo-triangulation has 2n minus 3 edges and n minus 2 pseudo-triangles. Always.

I should say, if you draw a pointed pseudo-triangulation, remember pointed means

at every vertex of the pseudo-triangulation. You have some angle which is-- you have a reflex angle bigger than 180. This is the pointed property. Pseudo-triangles-- all the faces look like this. They only have three convex vertices. All the other vertices are reflex.

So this is pseudo-triangle. Pointed pseudo-triangulation has both of these properties. And it's pretty easy to prove this by induction. I won't go through the proof here. It's interesting in the way that it is different from triangulation. So n here is the number of points. And I'm thinking about pointed pseudo-triangulations of the points. Meaning, I take some points and I add edges until I can't anymore. Subject to the pointed property.

So I add this edge, I'm still pointed. I add this edge, I'm still pointed. I add this edge, I'm still pointed. Can't add this edge. Can't really add that edge. Let's see if I can and this one and this one and this one.

Anymore? This one and now I'm done. I can tell that I'm done because at this point, all of the faces are pseudo-triangles. They each have three convex vertices. And as soon as you have that all the faces are pseudo-triangles, we can show that you can't add any more edges while still being pointed.

I'm still pointed because I only added edges that preserve pointedness. And this is a sort of greedy algorithm to construct pointed pseudo-triangulation. It always works. And we can count the number of edges. I won't bother, but it will always be 2n minus 3.

By contrast, if you look at regular triangulations-- so a triangulation would go all the way to the point where every face is a triangle, like this. And in that case they have more edges, obviously. In general they will have-- here is two more edges because there are two interior vertices. So, in general, it would be 2n minus 3 plus i edges for a triangulation, where i is the number of interior points. And it will have n minus 2 plus i faces, triangles.

Pseudo-triangulations are quite minimal in the number of edges. In particular, they

are minimally generically rigid. This is the number of edges I want for a nice rigid structure. 2 n minus 3 is what I should have. And indeed, pointed pseudo-triangulations are minimally generically rigid.

That's the next thing. Minimally generically rigid. This was the Laman condition. We saw a couple characterizations, but in particular, you can fairly easily prove that they satisfy the Laman condition. Why? Well, first thing to check is they have 2n minus 3 edges. That, I just claim.

Second thing to check is that if I look at any subset of the vertices, k of the vertices, they should only have at most 2k minus 3 edges among them. It should induce at most 2k minus 3 edges. Well, let's look at any k vertices. What are the edges that they induce? Well, the thing I know about them is that they will induce a pointed set of edges. Because this structure without the blue edges is pointed. If I remove edges from it, it will still be pointed, [CHUCKLE] right? If you have a big angle, you'll continue to have a big angle if you remove edges from it. It's a matter which subset of edges I look at, in particular, in the deuce subset. It will still be pointed.

If I have a pointed set of edges on some k vertices, I can use that greedy algorithm to finish the pseudo-triangulation. As long as you're pointed, you can add edges until you *can't* add edges, subject to the pointed constraint. At all times preserving pointedness. This is what I did to generate this triangulation the first place. If I take some subset of k vertices, I can do it again on those k vertices. When I'm finished, I will have a pseudo-triangulation. And that has 2n minus 3 edges. In this case, 2k minus 3 edges.

So I started with some pointed set of edges. I added stuff. I ended up with 2k minus 3. That means I started with, at most, 2k minus 3 edges. 2k minus 3. And so that's how you prove the Laman condition. It's kind of trivial, because pointedness is an inherited property. So if you believe in 2n minus 3, then you believe minimally generically rigid.

Cool. Even cooler is that the converse is roughly true. In some sense, all Laman graphs can be drawn a pseudo-triangulation. There's one catch, which is that you

need a plane area. Any planar Laman graph has a pseudo-triangulation as a realization. Realization was just a way to draw the graph in 2D.

Now, pseudo-triangulations are planar, meaning none of the edges cross each other. So you definitely need a graph that can be drawn in the plane without crossings. That's the planar constraint. But if you have a planar minimally generically rigid graph, you can always draw it as a pseudo-triangulation. So it's kind of a converse of this theorem. And a sense in which pseudo-triangulations are universal for planar minimal generic rigidity. Which is pretty neat.

