PROFESSOR: All right. Let's get started. So we have a fun lecture today about efficient origami design. Last Monday, we did inefficient origami design, but it was universal. We could fold anything. And let's see, Thursday, we talked about some basic foldability, crease patterns, what make the valid.

That'll ground us a little bit today when designing some crease patterns. Although, we're going to stay fairly high level today because there are two big methods I want to talk about. One is tree method which is hit it pretty big in the impractical origami design. Lot of modern complex origami designers use it either in their head or occasionally on a computer.

I demoed it quickly last time. So we are going to see some level of detail how that works. And then, I want to talk about Orgamizer which is one of the latest techniques for designing crazy, arbitrary, three-dimensional shapes that seems to be pretty efficient. Although we don't have a formal sense in which it is efficient, it has some nice properties.

And it's pretty cool, and I can also demo it. It's also freely downloadable software for Windows. Good. So just to get you motivated a little bit, I brought a bunch of examples. I'll show you more later. But this is the sort of thing you can do with the tree method. It's not going to be the tree method as I presented here.

This is a variation on it called box pleating which you can read about in *Origami Design Secrets.* And I don't think Jason will talk about that either. But it's a variation on what we'll be talking about. It lets you do crazy things like these two praying mantises, one eating the other.

This is a design by Robert Lang. Fairly new. I don't have a year here, but I think it's last year or something. And that's the sort of thing you can do getting all the limbs, all the right proportions, even multiple characters by representing your model as a stick figure.

And that's what the tree method is all about and doing that efficiently. So this is a statement last time of the theorem. There's some catches to this. It's an algorithm. Find a folding of the smallest square possible into and origami base with the desired tree as a shadow or as a projection.

So you remember, this kind of picture. You want to make a lizard. You specify the lengths of each of these limbs and how they're connected together into a tree. And then, you want to build an origami model on top of that, so to speak. So that it looks something like this.

And you want to find a square the folds into such a shape. This projection is exactly that tree. Now, say it's an algorithm, and it finds the smallest square. But to do that, essentially requires exponential time. We'll prove in the next class that this problem, in general, is NP-complete.

So it's really hard. But there is an exponential time algorithm, and I didn't say efficient here. It's efficient in terms of design, quality, or in terms of algorithm. But you have to pick one of the two. So in TreeMaker the program, there's an efficient algorithm, which finds a reasonably good-sized square. But it's not guaranteed to be optimal. It's just a local optimum.

In principle, you could spend exponential time here. So slow algorithm and get the smallest square. So it depends. The other catch is this folding. We're still working on proving that this does not actually self-intersect in the folded state. I checked the dates. We've been working on that for six years. But it's closing in. Maybe next year we'll have a draft of this proof. It's quite-- it's many, many pages.

Good. So those are the catches. Now, let me tell you about this term uniaxial. Essentially, it just means tree shapes. But I'd like to be a little bit more formal about that. And last time, I showed you the standard origami bases. All of these are uniaxial, I think, except the pinwheel which we folded.

So the pinwheel-- so let me tell you intuitively what uniaxial means. It means you can take all these flaps of paper and lie them, place them along a line. And the

hinges between those flaps are all perpendicular to that line. So this is the axis. Whereas something like this, essentially there are four axes. The flaps are here, or two axes I guess.

But definitely not one. So these cannot be lined up along a line, even if you've flapped them around some other way. That's intuitive definition. Multiaxial is not a formally defined thing. But uniaxial we can formally define. And it will capture things like this water bomb base, all the other bases there, as well as bases like this.

And it's defined by Robert Lang, I think probably around '94 was the first publication. And it's just a bunch of conditions. And a bunch of them are just technical to make things work out mathematically. First thing I'd like to say is that the entire base-- base just means origami for our purposes. It's sort of practical distinction not a mathematical one. Is that everything lies above the floor.

So the floor is equal to zero, and we'll just say everything's above that. And the action is going to be in the floor. That's where I've drawn it that way. Here, there's a floor. And the tree is going to lie on the floor, and everything else is above that.

Second property. Sort of a shadow property. If I look at where the base meets the floor is equals to zero, that's the same thing as if I look at the shadow onto the floor. This is essentially saying that this base does not have any overhang.

