PROFESSOR: All right, let's get started. So today we're continuing the theme of locked linkages. Last time we talked about the Carpenter's Rule Theorem, which brought together all of the rigidity theory and tensegrity theory we had built, essentially, and showed that there were no locked 2D chains, paths or cycles, as graphs.

And in general you can think in 2D, you can think in 3D, and you can think in 4D and higher. I drew this table way at the beginning of class. And you can think about chains, which are-- these are so-called open chains, like robotic arms. Paths is a graph. And polygons are closed chains.

And then another case it's interesting to think about, and there's been a lot of work on, are trees where you don't have any cycles in your graph, but you could have branching points with degree more than 2. You may recall Carpenter's Rule Theorem worked whenever you had a graph of maximum degree 2. So it allowed even multiple chains in one graph that could be disconnected. But it forbade any kind of tree branching stuff.

So Carpenter's Rule Theorem is that there's no locked chains in 2D. So we did that. But trees, there are locked trees in 2D. 3D there are locked chains. And these are two results that we will talk about in today's lecture.

And that implies that there are locked trees, sort of obvious. Trees are more general than open chains. And because there's a locked open chain in 3D, there's also a locked tree.

In 4D, though, it switches. So none of these things are locked. Of course, there are locked graphs in general. But if you look at trees, or paths, or cycles, none of them can lock. We won't have time to talk about that today, maybe another class.

Essentially we're always thinking about an intrinsically one dimensional structure. These edges, bars, are always one dimensional. In 2D, where the space are living in, the ambient dimension is just one more than the dimension of your edges, it's

really interesting. And that's what we saw for chains and what we'll see for trees.

In 3D it gets tricky. This mismatch of two in dimension allows you to lock things, kind of like knots. And in 4D there's such-- there's so much freedom in the way you move, somehow, nothing can lock, in the same way that there are no knots in 4D. Cool.

So that's the reminder summary about locked linkages, what they are, and what's known about them. Now we've seen that chains aren't locked. But before I get trees I want to talk about algorithms for actually doing this. This is, after all, geometric folding algorithms.

We've seen a proof that they exist, that in fact if we had a chain, there's an expansive motion increases all the distances and therefore doesn't collide with itself, and eventually straightens all the outermost open chains and convexifies all the closed chains that are outermost, not containing anything.

But how do we actually find such a motion? And this has been the subject of some study. There are three algorithms for doing it.

So we'll call this unfolding 2D chains, unfolding being the goal of straightening or convexifying. And the first algorithm is just to mimic the proof that we saw.

I call that proof CDR, because it's Connelly-Demain-Rote. And if you think about what we proved, we just looked at the sort of infinitesimal case. We said, if we're at some position then at least, for an infinitesimal amount of time, we can increase all the distances. All the struts can increase in length, and the bars stay the same length.

That's really only specifying the first derivative of a motion. So if you think of motion as being, I don't know, some configuration C that's a function of time, and you take the derivative with respect to t, then we computed what that derivative should be. That was the first order motion. I think we called it d, which is a little confusing. This is the d that we computed.

And we said, well that's an infinitesimal motion of some tensegrity. We can find it with linear programming. So at any moment in time we can figure out what our derivative should be. This is what's called an ordinary differential equation, for those who have taking 1803 or whatever. You just solve it.

And a very easy way to solve it is if you imagine configuration space, you start somewhere, you compute which way to go, you move a little bit in that direction. You're only supposed to go for an infinitesimal amount. But it goes some positive epsilon.

Then recompute your new direction, follow that, recompute your new direction, follow that for a little bit, and keep going. And you're almost tracking the intended curve. It's not quite perfect.

But as your step size, this epsilon, goes to 0 you approximate the intended path. And that's, I mean how close you approximate is computed in the textbook. You can see that for details. This is a method called forward Euler. It's like the simplest way to solve an ordinary differential equation. There are much better ways. But it's easiest to think about, and reason about.

And so it gives you some approximate solution. Let me show it to you.

I'm going to show you three methods. The top one is the one I'm talking about. So we're starting with the same polygon on the left, simple little v shape. A second method is called pseudo-triangulation. We'll talk about that next. The third method's called energy. That one's one you've seen.

Let's start with the first method, where we take some example like this teeth example, and we, at each step, compute what is the, in some sense, the biggest expanse of motion you could do. We go a little bit in that direction.

So in these animations, the edge links are changing a little bit. And I've chosen the step size so the edge lengths change by, at most, 10%, something like that. And here, this one, left side we're zooming, right side we're not zooming.

These are somewhat older animations. So apologies for the frame rate. What else do I want to say?

This is an example-- well, notice it's pulling the teeth off sort of from the end. If you want to be expensive you cannot pull the teeth apart. We're going to see other ways to do the teeth in a moment.

Here's another example. Both of these examples at some point we're conjectured to be locked. Before the Carpenter's Rule Theorem was proved, a lot of people were trying to find counter examples. There aren't any.

So this is nice. It preserve five-fold rotational symmetry. And you can do it for multiple shapes as well. So here we have one polygon in the center, four arcs, four paths on the outside. In this case, the simulation stopped when one of them convexified because I didn't implement all of the algorithm. At that point you just have to rigidify that polygon and expand the rest.

So that gives you an idea sort of visually what this looks like. Qualitatively, we can say a bunch of cool things about this method. One is that's it's the only method that's strictly expansive, strictly meaning not only do the struts not decrease in length, but they also don't stay the same whenever that's possible. They strictly increase.

Now, there are some exceptions. If I have a bar, obviously, if I add a strut here it's not going to strictly increase in length. Also, if I have a bunch of bars that are collinear and I add a strut from here to here, that's not going to increase in length. It's not possible.

But all other struts will strictly increase in length. You can check. That's what we proved here. So that's nice.

Another fun fact, I didn't write it here, but it preserves symmetry if you do it right. You may notice in this example actually, it was originally four-fold rotationally symmetric. It does rotate a little bit. That's from computational error. And you could correct for that. I didn't correct for it.

It should, in theory, preserve the four-fold symmetry there. Good. We can compute one step. So how fast can you compute this thing?