I will not prove that theorem, but I will give you a visual sense of what the proof looks like. It's based on the Henneberg number construction. So we know minimally generically rigid graphs can always be drawn in a Henneberg way, either by adding new degree-two vertices, or by adding new degree-three vertices and removing one of the existing edges. And the claim is, you take any Henneberg construction. You can do it-- you can implement it in a pseudo-triangulation.

So you start up with your single edge, that's a pseudo-triangulation. You keep adding things. Preserving the fact that at all times is a pointed pseudo-triangulation. It's tricky, but for example, if you know that you want to add this red vertex next to these two, you find a place that will guarantee that this is a pseudo-triangle. This is a pseudo-triangle. It will be pointed for free because it's just a degree-two vertex.

So the challenge is preserving pseudo-triangles. And same for adding the degree-three vertex. This is harder, of course, but you need to-- you find a little patch here where, if you choose the point in there, you get pseudo-triangles on all three sides. And also, that removing the edge still preserves pseudo-triangulation. It's definitely not trivial to do this, but it can be done. And it's kind of neat that it can be done for any Henneberg construction.

The last thing I wanted to tell you about pseudo-triangulations is why they work for the carpenter's rule theorem. Why do they give expansive motions. Well, they're rigid of course. They don't give you any motions. But what we claimed is that if you

remove a convex hull edge-- so if we take a pointed pseudo-triangulation, and then we remove a convex hull edge, we of course get something that's flexible. Because we were minimally generically rigid, so if you remove an edge, you will now be generically flexible.

It turns out, you're not only generically flexible, you are actually flexible for at least a little bit of time. So pointed pseudo-triangulation minus convex hull edge. It's not only as flexible, but it flexes expansively. Meaning, if you look at all pairwise distances-- expansively-- look at all pairwise distances, they either stay the same, they're connected by a bar, or they will increase.

And this is what we wanted for carpenter's rule. You could do that motion for a little while. Then you might have to switch to a different pseudo-triangulation according to a flip, which I won't talk about. We already did in lecture. But at all times the claim is there is an expansive pseudo-triangulation motion by removing single edge.

So why does this hold? This, it turns out, you can prove using things similar to what we know from our proof of the carpenter's rule theorem that we covered. So it's kind of neat to see the connection between the two. So, why is this true? Well, we already argued that the thing is flexible without the expansiveness constraint. So we want to add the expansiveness constraint. So just like before, we're going to add all pairwise struts.

So between all pairs of vertices, let's say they're already connected by a bar, we add a strut between them. Meaning, that could only increase in length. Now we have a tensegrity. We want to argue that tensegrity is infinitesimally flexible. So, we're going to do that using the duality just like CDR, just like the proof the carpenter's rule theorem we saw. We say, OK we want to prove this thing is flexible by duality. That's equivalent to proving something about the equilibrium stresses.

Now in the case of CDR in a single chain, what we needed to say was the equilibrium stresses were all zero. And then we said, OK, if in order to prove the equilibrium stresses are all zero, we use Maxwell-Cremona theorem. That's

equivalent to saying all polyhedral lifting are flat. And then we prove that by looking at the maximum z-coordinate.

In this case, we can't argue that all the stresses are zero. It's not true. But if you recall the duality claim, I said if you have a tensegrity, the tensegrity is rigid if and only if the underlying linkage is rigid. Like when you replace struts with bars. Which is going to be true here. And there is a stress, an equilibrium stress, that is non-zero on every strut. Equilibrium stress being non-zero in every strut means that all the struts basically have to stay fixed length. They become bars and then you're done.

What I need to prove the opposite, I want to prove its flexible, I need to find a strut where there's no stress. If I find a strut where there's no stress, then basically that strut can expand and there's a motion. That is roughly true. [CHUCKLE] It's a little bit more complicated than that but I'm going to just claim suffice as to prove.

There's a strut that has zero in every equilibrium stress. Which strut am I going to choose? I am going to choose the one that comes from this convex hull edge, e. So I removed a convex hull edge as a bar. It's going to get replaced when I add all pairwise struts, by a strut.