So if it had, for example, some feature like this that hung over its shadow-- was more-- went out here. The shadow goes out here, but the base does not. That's not allowed. So I want everything-- actually want things to get smaller as you go up in z.

This is a stronger statement of property two. And then, I want to define this notion flaps. And the basic idea is that you have faces of the crease pattern. so the faces are just the regions we get out of the creases, all these triangles for example. I can divide them, partition them into groups which I call flaps.

So for example, these two guys over here form one flap. They fold together. They're going to be manipulated together. And so in this case, I'll get four flaps. Anything I want to say here? Yeah. Each flap is going to project to a line segment.

It's going to be one of the edges of the tree. So then, there's the notion of a hinge crease. And these are just creases shared by two flaps. So they're the creases that separate one flap from another. These will always require that they projects to a point.

So this is equivalent to saying the hinge crease is vertical. It's perpendicular to the floor. I'm always projecting straight down onto the floor orthographically, just setting z to zero. And so that's saying these are the hinges. They should be vertical. So projection is a point.

And then from those two properties, I can define a graph which I want to be a tree. So each flap I want to make an edge of my graph. And that edge is going to be the line segment that the flap projects, each flat projects to. And I'm going to connect those edges together at vertices when the flaps share the hinge crease.

All right. That's a graph which you can define. And that graph is a tree. That's the constraint. And I think I have even more. I've got one more property. I think I actually want projects here. Let's try that.

All right. This is a bunch of formalism to state what's pretty intuitive. I want all the flaps of paper to be vertical, so they project to a line segment. When I look from the-- when I look at the projection, I can define a graph where there's an edge for each flap, where it's projecting.

And I join those edges together. Here, I'm joinging four them at a vertex. Because if you unfold it, they all share hinge creases. Hinge creases in this case are the perpendicular. These four guys. So because-- it's hard to manipulate. I've got a flap over here. A flap over here. They share a hinge, so I connect them together in the graph.

It's just a formal way to make the graph correct. It may seem tedious, but this definition sidesteps some issues which would occur if you defined it in the more obvious way which is just take the projection, call it a tree. But I don't want to get into why you need to do it this way exactly. Maybe, we'll see it at some point.

Essentially, some flaps can be hidden inside others, so you need this definition for it to really work. And then, there's this extra constraint which is that my base is pointy at the leaves. Leaves are the vertices of the tree to have only one incident edge. And so I want there to be only one point that lives at the leaf.

Obviously, elsewhere in the tree, there's a whole bunch of points, a whole vertical segment, that all projects to that point. Here, I just want one. That's important because I want to think about where the leaves are. And the whole idea in the tree method is to think about how to place the leaves on your piece of paper so that this folding exists.

So that's what we're going to do. The tree method is kind of surprising in its simplicity. There's a bunch of details to make it work. But the idea is actually very simple. Let's suppose you want ability uniaxial base. I'll tell you something that must be satisfied by your uniaxial base, a necessary condition.

Assuming you're starting from a convex piece of paper, which is the case we usually care about. Actually, we're starting from a square, a rectangle, or something convex. Here's what has to be true. I didn't give a name, but this graph that's supposed to be a tree, I'm going to call the shadow tree for obvious reasons.

And now, I want to take two points in the shadow tree, measure their distance in a tree sense. So I have some tree like this. I have two points like, say, this point and that point. The distance between them is the distance as measured if you had to walk in the tree, how far is it to go from here to here.

And because our tree is a metric tree, because we specified all the edge lengths, we can just add up those lengths, measure them. And that's the distance between two points in the tree. That must be less than or equal to the distance between those two points on the piece of paper.

What does that mean? So on piece of paper that's convex-- so it might not be a square, but square's easier picture draw. The distance between them is that. Pretty

simple. So what does this mean? I'm taking this square. Somehow, I'm folding it into a base whose projection is the tree.

So I look at these two points, p and q, I fold them somewhere in the 3D picture which is not drawn up here. Those points-- so maybe there's a p up here and a q up here. I project those points down onto the floor which is going to fall on the tree by this definition.

Call that, let's say, p prime for the projected version of p, q prime. I measure the distance here. That has to be-- the distance between p prime and q prime in the tree should be less than or equal to the distance between p and q in the piece of paper, for every pair points p and q.