We can compute one step in polynomial time. That's finding an infinitesimal motion of a tensegrity. That's linear programming.

But how many steps does it make? We don't really know. There is a bound in the textbook. It's not very small.

It's also-- with forward Euler you lose a bit of accuracy. Now you could correct for accuracy, force all the edge lengths to be the same, then you won't preserve the symmetry. There's a bit of trade off here. I don't want to spend too much time on it.

This is the first algorithm. Although it's the only one that's strictly expensive, to me it's not the nicest. So I'm going to move on to the next method which is pointed pseudo-triangulations.

This is a method by Ileana Streinu, basically at the same time as the original Carpenter's Rule Theorem. And these are examples of what are called pointed pseudo-triangulations for a given set of points.

So in general, a pseudo-triangle is a polygon that has three convex vertices. So a triangle has three convex vertices and that's it. A pseudo-triangle, I allow a bunch of reflex vertices in between those guys. So this should be a polygon. I've drawn it curved.

And if you check in this figure, all of the faces are pseudo-triangles. Most of them here are actually quadrilaterals. But in particular, that's a pseudo-triangle. Sometimes there's actual triangles. Here they're all quads or triangles.

And a pointed pseudo-triangulation has the property that at every vertex all of the edges incident to it lie in a half plane. They all lie on that side. In other words, there's an angle that's bigger than 180 degrees. So this was a pseudo-triangle, and this is pointed. And if you check this example, every vertex has an angle that's

bigger than 180 degrees.

Obviously the ones on the outside have to. They're always convex. And pointed pseudo-triangulations are-- they're not only supposed to have these two properties, but you're also supposed to have as many edges as possible, subject to having these two properties. If you tried to add any edge to either of these examples, you would violate the pointed property.

If I added this edge, then I'd destroy this 180 degree angle. Any edge you consider, either you get a crossing-- which is not allowed-- or one of the vertices loses a 180 degree angle.

So this is actually a really powerful concept. And it was introduced by this paper by Streinu in 2000. It has lots of theory built around it by now. I'm not going to talk about that theory. I want to talk about how it applies to Carpenter's Rule problem.

I guess the first thing to say is that if you have, say, a polygon-- maybe I should do a real example, so I'll make it not so giant. OK, you take a polygon. So far it's pointed, right? Every vertex-- one of the two sides has an angle bigger than 180.

So just add as many edges as you can while still being pointed. Do, do, do. This one, that looks OK.

I think that's it. I think this is a pseudo-triangulation. I've added as many edges as I can while still being point. It turns out if you start with something that's pointed, you'll be able to keep adding edges, keep it pointed. And in the end all of the faces will be pseudo-triangles. So mission accomplished.

This is a pointed pseudo-triangulation that contains, as a subgraph, my original polygon. So these are the edge lengths I want to preserve. Those should stay rigid. I'm going to be kind of crazy and also preserve the lengths of all of these edges.

Now if you did that, this thing would be rigid, which is not so exciting. All you need to do to fix that is remove one of these added edges that's on the convex hull on the outer boundary here. Then it will have one degree of freedom.

So there will be two ways to move it. One is to expand this distance, the other is to contract it. And the one that expands that distance will expand all distances. So if you look at some other pair, like this guy, every other pair of distances will expand or stay the same.

So this thing is expansive. But not strictly expensive, because we're preserving these lengths. These ones will stay the same. So let me show you some of the things that can happen.

So this is sort of at the heart of the algorithm. I'm not going to go into all the details. This can happen like n cubed times, where you say, OK I've added in all these edges. Here's a nice pseudo-triangle here. It happens to be a triangle with some stuff on the bottom, stuff on the top.

As I flex this to expand the distances this is going to move like that. At some point, those three points will become collinear, got to deal with that. And the way you deal with it is as you continue moving, it's no longer the case that those two blue edges are the edges in the pseudo-triangulation. Now it's this one red edge.

This is called a flip. And you see, we used to have a quadrilateral up here an a triangle down here. Now we have a triangle up here and a quadrilateral over here. That's pretty intuitive. You preserve that it's a pseudo-triangulation as you move this.

If you tried to go too far here you would actually get a convex quadrilateral. If this kept going then this would be a convex quadrilateral. That's not a pseudo-triangulation, that's bad.

But you can always do these flips to fix it as you go along. And it turns out this will only happen a polynomial number of times.

AUDIENCE: [INAUDIBLE].

PROFESSOR: Like shown, is how I flipped it.

AUDIENCE: Is it [INAUDIBLE] to a red?

PROFESSOR: Yeah, the red edge now goes from here to there. Whereas before there were two segments that went from here to there, now I'm going to go straight from there to there. That's the difference.

This is the general picture, you have to believe me, because I want to talk about other things. And, cool.

So here's an example of it actually running. So we have some polygon in the top left. We've deleted the edge that's cyan. So that one's going to expand. And we move a little bit. This is moving a little bit. This is moving a little bit more.

At this point, these three points are collinear. Actually, the way it's drawn it looks like a little bit too far. But they should be collinear. That would be a violation, because this would be-- if we go any farther-- that would be a violation because this would be a convex quadrilateral. So we do a flip here, and we end up with that edge instead.

So now everything's a pseudo-triangle, everything's pointed, everything's happy. You keep going, then these three points become collinear. So you end up getting that edge. And you keep going.

And all throughout this motion all pairwise distances are not decreasing. Everything's expanding or staying the same. And then the wrap up looks like this.

In the end, you get a convex polygon. Then you're stuck, because there's no edge to remove on the outside. You've got to keep all of those edges.

So that is the pseudo-triangulation method in a nutshell. And it's expansive. It has a polynomial in n number of moves.

It's not clear how quickly you can compute the moves. I know an exponential time algorithm. There might be faster one. I don't think this has been resolved. Let me just check my notes here.

It says best algorithm is exponential to compute one move. In these examples,

where they're all quadrilaterals or triangles, it's really easy to compute them. In general, if you get complicated things, it's unclear.

Pseudo-triangulations though have a lot of nice structure. So maybe it's easier than general linkage folding. This, I have here, implementing this algorithm would be a cool project.