So I've got the vertices on the corner here. One of these, I replaced with a strut. The other guys are whatever they are. This is a pseudo-triangulation. Not a very interesting triangulation in this case. I'm going to add a vertex here. Now it's a pseudo-triangulation. So this guy got replaced with a strut, for example. I claim this guy is zero in every stress.

Why is it zero in every stress? Well, I claim, we've essentially proved this, in any equilibrium stress non-zero stresses must be on or interior to convex polygons of bars. OK, first believe this claim. The only place I have non-zero stress is when I'm interior or on the boundary of a convex polygon of bars. And then, so there might be some non-zero stresses in here. These edges might be non-zero stressed. Intuitively, what's happening here, these guys are going to serve as mountains in the lifting. And there's a hole. So you go deep into the hole there.

Claim is that is the picture. So you've got some convex polygons. There's stuff in here that's-- could be bad. Could be stressed. But the stuff outside the convex polygons, whatever they are, have to be flat. Meaning they have zero stress. I'm jumping back and forth between the lifting and the stresses by Maxwell-Cremona. Those are the same thing.

Now look at edge e. Edge e is a strut. The other edges here that I've drawn, those are the bars. I haven't drawn all the other struts. I should really use a color for this guy. So this red edge-- there's lots of other red edges I'm not drawing between all pairs of vertices, but the only bars are the white edges. And those white edges-- well there's some convex polygons, here's a convex polygon, here's a convex polygon. But none of the convex polygons enclose e. Because e's on the convex hull. E's on the outside here. So it can't be stressed by this claim. And so if you believe this claim, then you believe e has zero stress because it's not interior to any convex bar polygon. Because it's a strut, not a bar.

OK. Now let's prove the claim. It follows from the argument we gave in, was it lecture 12? Previous one? Where we proved the carpenter's rule theorem. But it's not obvious that it follows. Let me explain why.

I want to look at the maximum z-coordinate. I want to look at the region of all the points in the plane that, in the polyhedral lifting, given by Maxwell-Cremona's theorem, they are at maximum z. That is some region. It could be two-dimensional. It could be one-dimensional. Can't be three-dimensional. [CHUCKLE] It's part of the plane.

So this region may have some one-dimensional parts. It may have some two-dimensional parts, like this whole part. All of this stuff might lift to maximum z. It could be-- have some non-convex parts, like this. Whatever. This is kind of generically what it might look like. I guess it could have cycles with not-filled interior. All these things are plausible, except we had this lemma. This is the key lemma, the heart of the proof of the carpenter's rule theorem.

If you look at m and you look at a vertex, v, of the boundary of m. I'll use this Dell notation to mean-- ignore the points in here. I want points along the boundary here. And in particular I want to look at these vertices in this drawing. Look at such a vertex, v, and suppose that v is pointed. We didn't use this terminology because we didn't have it at the time. But maybe v looks like this. And there is an angle here, this reflex. That's a point of vertex.

Now I'm looking here just at the bars, ignoring the struts. So suppose vertex v has a reflex angle among just the bars. Then the claim is locally, this entire region must be in m. Must be locally at the maximum z-coordinate. How did we prove that? Well. the claim is this had to be flat because there's no mountains here. So if you think of a reflex region and v is at the maximum z-coordinate, right? It's on the boundary of the maximum z-coordinate so it has maximum possible z. So there's no way you can go up.

These edges could be mountains. But all the other edges which are not drawn here, these blue-- there are some other edges. Those are the struts coming to v. Those all have to be valleys. They all have to lift to valleys because stresses can only carry positive stress. Positive stress correspond to valleys. These guys are all valleys and this is at maximum z. You really can't use those valleys, right? [CHUCKLE] You're up here at maximum z.

If you use valleys, you go to higher z, which is not allowed. So these actually all have to be flat out here. Which means locally, all this stuff is at maximum z That was the proof we had a couple lectures ago. Now, once you know that that's at maximum z, you know that a lot of these things are impossible. Because look, here's a reflex angle among bars. There might be more bars here, but we know that this is pointed. Whatever's missing here, there's some reflex angle. We're assuming we have a pointed pseudo-triangulation. And then we remove an edge. It's still pointed.