That's the condition. It's almost trivial to show because when I take this segment of paper, I fold the piece of paper. But in particular, I fold p and q somehow. I can't get p and q farther away from each other because folding only makes things closer.

There, I'm assuming that the piece of paper is convex. There's no way to fold and stretch pq because that's a segment of paper. It can only contract. I mean, you can fold the segment something like this. Then, the distance between p and q gets smaller than the length of this segment. Because if I took this-- this line segment of paper that got folded. If I project it onto the line here, it's only going to get shorter.

So I fold p and q. They get closer in three-space. And then, I project them down to the floor. They can also only get closer when I do that. So that's essentially the proof. Do I need to spell that out? So you have the line segment on the paper. You fold it. It gets shorter. You project it onto the floor. It also get shorter.

Therefore, whatever this distance is on the tree has to be less than or equal to the distance you started with. So this may seem kind of trivial. But the surprising thing is it this is really all you need. So this is true between any two points in the shadow tree.

In fact, we're going to focus on the leaves. We'll say, all right, so in particular, I've got a place this leaf, and each of these six leaves here, I have to place them

somewhere on the piece of paper. I better do it so that that condition is satisfied. I have to place these two leaves and the piece of paper-- let's say this distance is one, and this distance is one.

These two leaves have to be placed on the piece of paper such that their distance is at least two. And the distance between these two guys has to be at least two and between these two guys has to be at least two. And same over here. Let's say all the edge lengths are one. And the distance between, say, this leaf and this leaf has to be at least three because the distance in the tree is three.

So at the very least, we should place the points on the paper so that those conditions are satisfied, and it turns out, that's enough as long as you find a placement of the points such as those conditions are satisfied. There will be a folding where those leaves actually come from those points of paper.

That's the crazy part. But this idea is actually kind of obvious in some sense. I mean, once you know it, it's really obvious. But what's surprising is it this is all you need to worry about. There's a lot of details that make that work, but you can.

So let me just mention one detail which is the scale factor. If you fix the size, the edge lengths on the tree which is the usual, which is one way to think about it, and you start with some square-- like if I start with a one by one square, there's no way I'm going to fold that tree. There's just not enough distance in the square.

So what I'd like to do is find the smallest square that can fold into this thing. Or equivalently find-- you can think of scaling the piece of paper, or you can think of scaling the tree with a fixed piece of paper. Doesn't really matter. In general, you get this problem which I'll call scale optimization. This is the hard problem.

So let's say-- just defining some variables. So P i, I'm going to maybe number the leaves or label label them somehow, various letters. And then, P i is going to be the point where that-- of paper that actually forms that leaf in the folded state. That leaf which corresponds to a single point of paper projects to that leaf.

And then, my goal is to maximize some scale factor which I'll call lambda. Subject to

a bunch of constraints which are just those constraints, except that I add a scale factor. So for every pair of leaves, i and j, I'm going to measure the distance between those leaves in the tree. This as a tree distance.

Compare that to the distance and the piece of paper between those two points, the Euclidean distance. And instead of requiring that this is greater than or equal to this, which is the usual one, I'm going to add in the scale factor which you can think of as shrinking this or expanding that. It doesn't matter. But I want to-- because here I'm sort of shrinking this amount.

I want to maximize that factor, so I shrink it the least possible. You can formulate it this way or maybe a more intuitive way. But this is the standard set up. And this is something-- this is called a nonlinear optimization problem. It's something that lots of people think about. There are heuristics to solve it.

You can solve in an exponential time. In general, it's NP-complete, and we'll see next class that actually this problem of origami design is NP-complete. So there's not going to be anything better than heuristics and and slow algorithms. So the idea is, you solve that.

Now, you have your leaves on your piece of paper somewhere. Now what? Now, you have to figure out how everything folds. That's where we get to some real combinatorial, some discrete geometry. Fun stuff. Yeah. I have one extra motivation here.

Origami design is fun, but here's a puzzle you can solve, too. Which we can already see. Margulis napkin problem. Origin of this problem is not entirely clear, but I think it came from Russia originally. And the problem, the puzzle is usually stated as follows.

Prove that if you take a unit square paper-- so it has perimeter four that you can, no matter how you fold it, the perimeter always gets smaller. Never bigger than four. We used a very similar thing here. We said, if you have two points, they're distance can only get smaller.