Oh, another open problem is how many steps does this algorithm really need? Can you prove a pseudo-polynomial bound? I'm not sure.

But let me tell you about my personal favorite algorithm, the energy method. This is also the most recent, in 2004. The weird thing about this algorithm is that it is not expansive.

It's really easy to compute one step of the algorithm. I can do it in quadratic time, even exactly. This one you have to specify some error tolerance epsilon. This one you have to specify some error tolerance epsilon. Here you can do it perfectly if you have exact square roots.

And the number of steps is also small. It has a pseudo-polynomial number of steps. Have I talked about pseudo-polynomial yet? Maybe, briefly at some point. Let me tell you what I mean.

I want to be polynomial in the number vertices n and another parameter r. r is going to be, basically, the maximum distance in your initial configuration divided by the minimum distance. Exactly how you define distance, don't worry about it. Not a big deal.

So this is a geometric feature of the input. It has nothing to do with n. n could stay fixed and you could change this to whatever you want by making nastier and nastier examples. Nastiness is measured here by how big your linkages versus how tight things get.

So as long as that's reasonable, this is going to be a polynomial number of moves. So this is a good bound. It's better than all the others that's been proved anyway. It

might hold for some of the other methods.

Here, of course, we have a better bound on the number moves. But each move is kind of complicated. It's pseudo-triangulation. So there is a trade off between complexity of move and number of moves. But this gets a decent bound for everything.

You've seen this method. I showed it in lecture one. But it's interesting to contrast it with the CDR method, which sort of unrolled the teeth from the end. Here, this method basically pulls the teeth right apart. And that's possible because it's not required to be expansive.

And we have the double tree. This one looks more or less the same. It's a little smoother, I would say, than the other method. I don't have it side by side. So it's a little hard to guess.

And what's really exciting about this method, to me, is that you can run it for really large examples in like a minute or so. 500 vertices, no big deal. Whereas, because each step is pretty easy to compute in quadratic time, with this method you need it to solve a convex program. And that's a little costly for large examples.

Cool. So that's the energy method. Now I'm going to actually tell you how it works.

Big difference is it's not expansive. And the idea is, well, being expansive on every edge, that's kind of hard. How do I figure that out? Oh, it makes my brain hurt. I've got to solve this tensegrity.

But if I was just expansive on average, maybe that would be good enough. That's the crazy idea.

So we define this energy function that does just that. It's a function on configurations. So I'm going to write E of C. No M in this equation.

We're going to sum over all edges Ew, and then sum over all vertices U different from V and W. And we're going to take 1 over the distance from U to the edge. This is just saying sum over all non-incident vertices and edges, take their distance--

there's many ways you could define that, like the minimum distance between them would be fine-- take the reciprocal, add them all up. This is my energy function.

Kind of weird. But it's averaging not of distances, but 1 over distances. So why is that so interesting? Because distances are always positive. So I only have to go over here.

If I write distance in the x-axis, and 1 over the distance in the y-axis, the plot of that function looks like this. It has this vertical asymptote at 0.

So if the distance were to go to 0, which is bad for me because that's one thing start crossing-- they will start touching at d equals 0, and after that they might cross-- the energy shoots to positive infinity. So if I take the sum over all these, if any of the distances decrease to 0 then the energy will shoot to infinity.

What if my distances increase? Because I know expansive motions exist. That's what we proved last time. I'm not going to compute them. But I know they're out there somewhere.

If I had an expansive motion, all of these distances increase. So that means this reciprocal decreases. So I have-- it's hard to imagine-- I have this giant dimensional configuration space. It has d times n dimensions, whatever. A lot of stuff.

I'm somewhere. But if I now plotted-- imagine that's in two dimensions, in the plane here, so that's there-- and then in the third dimension I plot what is this energy landscape. For each of these configurations I compute some height. What is my energy?

I get some 3D surface here. In general, it's going to be d times n plus 1 dimensions. Here's the configuration space. We plot over that.

What I'm saying is, because expansive motions exist, there has to be a motion for every point that decreases energy. Because if you decrease every distance, you also decrease them on average in this sense. If every term decreases, then of course, the sum will. So this means energy decreasing motions exist.

Now energy decreasing feels like a good thing. Because I start somewhere, it has some energy, not infinity. If I decrease the energy, it's really hard for my energy to go to plus infinity. It can't happen. So all I need to do is follow any energy decreasing motion.

It might be an expansive one, but probably not. We know that there are energy decreasing motions because there are expansive motions. But let's just take any energy decreasing motion, sort of expansive on average. Then it won't self-intersect because if energy decreases it will never get to plus infinity. And therefore none of the distances will go to 0.

So what this algorithm does is follow the gradient. So I should say something about what this notation is. This is whatever, higher order calculus.

You live in some space. You're on like some hill, or whatever. You're on some surface. And you say well I'd really like to go downhill from here.

And there might be many downhill options. There might be many uphill options. This is some crazy high dimensional choice.

Just take the option that decreases energy the fastest, the most downhill. That is negative gradient of E. Believe me that it exists. It exists because this energy function is smooth, in some sense.

You're basically taking first derivatives. That gives you the highest chain-- or you take the place that has the highest change and boom. That gives it to you.

It's very easy to compute because this function has quadratically many terms. It takes about n squared time to find it. And that gives you some energy decrease in motion. It's just an easy one to find. And that's what we're animating all the time.

So we just find energy decreasing motion, move a little bit in that direction. Not too much, because if you go-- it's again a first derivative. This is only an infinitesimal motion.

But we're actually going to move in that direction for a positive amount of time. As long as we don't go too far, and we can find how far by binary search, we won't self-intersect. Because we know locally it's decreasing energy. If we go small enough step it really will decrease energy, and then life is good.

So it's really easy to do this. Super simple algorithm. Good. It's non-expansive because we're only decreasing the average, not each of the terms. Do, do, do, do, do.

You can prove the number of steps-- this is the part I'm most proud of here-- is pseudo-polynomial, polynomial in n and r. What is the polynomial? n to the 123 times r to the 81. This is the largest polynomial bound I know of that's not dependent on dimension. So I'm very proud.