So that means all of this has to be in m. Is it in this picture? No, contradiction. So you cannot have one-dimensional parts because wherever you have a one-

dimensional part, there's a reflex angle that's not in m. So that's bad. In fact, they all have to be two-dimensional parts like this. Can it be a two-dimensional part like this? No, because here's a reflex angle that is not in m. Contradiction. So the only situation you can have is a convex polygon with the interior all in m. Sorry , wrong. The reverse. The exterior should all be in m.

Because, you have all these reflex angles. That has to contain the pointed part. And locally, by this lemma, it has to be in m. So all this stuff has to locally be in m. The only way is for all of that to be in m. Inside, we don't know. Could be stresses there, and this could be a hole where you go deeper. All of this is at the maximum z-coordinate. The boundary, here. The inside, who knows. Could go down.

So you can have stresses interior to convex polygons in bars, but not exterior to the convex polygon. M is either everything, and there's no boundary, and then there's no stress. Or it's not everything, and then it will have to have stresses only interior to convex polygons. And that's what we're claiming, is that stresses can only be interior to convex bar polygons. And in particular then, e is not stressed. The end.

This is definitely harder than the case we talked about for carpenter's rule theorem. When you have a single chain, you're just trying to prove there's an expansive motion. It's a lot easier than this because m is much simpler. It could basically only be a path or cycle. You only have a single chain. Here m could be more complicated, but in the end, it's actually pretty darn simple.

Any questions about that? Cool. Perfectly clear? [CHUCKLE] Definitely a bit complicated, but nice.

Now, it turns out pseudo-triangulations are really at the core of expansive motions. I'll just mention one more theorem about them. So you may recall I mentioned, if you look at the space of infinitesimal motions of a linkage or a tensegrity, they form a cone. It lives in some high-dimensional space, but I'll try to draw a cone here. These guys go off to infinity. I'm drawing it in three dimensions, I guess. You have some place here-- this is the zero zero zero motion we're nothing moves.

And these are potential different motion-- each of these points corresponds to an assignment of velocity vectors where you move and you preserve all the constraints. If I take such a motion, I can scale it up. It's still a motion. Motions form a cone. It's called a convex cone. Every motion can be scaled up or down all the way to zero. So if we look at the cone of expansive motions, it's also a cone. It's a cone of motions of a particular tensegrity where we add all pairwise struts.

Then, the pseudo-triangulations correspond to these edges of the cone. These edges, they're really rays. They go off to infinity. These things are called extreme rays. Meaning, they're extreme in a particular direction. Like if I want the motion that is the most *this* way, then it will be this ray.

They're kind of the corners of this polyhedron. The edges of the polyhedron, if you will. Although in higher dimensions, it is edges but they're called extreme rays in the case of a cone. The extreme rays in the cone of expansive motions equal pointed pseudo-triangulations minus one edge.

I think I have a slide about it. Yes.

So this is a paper by [? Roto ?] [? Santos ?] [? Enstrenu ?]. This is just a particular example for five points. These are all the different pseudo-triangulations minus an edge. In some cases, like here, you get two triangles. So that becomes rigid. So they fill in the whole clique to indicate that's a rigid component. But the claim is for five points. These are the edges of the expansive cone. Each of these guys has an expansive motion. Like this guy would rotate. Pretty simple. This one would open up a little. Like that. In general, all these guys have an expansive motion. Those are characterizing the cone. Those are kind of the extreme rays in this fairly high-dimensional cone. It has something like 10 dimensions if it's 5 times 2.

And you can prove that if you're interested. Read the paper. So, in fact, pseudo-triangulations are really core to expansive motions. If you wrote down the linear program, which is the expansive motions, then you are essentially characterizing this as the polyhedron in the linear program. If you know linear programming. And

you follow the simplex method. Simplex method is all about finding where the edges are. And here they are.

So you would actually, if you implemented the linear program, ran it on a simplex solver, you would get pointed pseudo-triangulations automatically. They would just pop out. That's actually originally how they were discovered. But then this is why they were discovered that way. End of pseudo-triangulations. All right.