That's true. Margulis napkin puzzle is not true. That's the difference. Perimeter is different from distance. And in fact, you can fold a piece of paper to make the perimeter arbitrarily large, which is pretty crazy. And this is something that Robert Lang proved few years ago, using--

It's sort of easy once you have the fact-- which I haven't quite written down here, but I've been saying. As long as you place your points subject to this property, there is a folding that has that shadow tree. And so the idea with the Margulis napkin problem is let's make a really spiky tree, a star.

I want to fold the smallest square possible, so that projection is this thing. Let's say that it has-- I won't say how many limbs it has. But the idea is, if you're using paper efficiently, in fact, the folding will be very narrow. It'll be a pretty efficient use of paper, hopefully.

And so the actual 3D state will just be a little bit taller than that tree. And then, you just wash it. And the idea is that then the perimeter is really big. You've got a-- the perimeter as you walk around the edges of that tree. So how big a tree can I get? I'd like to somehow place these leaves-- now, what's the constraint on the leaves?

Let's say all of these are length one. Then, this says it every pair of leaves must be at least distance two way from each other. So I got to place these dots in the square so that every pair has distance at least two. This is like saying-- here's my square. -- I'd like to place dots so their distance is at least distance two.

That's like saying if, I drew a unit disk around two points-- I got to remember. You should always draw the disk first and then the center. Much easier. Those disks should not be overlapping. If this is length one, and this is length one, the disks will be overlapping if and only if this distance is smaller than two.

I want it always to be greater than or equal to two. So I just have to place a whole bunch of disks in the square so that they're not overlapping. So how big a square do I need to do that? This is a well-studied problem, is the disk packing problem. A lot of results known about it. It's quite difficult.

But we don't need to be super smart here to get a good bound. Let's put a point-- let's put points along a grid. I'm going to regret making such a big grid. Let's say, an n by n grid.

And I'm going to set the size of my disks right so that these guys just barely touch. This is actually not a terribly good packing. You should do a triangular grid instead of a square grid. But it'll be good enough asymptotically. You get the idea.

If I set the size of my paper to be n by n, I can fit about n squared unit disks in there. N by n paper folds something like n plus 1 squared. But let's just say, approximately n squared disks. So that means I can make a star with about n squared limbs. It's insane. It's like super efficient.

Each of these little portions of paper ends up being one of these segments. That's the claim is, you could fold that. So once you fold this thing, I have an n by n square. You started with perimeter about four n And now, I have perimeter about n squared. That's huge with respect to four n.

So this is much bigger than four n, for n sufficiently large.

AUDIENCE: [INAUDIBLE] about the length flaps.

PROFESSOR: Here, I was assuming all the flaps are length one. So the disks are size one, and so it's an n by n square. Clear? So this is more motivation for why this theorem is interesting. It lets you solve this fun math puzzle and show not only does a perimeter not go-- not only does the perimeter not only go down, but it can go arbitrarily high.

It just takes a lot of folding. So let's say something about how we prove that once you have a valid placement of the points, you can actually fill in the creases, find folding. Let me bring up an example. So this is actually the example I keep using which is, you want to make a lizard or some generic four-legged tail and head kind of creature.

This is the output from TreeMaker, complete with crease pattern and everything. But here, I've labeled all the-- or actually Robert Lang, I think, has labeled all of-- this a figure from our book. --all the vertices of the tree and the shadow. And then, we're labeling where they come from on the piece of paper.

So in particular, you see something like a leaf h. And it comes from this one point on the paper. This leaf d comes from this point, and g comes from that point. It's actually kind of similarly oriented to this guy. The interior vertices, they come from several points. It's a little messy.

But let's-- one of the things is to try to locate where those points ought to be. s So there's this idea of an active path which is a path in the tree between two leaves. I'll call them shadow leaves to say that they're in the shadow tree.

And the length of that path equals the distance in the paper. So in the case of making a star graph, this is exactly when the disks kiss, when the just touch each other on the boundary. So in other words, we have this inequality, saying the distance between in the paper should be greater than or equal to distance in the tree.

If that inequality is actually an equality, if they're the same thing, then it's kind of critical. I can't get those points any closer in the paper. Those things I call active paths. And that is some of the lines up here. I guess the black dashed line, actually, in a lot of the dash lines. All of the dash lines, I think.