In practice, the number of steps is much, much smaller than this. This is what we could prove in an easy way. If you want a project, it's a little tedious, but it would be easy, I'm pretty sure. You could decrease this to, I don't know, at least n to the 20 or something by being a little more careful on the analysis.

But once we got something that was pseudo-polynomial we were happy. So we could leave others to figure out the right bound. It's a little tricky with these gradient descent algorithms to get good bounds. In practice they work really well, because the gradient is not as nasty as you might imagine it to be in the worst case.

But the worst case bound, hey, it's pseudo-polynomial. It's a nice theoretical guarantee. And in practice it also happens to work. Like these examples only take, I don't know, a few hundred steps, 1,000 steps, whatever.

AUDIENCE: Where do those numbers come from?

PROFESSOR: Where does 123 come from? We really wanted it to be 1, 2, 3. 81 is my birth year.

[LAUGHTER]

PROFESSOR: That's why I like this number so much. Oh, actually here I have 41 written. I'm pretty

sure it's 81 though. I should correct the notes, double check, something.

It's really you're just adding up twos and threes a lot, and fours, and things like that a whole bunch of times. And then just luckily it came out to a nice number. As far as I know, not intentional.

There are few authors though, so I don't know. Maybe one of them increased a bound to make it a little cooler number in the end. Good.

So that's pseudo-polynomial number of steps. It's interesting, each of the steps is actually a very nice motion. It just moves along a straight line in the configuration space.

I would be nice to know whether you can actually achieve a polynomial number of steps independent of r. I think the answer is no, it's not possible for some linkages that are really tight. I think you need a dependence, a polynomial dependence, on r-- at least a linear dependence on r. But I don't know how to prove that. It's a nice open problem.

Another fun problem, these examples with polygons end up with a particular convex shape in the end. Is that shape unique? If I gave you the sequence of edge lengths here, they're all 1, or they're almost 1. Not quite the same. No, sorry, they get tinier as you go to the center.

But for that sequence of edge lengths, is there a unique minimum energy configuration? I think so. But I don't know.

We know that this method will get to convex, because only at the convex configuration do you no longer have an expansive motion and no longer have a decreasing energy motion, maybe. Cool.

Those are the three algorithms. Any questions about them?

Before we go to trees, again, tell you a cool application to origami, which is to rigid origami. Remember, rigid origami, we're not allowed to add any creases. And all of the faces between creases have to stay flat.

They're like made of-- imagine you have pieces of metal representing the faces. You have piano hinges making the edges. That's rigid origami.

That's kind of like a linkage. In particular, when you have a single vertex origami-- say that's five foldable, I don't know. And if you say each of these wedges is a rigid piece of metal, well we already know this kind of set up can be modeled by its boundary. Ignore the interior. Just think of there being hinges of this one dimensional structure like that.

So that looks a lot like a linkage. These have to stay rigid. They're not straight lines. It's a little different. It's almost the same.

If you think about how you're allowed to fold these things-- really you should think about it here, I guess-- what happens, by a continuous motion what happens is that you're living on a sphere. So you start out on the equator of the sphere. This thing, just plop it down the equator. The boundary lies on the boundary of the sphere. The interior of the paper is inside the sphere, right along the flat part there.

As you fold, these points will stay on the sphere. And these edge links will be preserved, their arcs on the spheres. There will be great circular arcs at all times.

So folding this thing in three space is really equivalent to folding this linkage on the sphere. And that looks an awful lot like a polygon on a sphere. What we really want is a spherical Carpenter's Rule Theorem.

And there is. This is by Streinu and Whiteley. So if you have a closed chain, a polygon, of total length at most 2 pi on a unit sphere then you have a connected configuration space. So in the plane, closed chain always had a connected configuration space. On the sphere you need that the chain is not too long, because there's only sort of a bounded amount of room on the sphere.

The equator has total length 2 pi, perimeter of the equator. For unit sphere it's 2 pi. So we're just canonically making it a unit sphere. And 2 pi really corresponds to this situation, which is 360 degrees. That's how much total length we have. So it just fits

in the equator, life is good.

The reason you want it to have length at most 2 pi is because then you actually do have a convex configuration. In the case of 2 pi, it lies along the equator. If it's smaller than 2 pi it'll be like some smaller portion. This would be less than 2 pi on the sphere.

If it's greater than 2 pi, you can't draw it convexly on the sphere. You can draw something. Like I could draw something like this, that'll have really long length on the sphere. But there will be no way to convexify it. You get stuck at some point.

And as long as you don't have that situation, you can take this-- what this actually implies is that your polygon at all times will lie in a hemisphere. You take that hemisphere and you sort of unroll it, you splay it out, you project it to the plane. You apply the planar Carpenter's Rule Theorem.

You apply the fact that unrolling operation, or that projection if you will-- I think it's like projection from some point here out to the plane-- that projection preserves infinitesimal flexibility of tensegrities. So it'll still have the expensive motion in the plane. And then you could turn in to an expansive motion on the sphere. And eventually you'll get a convex polygon.

So that's sort of how this is proved. Then you can apply it to rigid origami, and say for single vertex origami, it's always rigidly foldable. Any state you want to reach can be reached by continuous motion, without bending any of the faces. So a fun little application.

And one of the few things we know about rigid origami, for multiple vertices it gets a lot harder. Not always possible, and we don't have a nice characterization.

So finally, let's move onto locked trees in the plane. So these are sort of the classic examples. The top two were in a 1998 paper. This was when I started working on folding stuff, very beginning. A whole bunch of people from a big workshop.

First example is this one. And it's a bunch of sort of arms tucked into their armpits,

and in a cyclic way. And the dotted circles mean objects in this mirror are closer than they appear.

So imagine that all of these points are actually really, really close and tight in here. And this guy's actually really close and tight against this edge. So I've drawn it with lots of slack so you can see the combinatorial structure. But geometrically it's much tighter.

Then the intuition is that you can't get any of these arms open unless you could somehow expand one of these wedges. It's like, if you could expand this angle then this guy would have room to come out.

But how do I expand that angle? Well I'd have to compress the other angles because of the cyclic picture. And I can't, if I look at some other angle like this one, I can't compress it because the arm is in the way.