Next question is have any of the open problems been solved? [CHUCKLE] I get this almost every lecture and usually the answer is, no. So I don't tell you anything. But this time, there is a nice result which is related to this picture. These are slides from the lecture. We had this linear locked tree, which is minimal among all linear locked trees. Minimum number of edges. We have this equilateral locked tree. All the edges were the same length. Was not strongly locked. There was some positive distances here, but it is locked.

I didn't mention this question, but an obvious open question is, can you get both at the same time? Is there a linear equilateral locked tree? If there were, it'd probably be a lot easier to prove locked that this mess. So it'd be nice to get both but, in fact, you cannot get both. There's this paper by a bunch of people in the open problem session from two years ago. It finally got published last year. And it proves a bunch of things, but one of the things it proves that there's no equilateral locked tree that is also linear, also lives in a single dimension.

And the way it proves that, essentially, so you take any linear equilateral tree. What does it look like? Well, it lives along a line here. It's segmented into equal-length chunks. So from a high level, it looks like this. Now in reality, there could be lots of edges along each of these segments. And there's interesting things happening inside one of these vertices. Could be something like this. That's one way to do a tree-like structure in there. It could have parts like this that go from side to side without touching these guys. Who knows what's happening here, but from a high-level perspective, it's a bunch of segments in a path.

The idea is this is basically just looking at five of the vertices out of all n of them. But

if we just look at those five vertices, everything looks good. This is in canonical form. Canonical form for a tree is where all the edges point to the right. Something like this is nice and canonical. So right now this is canonical. What we do is look inside the path here. Look for a break point where things are not canonical. And then we fix it.

So here's just an example of that. Here, this is what it looks like currently at a high level. Here's what it looks like in reality. So imagine-- so the tree is doing this stuff locally at all these vertices. Right now, we're coalescing this all into one big mega-vertex. And then it looks like this. But suppose now we realize, oh actually there's kind of a split here. There's stuff over to the right of this. There's stuff over to right of this. They're not directly connected.

Then what we'll do is pull it apart. Treat these as two separate vertices. Increase the number of vertices in this picture by one. So we end up splitting w here, according to whatever is actually happening in the linkage. And then we say, oh gosh, this is not really canonical, right? This edge is pointing to the left. These guys are pointing to the right, still. But this one's pointing to the left. So it looks like this picture got a rightward edge and then a leftward edge. We fix it by just rotating this edge 180 degrees, keeping at all times this thing pointing to the right.

So this comes along for the ride. And then we will end up with this picture. Now we're canonical again with one more vertex. We'll repeat. Eventually we'll be canonical with all the vertices. And we're done. So if you take any two configurations, you canonicalize both of them. You will end up with essentially the same picture. It doesn't matter which route you choose. But it's already known that it doesn't matter which route you choose. You can change the route just by flopping the tree with one motion or a very small number of motions.

So if you have two configurations, you canonicalize both of them then. Then there's a motion between them and you get a motion from one configuration to the other. Therefore, there are no locked trees that are equilateral and linear. Kind of nice.

The last thing I want to talk about is 4D. A couple people asked about 4D. Why is it

so different from 3D? The essential reason is we have one-dimensional bars, but four dimensions of motion. That's a gap of three dimensions. Three dimensions is a lot. When you have one-dimensional chains in 3D, you have a gap of two dimensions. Two dimensions turns out to not be a lot.

[CHUCKLE] So why is that? Let me prove it to you. Here is actually an animation of the motion, just for fun. This is in the textbook. The top row is if you are zoomed out from the very beginning. It's this mess and you basically pull out a string. It actually wiggles around a lot. Here's what it looks like zoomed in. This is the original mess. And you end up flopping this over here, then flopping it over there. And it's hard to see because this is a two-dimensional projection of a four-dimensional thing. But eventually you pull open the whole thing.

So how does this algorithm work? It's actually very simple. For open chains, it's very simple. Closed chains, a little more complicated. Trees, also very simple. So unfolding 4D open chains. This is by Roxanna Cocan and Joseph O'Rourke. Back before the carpenter's rule theorem. This publication was later but the original version was in 1998, I think, or '99.