So for example, d to h, that's a distance between two leaves. And if you measure the distance here, it's two. And just imagine, this example has been set up so this is exactly two. So this is tight. I can't move h any closer to d or vice versa. And also from h to a, a is actually in the middle of the paper and corresponds to that flap.

That's all of those green, actually it's just the green lines, green dashed lines are active. They're kind of critical. And what's nice is that subdivides my piece of paper into a bunch of smaller shapes. So I have a little triangle out here. That turns out to be junk. We're not going to need it because the sort of outside the diagram.

You could folder underneath. Get rid of it. You've got a quadrilateral here between the green lines. We've got a triangle up here, a triangle at the top, triangle on the left. All we need to do is fill in those little parts. Fill in that triangle. Fill in that quadrilateral.

Of course, in general, there might not be any active paths, and we haven't simplified the diagram at all. But if there are no active paths, you're really probably not very efficient. That means none of these constraints are tight. That means you could increase the scale factor lambda, make a better model.

You can increase lambda at least a little bit. If all of these are strictly greater, you can increase lambda until one of them becomes equal. So you should have at least one active path. And in fact if you're efficient, you should have lots of active paths. I don't think I need to be too formal about that. But it's true.

And here's one thing you can show about active paths. So what would be really nice, in this example, I have triangles and quadrilaterals. In general, I'm going to have a whole bunch of different shapes. Some of them could even be non-convex which would be annoying.

I would really just like to deal with triangles because I like triangles-- geometer. And triangles are simple. And it looks like the crease pattern in a triangle is pretty simple. In fact, it's just angular bisectors of the triangle plus a few extra perpendicular folds. So that would be kind of nice if I could get everything triangles. To do that, I need lots of active paths.

So how can I guarantee that there's lots of active paths? I'm going wave my hands a little bit about how this is done. But the idea is to augment the tree. So I have some tree that I actually want to make, like lizard. And I'm going to add some extra stuff. Like maybe I'll add a branch here and a branch here or whatever. Whatever it takes.

I got to do so carefully. So let me say what that means. So I'm going to add extra leaves to the shadow tree. My goal is to make the active paths triangulate the paper

without changing the scale factor.

So this is kind of a cheat. And most of the time, you don't actually need this cheat. But for proving things, it makes life a little easier. So we want to show that it's enough to place the vertices subject to this, the leaves subject to this constraint. So ideally, we make our tree.

But if we make an even more complicated tree, like with these extra little limbs, we can get rid of them at the end. You just fold them over and collapse this flap against an adjacent flap. So if we make our life harder, that's OK, too. If we could fold a more complicated tree, in particular we folded the tree we wanted.

If we can do that without changing the scale factor, then great. Then, we did what we wanted to do. We folded our piece of paper with the desired scale factor. In reality, we're actually going to move the leaves around a little bit so that we have to do.

We're going to move around the leaves that you already placed in order to make room for the new leaves. But here's the idea. We have these leaves. There's some active paths, these green lines. And we'd really-- we have this quadrilateral in the center. We'd really like to subdivide it. Like this black line is kind of asking for it.

It would be really nice if we could just add in an active paths there. And you can do it. Let's see if I can identify what we're talking about here. So a fun thing about active paths, you look at two leaves like g d here, which corresponds to this path g d here, because it's active, you know this length is exactly the length traced right here.

So that means, this segment has to be folded right along the tree here. You know that this segment is that. And so in particular, you know where c is on that segment. C actually comes from multiple points in this diagram. But you know that this point right here must fold to c. And you know this point must fold here and so on. These guys correspond.

So that's good. So if I look at this quadrilateral, it corresponds so g to c to d to c to h to c to b to a back to b back to c. And so my guess is if you add a little limb in here--

I think I can draw on this. That would be nice. Should really tell you about-- is this going to work? Yes. It's kind of white, but there we go. So great. Draw a fun diagram here.

This is how I make my lecture notes if you're curious. This is a tablet PC. Now, I've got some-- Tell me if I make a mistake, those who know what I'm drawing. What the hell is this? I think it goes there. There. There. I'll explain what I'm drawing once I've drawn it. It's easier. Something like that.