So I can't open an arm until I've closed some other arm. And I can't close an arm before I've opened it. And so nothing can happen.

That's the intuition behind the proof. The details are very messy, because you have to define open and closed, and what things must happen before what things.

This example should look familiar because if you double it, if you replace every edge with two edges, we get one of the examples that was expanding with the five-fold symmetry. So people have sort of known for a while, and then we finally proved, that this is locked. If you double it, people thought well maybe it's still locked.

Turns out no, because the center vertex can expand in a polygon, but with a tree it can't. OK, big deal.

These examples, this is just yet another way to do that. But what's interesting is if you double some of the edges you get this tree. So you have 1 degree 3 vertex in the center. And you go around, you visit this arm. You go back, you visit this arm.

Turns out this really does simulate this tree, because we haven't touched the central degree 3 vertex. And this is nice because it has one degree 3 vertex. Everything

else is degree 2 or 1.

So in the Carpenter's Rule theorem where we said maximum degree 2, it's really tight. And as soon as you add one vertex of degree 3 you can be locked. That was the point of this example.

What other fun things can you do with trees? Well, three years ago in this class we started thinking about other locked trees. How low could you go? How many edges do need to get locked?

This example-- no I guess this example would be minimum. Here I've drawn it with eight petals. You can get away with five petals, or are five arms, each of length 3. So that's 15 edges. That was the state of the art.

And then we had this crazy idea in a problem session of this locked chain, locked tree. It has two degree 3 vertices. And it kind of just winds in there. We're going to see why these things are locked.

But OK, that's kind of neat. 11, that's much better than 15. Can we do better? And every week we improved it, for a few weeks.

This one looks messier, but it has one fewer edge. And then this one came along, and we're like whoa. So symmetric, so beautiful. What's interesting is it doesn't have the cyclic structure. It's almost flat, in fact.

You could think of all these guys being in one point. All these guys being in one point. All those guys being in one point. It's like they're, all three, collinear. So it's a very different kind of example.

Before we thought locked trees required this kind of cyclic condition. And so, for example, we conjectured-- or I guess Poon, one of the authors here-- conjectured that there would be no way to lock something if all the edges were horizontal and vertical. Because if you have that you couldn't have five things in a circle.

Now suddenly we think, oh, this is interesting. Because it has two degree 3 vertices.

It's so symmetric. Surely this is the fewest possible edges we could get away with.

But no, then we found the eight edge example. And this is kind of funny. It's like instead of being nice and symmetric, essentially, what we're doing is removing this edge. And we want to instead attach it in here. But then we have to futz around with the vertices to make it work out and be a tree.

And now has only one degree 3 vertex. So it also has that nice property. It only has eight edges. And this, we believe, is optimal.

And we can actually prove something along those lines. So a linear tree is one where all the vertices lie nearly on the line. So this is a linear locked tree. And we can prove that among all linear locked trees, they must have at least eight edges.

Locked linear tree has at least eight edges. So at least among linear locked trees that example is optimal. But maybe you could use the second dimension to do something better than eight edges. But I don't think so. That's an open problem.

What other good things can we do? You can make it orthogonal. You can mimic exactly this structure with an orthogonal structure, all the edges horizontal and vertical. Just expand each of these vertices into very tiny, and if this is drawn really squished, these are super short edges. So they really don't change the motion space hardly at all. You can actually prove that.

And this is also locked. That's one of the nice things you do once you get out of the cyclic kind of structure.

I think I have something else, no. OK, stay there. Yeah, actually maybe I do want to go there.

This is an example, it has a cyclic structure again. Same paper, last year. This is a newer example. And it has the property that all of the edge lengths are the same, and nothing touches.

Now you'll appreciate this in a little while, how crazy this is. Because all of the previous examples required things to be very close and tight. And we crucially use

very tight proximity in order to prove things are locked.

And if you take the example that looks like this, with six arms-- I didn't draw it very well. But if you draw it with six arms, these are like equilateral triangles, the edge lengths will be almost all the same. In fact, if you allow them to touch they will be exactly the same.

So there was this open question, well, if I want them to be exactly the same but not touch, can you still lock? We thought no, but in fact you can do it. And this is very tricky to prove locked because we can't make these arbitrarily tight. It really has to look like this, because the edge lengths are all the same.

Now this has seven-fold symmetry. Because sixfold they would all touch. It's tricky. Cool.

A big open question here, of course, is to characterize locked trees. I think that's quite hard, because in particular, if you're given a tree and you want to know does that go from this configuration to this configuration, that's known to be p space complete, which is really, really hard. But all of those examples are locked. So maybe unlocked trees have some special structure that's easy to find.

I guess not. But who knows? What I do think has a nice structure are these linear locked trees. They're pretty simple. They're basically one dimensional.

And I think we could characterize linear lock trees in polynomial time. Maybe we'll work on that this afternoon. But that is open.

All right, next thing I want to talk about is how the heck do you prove that these things are locked?

Now historically there have been lots of different proofs. And this tree still does not have a nice proof that it's locked. But all the other trees I can give you very succinct proofs that they're locked. And so I want to tell you how we do that. Because it uses tensegrity theory, our good friend.

This is the idea of infinitesimally locked linkages. So ignore this part of the picture for now. If we take some tree, most of the examples I drew these little scion circles to say, well, these guys are really tight. And the intuition is the tighter you make it, the less that configuration can move.

If you look, we're claiming the configuration space is disconnected. But not only that, we have this configuration and we say there's a small ball of motions that you could possibly do. It does wiggle a little bit. And then there's other stuff over here that you can never reach because this can't move very much.

Well in fact, if you make these circles tighter and tighter, the claim is that this, in the space of motions, you get less and less freedom. How could we formalize that? As you draw the thing tighter you get less and less motion.

Well, let's go to the limit. Suppose we went all the way to the point that these things are touching. Now this is going to be a little tricky mathematically, because we have to remember that this vertex is in this wedge, even though it's actually right on top of this point. So we have to remember, sort of, how things look locally. But geometrically the picture is going to look like that.

If you look from afar, you won't be able to tell that there's three edges along here, because they're right on top of each other. But if you, sort of, imagine zooming in infinitesimally in each of these vertices, it's going to be whatever the heck it is.