So what do we do? Well, what I do is look at the end bar. So let's say this is the end of the chain. From here it goes in some direction, and stuff. What I'd like to do is rotate this bar so that it extends *this* bar. So I have an 180-degree angle here. That's my goal. If I can do that, I can fuse that vertex, never rotate it again. Treat this as a longer edge. Therefore the remaining thing has n minus 1 edges if I had n edges originally. So I can just apply induction. In other words, just repeat that. Eventually the whole thing will be straight.

So the challenge is, can I make a move-- can I fold this last bar to extend the next-- the second bar? Two issues. Two problems that could happen here. One is that there's no continuous motion to get there. The other problem is that even instantaneously, if I just picked it up and dropped it into the right location, it might intersect the rest of the chain. Maybe the chain here comes and intersects that ray. If it intersects that ray, I can just jiggle the whole chain a little bit, and it won't.

[CHUCKLE] Is that clear?

Let's see, you've got these one-dimensional bars, and this is a one-dimensional target ray. So if a bar happens to be on this ray, just wiggle it a tiny bit. It's going to come off of that place. This will actually even work in three dimensions. This step. There's an even easier way to do it, is just to rotate this bar. If you rotate this bar, you can basically point the ray wherever you want. And if you look at where that bar is pointing, it's kind of hard to draw the picture but you just have to miss all of these one-dimensional things. So it's very easy to miss.

So let's just assume for now-- we'll kind of see this again in a moment. Assume that this ray happens to be clear. There maybe things that come very close to it. Maybe they go on the backside and then come on the front. But they never-- don't actually touch the ray. Now think about where this bar can go relative to this vertex. If I draw a little sphere here, I'm going to draw it initially in three dimensions but really it's in four dimensions. So I have a little sphere. Where this bar is currently, corresponds to a single point on the sphere. I'm currently here. Where I want to be corresponds to another point on the sphere. My goal is to find a path on the sphere that gets to the x. Gets to the buried treasure.

This is essentially treating this as a whole ray. So I don't even care that it's short. If I just imagine it going off to infinity, can I go from position a position b? Now in three dimensions, this is a two-dimensional sphere. And if you look at the obstacles, the things I have to avoid, those correspond to one-dimensional bars here. Which, if you project them onto the sphere, correspond to great circular arcs. So the worry would be-- let me draw them in red, the obstacles. The things that will kill you if you hit them. The worry would be you have what we call a cage, something like this, of one-dimensional obstacles on the sphere that block this x-point. Block the target from the source.

And in 3D this-- if you draw what happens in the knitting needles, this is what happens. You can't move the last link to extend the previous one because one-dimensional barriers are a problem on a two-dimensional sphere because 2 minus

1 equals 1. But on a three-dimensional sphere, if you draw one-dimensional obstacles, they have basically no effect on the sphere. They do not disconnect the surface of a three-dimensional sphere.

Can you see that? [CHUCKLE] The analogy-- if you just keep the difference in dimensions the same-- so we have a three-dimensional sphere versus a one-dimensional obstacle. Let's go down a dimension. You have a two-dimensional sphere and a zero-dimensional obstacle. What's a zero-dimensional obstacle? A point. So imagine you have all these red points which you can't touch. And then you have the initial position and the target position, b. How do you get there? No problem. There's tons of paths.

It's basically the same in four dimensions, just harder to see. You have a three-sphere, which is the boundary of a four-ball. And you have these one-dimensional arcs on it. And they just don't block your way at all because they're so low-dimensional. So that's intuitively why 4D is easy, 5D is also easy, 6D, and so on. Because the obstacles are so tiny. The obstacles remain one-dimensional on this high-dimensional sphere. You can just fix one link at a time. Eventually the whole thing will be straight. Ta-da, 4D!

Any final questions? Yes?

AUDIENCE: Is the situation of folding a surface in four dimensions roughly analogous with folding [INAUDIBLE]?

PROFESSOR: Folding a two-dimensional surface in four dimensions? Definitely, this argument breaks down. And so you can get locked things, there. If you'd asked earlier I could show one example we have of a kind of locked surface. I don't know if there's actually a locked 2D surface in 4D known, though. I suspect there is one. That could be an interesting question to work on. Think about that. All right, see you Thursday.