This is a bunch of disks and a bunch of other things, there's only one here called rivers. And this is a geometric way to think about the constraints. If you look at this structure-- so I have a disk down here corresponding to d. I have a disk corresponding to h, a disk corresponding to g, a river corresponding to the segment b c.

The reason I only have one river is there's only one interior edge in this tree. Everything else is a leaf edge. So leaf edges are going to be disks. All non leaf edges are going to be rivers. And the structure, the way that those things connect to each other is the same as the structure in this tree.

So you've got the three disks down here, which corresponds to these leaf edges. They all touch a common river because all of those edges are incident to that edge in the center. And there's three disks on the top that correspond to the three leaf edges up here.

This is really just the same thing. It's saying that if you want to look, say, at the distance between h and a here. The distance between h and a should be length three. And those three lengths are represented by the size of this disk, followed by the width of this river, followed by the size of the a disk.

It's say exactly the same constraints, just represented geometrically. Now, if I'm lucky, these regions actually kiss, they touch at points. That's when things are active. And you could draw straight across from a to h and never go in these outside regions.

If you're not lucky, they won't touch. If they don't touch, make them touch. That's all I want to do. And so I just want to blow up these regions, make them longer, for example, until things touch. When they touch enough, if you do it right, you can actually get them to triangulate.

That's my very hand wavy argument. It's proved formally in the book, and it's a little bit technical. So I think I will move on and tell you what to do with triangles.

So suppose you have some triangle. And each of these edges is an active path. So there's some leaf here. We'll call them a, b, and c. And this segment we know will map right along the floor to make up that path, that active path in the tree. Like I said, we're going to follow along angular bisectors.

You may know the angular bisectors of a triangle meet at a single point. And then, we're going to make some perpendicular folds like that. Where the perpendicular folds go, well, they go whenever there's a shadow vertex along this segment.

Remember this edge, b c corresponds to some path between b and c in the tree which looks like whatever. And so for each of these branching points that we visit along that, we can just measure. As we move along here, we get to some vertex then another vertex then another vertex then c, except I did it backwards.

And so for each of these guys, I know that I need to be able to articulate there. I need a hinge crease. And so I just put in a hinge grace perpendicular to the floor, essentially, because we know this is mapping to the floor.

And conveniently, those will all line up. So if I have some vertex here-- let's call it d. --d will be here. But d will also be here. Because if I follow the path from b to a, a is some other guy, maybe this one, I also have to go through d. And so these things will conveniently line up perfectly. I'm not going to prove that again. But it's true.

And you just get this really nice simple to fold thing. Shoot, I'll fold one if you haven't. This is a standard rabbit ear molecule in making origami. You have a little triangle. You want to make it an ear. You squeeze along the angular bisectors, and it makes

a cute rabbit ear.

And you can see it also, the crease pattern, in here like in this triangle in the upper right. You've got the red lines which are the angular bisectors. And then, you've got all those perpendicular folds. And they go exactly where those letters go. And the triangle at the top is similar.

It's a little different because the very top edge of the paper is not actually active. So there's really a special case there. Upper right is also not active. Oh, that's annoying. Yeah. There's a little bit of extra stuff that happens at the boundary of the paper where you don't have active paths. But it's, as you can see from the crease pattern, it's basically the same.

In fact, I could call it the same. It's a little bit less pretty because this is not green. And so you don't actually know that c is here. And you don't know that b is there. But you know about all the other edges. There's just one edge you might not know about. And so you can figure out what the right edge is based on the other edges of the triangle, the other two edges.

That's just a feature. You can triangulate everything except the boundary. You may not be able to get active paths in this step. That kind of does the tree method in a super abbreviated version.

I showed you a demo last time, just in case you forgot. You draw your favorite tree. See if I can get it to do the same one. And you optimize, generate a crease pattern. Oh, it's a different one. Fun. There it is. And here, TreeMaker knows how to draw the disks. It doesn't currently know how to draw the rivers because it's kind of tricky to make a snakey path in a computer program.

But you see the three disks down here, the three disks up there, and you can imagine the one river in the middle representing the central segment of your tree. And one of the problems on the problems set, problem set one is released, is to just make something using TreeMaker.

I would encourage you to start simple unless you know what you're doing. You don't

have to use the program. You could do it by hand, placing disks. That's how most origamists actually do it. I'm sure Jason will do it that way. You can use the program, print out a crease pattern, see what it looks like.