So you remember the fact that right here there are four vertices. And this is how they're connected to incident edges. There's actually three edges here, and two edges here, and so on. So that's how you describe one of these, we call them, self-touching linkages. Because the edges are touching each other. They're right on top of each other.

Now normally, if you wanted to capture this notion of wiggling a little bit, we could define the idea of being locked within epsilon. So a configuration is locked within epsilon if it's impossible to get farther than epsilon in configuration space. So that's this little ball, thinking about radius epsilon here.

And if you can't get outside that ball, you have some weird space you can get to. But if it's bounded by a ball of radius epsilon then I say you're locked within epsilon. And if epsilon is small that really means there are other things you can't get to.

Well as soon as I get to self-touching I can actually think about being locked within epsilon for epsilon equals 0, also known as being rigid. We've already talked about being locked within 0. If you can't move it all, that's rigid.

So this thing could actually be rigid. Whereas all the locked trees, they can never be rigid because I don't want them to really touch. This is an analysis tool. I don't like trees that are self-touching. It's kind of ugly and cheating.

And so real trees, non-self-touching touching trees could only be locked within some positive epsilon. But if we consider for the moment the extreme when they're touching then we could hope for rigidity. Rigidity is good because we know how to prove things are rigid. We know how to test things are rigid in two dimensions. It's pretty easy.

We could test is it infinitesimally rigid? If it's infinitesimally rigid we know it's rigid.

It's a little bit trickier because we have to represent the non-crossing constraints. And that's what these purple edges do. So the idea is, well I definitely want to preserve the lengths of these edges. I'm not interested in expansive motions, because that's a subset of possible motions.

But I do know that this vertex should move away from this edge, or stay on the edge. It's not allowed to go to the other side of the edge. So I imagine there's a little tiny strut here. It's of infinitesimal length right now.

It can get longer, but it has to get longer in that direction. You have to move away-- you have to stay on the right side of this edge. And you have to stay on the left side of this edge.

It turns out you really can represent that by a strut. It's now a strut between a vertex and an edge, because this guy can slide along the edge, or move away from it. But

it's a strut. And you can think of this as a tensegrity, all the usual tensegrity theory applies. You can define equilibrium stresses, polyhedral liftings, infinitesimal rigidity.

And so what's actually drawn here-- you should look at the book for more details. I don't want to go into details on this. But originally the state of the art for proving something like this is locked is you basically give-- you show an equilibrium stress that is positive on all the struts. We know that if you find a stress that's positive on all struts than all the struts in fact must act as bars.

What that means is that this length, which is currently 0, must stay 0. Therefore this vertex is actually pinned because it has to both be on this edge and be on this edge. That's what a 0 length bar here and here would mean. It really has to be right here. It can't move it all. And therefore this whole thing is rigid, and nothing moves. Clear?

So we can use all the tensegrity stuff to prove that when this thing is actually self-touching, and all these distances are 0, it is rigid. But so what? Who cares about the self-touching thing being rigid? It's nice, but what I really care about is the non-self-touching configurations are locked within epsilon, for some tiny epsilon.

Well good news. Rigidity of the self-touching configuration implies what's called strongly locked, yet another term which I need to define. Strongly locked means that sufficiently small, sufficiently small perturbations of the linkage, or I should say of the linkage configuration, are locked within epsilon for any epsilon.

This is exactly the property I wanted to formalize, saying that if you draw the example tighter it can move less. For any epsilon, we want to say you can only move within epsilon, there's some notion of sufficiently small such that if I take this example and imagine currently all the vertices are on top of each other. But now I perturb it a little bit, which involves changing not only the vertex coordinates, but also the edge lengths-- but a tiny amount, some delta that's a function of epsilon.

Whatever epsilon you choose, there's a very small disk I can draw like this, such that as long as all the vertices stay within that disk your example will be locked within epsilon. This is great because it lets us analyze rigidity, which is easy for self-

touching configurations, which are not interesting in some sense. But we get to conclude something about the perturbations which are not self-touching and therefore nice. And we get the property we want, that you're locked within epsilon for any epsilon you want. You just draw it tighter you'll be locked within a smaller epsilon.

So this is pretty cool. And I didn't defined perturbation. But I just mean every vertex stays within a radius delta disk. And sufficiently small here is delta, which is a function of epsilon.

So this is pretty cool. And it turns out also-- this is proved much later-- these results are like 2002. And then 2006 proved that if you take any self-touching configuration, which is like this, you specify the geometry where things are all on top of each other. And then you'd say what you want every vertex to look like. It turns out there really is a valid perturbation that preserves that combinatorial structure and is arbitrarily small.

So here, of course, I've drawn it so it's clear. You can perturb things. And I changed the edge lengths a little bit. But I can actually realize this combinatorial structure. It turns out that's always possible. So you can take any self-touching linkage, you can perturb it so it's not self-touching, and it is arbitrarily locked within epsilon.

And the way you prove it, or one way to prove that something is rigid, is to say well this proof is infinitesimally rigid by constructing an equilibrium stress that's positive on all these 0 length struts. As long as it's positive all those 0 length struts you know that they're effectively bars. Usually once they're bars it's really obvious that the thing is rigid because it pins vertices into corners. Then you know that the whole thing is infinitesimally rigid, therefore it's rigid, therefore it's strongly locked.

And you could see some examples of doing that in the book. But this is no longer the state of the art. There are now easier ways to prove that trees are locked-- some trees. It doesn't always work. Of course, it might be infinitesimally flexible. Lots of things could fail.

But if it succeeds in proving something is rigid, then you know it's strongly locked and you're happy. So it's a conservative test, you might say. It would be a cool thing to implement. This is not hard, it's just linear programming.

But there's a cooler way that's even more-- like one where I can really draw the pictures here, you know we had to draw all these diagrams and figure out that stress positive numbers worked on all these edges, and eh, it's tedious. There's a much slicker way.

And for whatever reason, they have become known as the rules. There are two of them. Although we've tried to come up with various rule threes, the ones that have been tried and tested and used all over the place are rule one and rule two.