Next thing. If you want to do this in reality-- and what TreeMaker is doing is not this triangulation. Doing a triangulation is a bit of a pain, but you could keep modifying your tree until it triangulates. The alternative is you just deal with polygons that are bigger than triangles.

And there's this thing called the universal molecule by Robert Lang. Here it is for a quadrilateral. And it makes it-- this works for any convex polygon. Now sometimes, you're active paths don't decompose your shape into convex polygons. And this still doesn't work. You still have to do something here. You need to add some extra leaf edges to the tree to just fill things up.

But you don't have to stop. You have to go all the way to the point of triangulation. You can stop at the point which happens most the time when all of the faces are convex. And then, it's a slightly more general picture what happens. Intuitively, what you want to do is, this is the tree you want to make among those leaves.

All the boundary edges here are active paths. You have g d h a. Those are active paths, so you know where all of those branching points are in the middle. You'd like to build that. And so what we're going to do is build it bottom up in the literal sense from z equals zero, increasing z.

And what that corresponds to in this picture is shrinking or offsetting these edges inward. So you offset these all by the same amount. That's like traveling up over here. So you see the red lines here correspond to the red cross sections. So I just see what happens in cross section is I shrink things in.

And the first thing that happens at this first critical red drawing is that the path from d to a becomes critical, becomes active. Before it was inactive that-- that was kind of annoying for me. I wanted it to be triangulated, but it wasn't. The distance from a to d in the piece of paper was bigger than the distance between the leaves in the tree.

I wanted them to be equal. Well, it turns out, if you shrink this thing, eventually they might become equal. And that's what happens. And that's what TreeMaker computes. And what you should do if you're building the universal molecule. If you discover, oh, now a d is active, now, I subdivide into two triangles. And then, I do the thing in the two triangles.

And generally, you start with some convex polygon. You shrink it. At some point, some diagonal might become active. You split it into two, just keep going in the two. And there's one other thing which can happen, which is what's happening at the end of a triangle. You shrink. And then, it could be two vertices actually collide with each other.

And then, you just think of them as one vertex and keep shrinking. So that's the general universal molecule construction. You see in a sort of-- these are the cross sections from above. You see that as you go up, things are getting smaller. That is one of the statements of the uniaxial base as you go up. Cross sections get tinier.

And that gives you the crease pattern. If you follow along where the vertices go during this process, and you draw in and all the active path that you create along the way, that's your crease pattern. So that's how you do it more practically is you use the universal molecule. But to prove it, you don't actually need that.

All right. I have now some more real examples by Robert Lang and by Jason Ku. So here is Roosevelt elk. And Rob is all about getting very realistic form. So all of the branching measurements and-- I'm sure if you knew a lot about elks, you could recognizes this a Roosevelt elk not some other elk.

And you can achieve that level of detail and realism using the tree method because you can control all of the relative lengths of those segments and get perfect branching structure and get the right proportions for the legs and tail and so on.

And you can see here, the-- and you can go to Robert Lang's webpage, landorigami.com and print this out. And try it out if you want. This will fold not this but the base for that model. And you could see the disks. And you can see some

approximation of the rivers here. But they're not quite drawn in in this particular diagram.

But a lot of detail. And if you look carefully, you can really read off what the tree is here. You can see how these things are separated, and it will correspond to the branching structure over there. Here's a more complicated one. Scorpion varleg which you can also fold at lifesize if you're really crazy.

And you can also see from these kinds of diagrams that paper usage is super efficient in these designs. And presumably that's how Robert design them. The only paper we're wasting in some sense is the little regions between the disks and the rivers which is quite small. Most of the papers getting absorbed into the flaps.

Here's one of the first models by Jason Ku that I saw, the Nazgul from *Lord of the Rings*. And pretty complicated. So here, the bold lines show you essentially where the disks and the rivers are that have been--

AUDIENCE: Those are actually the hinge creases.

PROFESSOR: Oh, those are hinge creases. Yeah. Good. And the top is the actual crease pattern. And it's pretty awesome. You've got a horse and rider out of one square paper. Here's a shrimp. He's super complicated and super realistic. It looks very shrimpy. I know some people who are freaked out by shrimp. And so this should really elicit that similar response.