So rule one. You'll see why this is interesting in a moment. I have some linkage, self-touching linkage. We're in the same framework. Suppose these two edges have the same length. I'm drawing this one slightly smaller just so I can show you, which is on which side.

But suppose they have the same length. And suppose that both of these angles are acute, strictly, less than 90. What do you think happens in this picture? I have this bar floating around. And I have these three bars. What happens to this bar?

AUDIENCE: It's confined to be against the other bar.

PROFESSOR: It's confined to be right against this edge. It can't move at all until this angle-- both of these, and I guess at least one of them would have to get to 90, then you could try to slide it out. Like if this one goes beyond 90 then you can slide out.

As long as they're both less than 90 it's pinned there. So what I'm going to do is redraw this diagram as with two edges there, which I mean, you can just ignore. The point is if there's something attached here and here-- maybe many things, it's a tree, who knows?-- you can just attach them right there.

This is a simplification to the linkage. It does not behave the same. But it has the same rigidity. Because if all I care about is rigidity, I care about can I move at all? So

in order for this guy to move and all, these guys would first have to move for quite a while, a positive amount of time. I just want to know can I move 0 or more than 0?

If I can move more than 0 I could move this guy more than 0 without this guy moving at all. So really I don't care about how this guy moves. He's effectively pinned for at least a small amount of time. I really care about can the rest move it all? So you can simplify your linkage like this, and the rigidity will be preserved.

This is awesome because it's easy to see when this applies. And you just simplify your linkage until you can just tell whether it's rigid.

All right rule two is sort of a special case. It looks like this. So here, this bar and this bar actually share an endpoint.

And again, I need that this angle is acute. And I need that these two have the same length. And I can simplify to that, where if anything was attached here it just gets attached there.

Let's try it out. Actually, I could use some more boards. So I'm just going to copy the example we had from before. And this will work on basically all the examples I've shown you except the equilateral one.

This is the one, two, three, four, five, six, seven, eight bar example. This is conjectured minimum. Let's prove that it's locked.

See any rule one's we could apply? Do you see any edges that are effectively wedged against another edge? Remember, all of these vertices are actually on top of each other, we're thinking about the self-touching version. These guys are actually touching, and these guys are actually touching.

AUDIENCE: This rule one [INAUDIBLE].

PROFESSOR: There's a rule one--

AUDIENCE: That one, there.

PROFESSOR: --here? Yeah, good. This edge is pinned against this edge. Because look, we have acute angles here, namely 0. There's an acute angle here, that's 0. And this edge, if these guys are on top of each other, and these guys aer on top of each other, these two edges have the same length. Therefore these are pinned together.

So let me redraw it when they're pinned together. I'm drawing things with curves just so it's easier to draw, but you understand this is also basically flat. Again these guys are all on top of each other, these are on top of each other, these are on top of each other.

Did I do that right? I think so, yep. Anymore?

AUDIENCE: [INAUDIBLE].

PROFESSOR: Yes, on the right side. It's another rule one. Here it's actually pretty symmetric.

Is this locked? Is this rigid? It's kind of hard to say. But now it's going to be pretty obvious.

Because when these guys join together that means these two vertices really are on top of each other for positive time. Same for these. So a new example looks like two triangles with an edge floating there.

Now it's pretty obvious this is rigid. But if you really want to make it obvious you can apply rule two to this guy. And then these guys are pinned together. And then you have two triangles. Two triangles are rigid.

I think that's pretty obvious. You could check it, whether they're infinitesimally rigid, whatever you feel like. But because that's rigid, this is rigid, this is rigid, this is ridged. Because these operations preserve rigidity.

They don't preserve locked or whatever, but they preserve rigidity. Once you know that this is rigid you know that it's strongly locked. So when you perturb it so these guys are not on top of each other, but they're slightly spread out, it will be locked within epsilon, for any epsilon you want.

Now this is super easy. And this is how we were able to iterate through all those locked trees, and say, oh yeah, this is still rigid, so it's still strongly locked. Now this doesn't always work. But it seems pretty good.

And one of the conjectures on the tables is that for linear trees rules one and two are kind of almost enough. It's not literally true. But it's hopefully mostly true.

AUDIENCE: It seems that that argument would mean you wouldn't need one of the n's on--

PROFESSOR: Right, so we could think about this example. And you're asking do you need-- does this need to be that long or could we throw away the last bar? Right, it looks like that's good. If you throw away this bar it's still the case that this edge you can apply rule one, and say that it's pinned against this edge.

And once those are there, this thing basically acts as a single triangle, and life is good. Yeah, so you can remove this one edge. You could not remove both of the edges, at least for rule one to apply, because then this guy's not wedged into anything. He's wedged on this side but he's not wedged on that side.

But you can remove this edge. And that's a super easy way to prove that this thing is locked. We didn't know this when we wrote the textbook, otherwise we would have given that proof. There's one based on stresses in the textbook.

But here, yeah, you can make very easy judgments like that. Now it doesn't mean that it's not locked when you remove two of the edges. I'm not sure, no I think it's not locked. But just because the rules don't apply doesn't tell you that's not locked. But it at least makes it hard to prove. And it's a good sort of guideline.

Questions? Yeah.

AUDIENCE: Did the perturbations that are positive and finite have numbers? Like, what's-- it's sort of disturbing that they get very, very small-- what are they?

PROFESSOR: So you want to know how big is delta? How big is epsilon? Well it depends, of course, how much motion you want to allow, how small the perturbations have to be.

And I don't have a great answer. I do recall that we computed a bound, probably in terms of something like r, the maximum distance divided by the smallest non 0 distance. So it depends how close to tight you are in other places. And it depends on your epsilon.

There is an explicit bound. I think it's polynomial on those two things. But I don't quite remember.

So you can actually compute how much perturbation will give you locked within epsilon. But it's certainly not clean. It's actually not too hard to prove this statement just using topology.

Should I try to remember how to prove it? Basically, yeah, so there's this fun fact. Suppose you have some tensegrity.

So tensegrities are kind of hard. They say, look, a bar has this length, ain't changing. Let's be a little more flexible. What if you said, oh, this bar can change within epsilon? Because in reality you could probably pull the metal a little bit, just not very much.

Struts, it's not supposed to get smaller. Let's say it can get epsilon smaller than it's supposed to. It turns out if you take some tensegrity that's rigid, and then you add this little bit of flexibility so the edges can change in length a tiny amount, then before you made this change your configuration was a point, and there might have been other stuff. But locally you couldn't move at all.

If you add this flexibility, the new picture is a point with a tiny ball around it. And whatever, I mean this might change a little bit also. But the point is this point doesn't get much bigger when you add just a little bit of flexibility.

This is a fact that was known in rigidity theory. It's called sloppy rigidity. And it's essentially what's going on here-- that we're adding a little bit of perturbation. Before you couldn't move at all. Now you can move a little bit. We just had to generalize from regular tensegrity so these weird tensegrities with sliding struts.

And this is kind of intuitive. To really check it you just need to check that the constraints on edges and struts are closed sets, and then the have to behave this way. But you can actually compute how quickly they change. It's just messy. So I'll leave it at that. Hey, there's a little proof addition, glad I still remember it.

Other questions? This is the end of 2D trees. And now I want to talk briefly about 3D chains. So we did this one, now I want to do this one.

Actually this is one of the oldest results, from 1998. So right around the same time as the locked trees. I may have shown this example last time, or in lecture one. But here it is again.

Three bars in the center, and two really long bars in the ends. We call this knitting needles, because it's like two knitting needles with a short string connecting them, tied in a knot, sort of.

Topologically this is not knotted. If you could add extra creases here you could pull that through, no problem. But if this is rigid, like a linkage, then this thing is locked provided each end bar has length strictly greater than the sum of the middle bars.

So there's three middle bars here, add up their lengths, it should be shorter than this one and it should be shorter than this one. That's the cutoff. And I believe once it's the other way around, this is not locked.

Sadly, rules one and do not prove this thing is locked. They only work in two dimensions. But for this one example-- and to tell you the truth, this is pretty much the only example of a locked open chain that we have-- there's a really simple, nice proof. Let me draw the picture again.

AUDIENCE: Do you need the bottom one?

PROFESSOR: You're asking do I need three bars in the center, or could I get away with two? In fact, to draw this in 3D you need three bars. That's maybe not obvious.

But if you tried to draw it would just four bars, like this, it's not really possible

because these three points are coplanar, as all three points are, and then yeah. You can't get this weaving pattern.

This guy's going to be either above or below the plane. And this guy's going to be above or below the plane. And in all cases, you don't get this weaving. So you really need the five.

This is the minimum. You can prove all four bar 3D chains to not lock. But with five you can do it. Good question.

So here's how we're going to prove that this thing is locked. So there are these three edges, they have various lengths, who knows? But add up the lengths, divide by 2, and measure out that far. So I'm going to call that the midpoint of those three segments.

I want to center a ball here. It's going to look something like that. It's a 3D ball centered there. So it's a ball, diameter equal to the sum of the middle bars. So radius is half that.

And the center is the midpoint of the sum of the middle bars, whatever. So the radius is half the sum of the middle bars. And this is at half the sum of the middle bars. Therefore these middle bars stay inside the ball.

Maybe they touch the boundary. But they're inside or on the boundary of the ball. So that means the middle bars are in the ball. Maybe just barely, but no matter how you move this thing-- I mean if you move this point then the ball moves with it. So no matter how this thing folds, those three bar stay inside the ball.

What about these bars? Or what about the endpoints? Well if this point is inside the ball, which it is, and this point is inside the ball, which it is, then this thing is longer than the diameter of the ball. This thing is greater than the sum of the middle bars. The diameter of the ball is the sum of the middle bars.

So if I take any point inside the ball and move straight from there by the radius of the-- more than the diameter of the ball, I must go outside the ball. So the endpoints

are outside the ball.

How the heck are you going untie that not when all of the interior vertices stay inside of the ball, and these guys stay outside the ball? To formalize how the heck, you can say well if you ignore what's inside the ball-- something's happening there, who knows? But outside it's like there's a ball and there's two sticks coming out of it. So you can just imagine, for example, tying a string between the two ends.

And something happens in the inside. But if you think of just from the outside perspective, these sticks just move around. It's really easy to not tangle this cord when these sticks move around.

So now, in order for this thing to become unlocked, to straighten out for example, somehow inside you have to do some magic to get rid of this topology. Well what do I mean? Well if you could do it inside, you could do that motion even when there's a string tied out here.

But when I tie the string out here, it is a knot. It's a trefoil know. There's no way to untie a trefoil know without crossing. So either there's crossing in here-- which better be-- or there's crossing out here.

There can't be crossing out here. It's just two sticks and a string. You can easily arrange the motion of the string to not cross the two sticks.

Therefore this thing in fact cannot untie. Therefore this thing is locked, cannot straighten out. So that's locked 3D chains. Let me tell you a bunch of cool open problems.

This is really the only good example of a locked 3D chain we have. And it has length ratios, the best you could do is like 1 to 3 plus epsilon. If each of these is length 1, these have to be 3 plus epsilon.

Is that the best? Or could it be, for example, that all the edges or between length 1 and 2, and the chain is locked? We don't know.

In the extreme case, what if all the edge lengths of the same, all length 1? Is there a

locked 3D chain? We now know there's a locked 2D tree, but for chains it's tricky. It's even open if you add thickness.

You say, hey, let's think about a 3D chain, all the edge lengths are the same, and you get to specify some radius of the bars. For a while I thought maybe this was locked. I don't think it is. We can unfold.

All of these questions are open, and pretty fascinating. Especially because proteins are a lot like equilateral-- like all the edge lengths the same-- 3D chains. It's even open for equilateral 3D trees.

So we know 2D equilateral trees can lock. But 3D, it's open. I think that's enough open problems. Lots of cool questions here. All right, I have one more. It's just fun.

Even if you have a 3D chain where all the edges are on top of each other on a line segment-- it's like a linear tree but now it's a linear 3D chain-- it's like a bundle of segments. Is that locked? I don't think so. But even that is tricky to come up with algorithm.

Maybe we'll work on some of these in the problem session. That's it.