Or other people get really hungry at this point, I guess. But you could see the tree is pretty dense here, lots of little features getting that branching right. And one last example is this butterfly which is pretty awesome in its realism. And I guess the tree is a lot simpler here. But there's a lot of extra creases here. You see just for getting the flaps nice and narrow.

So in general, these kinds of constructions will make this guy rather pointy and tall. And you can just squash it back. And it's called a sync fold and make it tinier like-- you have something like this. The flaps are you think are too tall. You just fold here. Which, if you look at the crease pattern, makes just an offset version of the original.

And hey, now your flaps are half is tall.

And if you're a proper origamist, you-- I shouldn't do this live. You change the mountain valley assignment a little bit, and you sync everything on the inside instead of just folding it over. It's not going to look super pretty. But same tree structure, just the flaps are half as tall.

So that's all this pleating here. And I think that's it for my little tour. And Jason Ku next. Next Monday we'll be talking more about the artistic side, history of origami design, and what it takes to really make something real by these approaches. That should be lots of fun.

I want to move on to other kinds of efficient origami design. Less directly applicable to real origami design so to speak, at least currently. But mathematically more powerful. Uniaxial bases are nice, but it's not everything you might want to fold. So what if we want to fold other stuff.

And to a geometer, most natural version of folding other stuff or folding anything is a polyhedron. You have a bunch of polygons, flat panels in 3D, somehow joined together to make some surface. How do I fold that?

And let's start with a super simple example which is I want to fold a square into a cube. How big a square do I need to fold a unit cube? Or how big cube can I fold for a unit square? Either way. And I'm going to make it a one by one square. And I'm going to fold it into a cube of dimension x.

And I want to know-- it looks funny. It's the quintuple x cubed. It's the x-coordinate. That's my motivation. So we talked-- one thing we can think about is what makes the corners of the cubes and how far away should they be.

So if I want to fold this cube, I look at, let's say, the opposite corners of the cube. They're pretty far away on the cube. And I know that by folding I could only make distance is smaller. So somehow, if I measure the shortest path on the cube, from this point to this point, it's that if you believe-- when you unfold this thing, it should be flat.

If I unfolds to just those two squares, it's a straight line between the two. And so that goes to the midpoint of this edge and then over there. And you measure that length. And oh, trigonometry. Root five, that's not what I wanted. So we have x here, 2x here. So this distance is-- yeah, I see. Why is that different from what I have written down?

Because that was not the diameter of the cube. I see.

AUDIENCE: You want them equidistant.

PROFESSOR: No. I do want this but, I think if I go from the center of this square-- this is hard to draw. --to the center of the back square, which is back here, that distance is going to be wrapping around. Which is just going to be like 2x. Is that bigger or smaller than root 5x?

AUDIENCE: It's smaller.

PROFESSOR: Smaller. Interesting. One, two, three, four. What did I do wrong? Oh, I see. OK. Here's a fun fact. This is actually the smallest antipodal distance. Get this right. So if you take some point on the cube, and you look at the point farthest away from it on the other side of the cube, it will always be at least 2x away.

So here, it's bigger. This is probably the diameter. It's bigger than 2x, but it will always be at least 2x away. This is actually the smallest situation you can get. And so I want to think about the point that corresponds to the center of the square. Right? Yes. Now maybe that maps to the center like this. And the antipodal points is 2x away, or maybe it's bigger.

But at least I know that this length is greater than or equal to 2x because that's-- the antipodal point has to be made from that. I need to think about all situation because I really want to think about the center of the square. Once that is at least 2x, then I know that the side of the square is at least 2 root 2x. Yes.

And so I know that this is one. And you work it out. And x is root 2 over 4. Or it's at

most that. And so that gives you some bound on what it takes. So this is actually really the only technique we know to prove lower bounds on how much-- how big a square you need to make something.

It's this kind of distance increasing argument. And it turns out you can actually achieve x equals this. So this is what I call lower bound. It says, you can't do any better than this. But there's also a matching upper bound which achieves this and not going to draw it perfectly.

So there are the six sides of the cube. You've got one, two, three, four, five. And the sixth one is split into quarters. And you can see, you just actually fold here, here, here, and here to get rid of that excess. And it will come together as a cube. You also fold along the edges of the cube. And it perfectly achieves this property.

That from the center of the paper, you have exactly one this distance 2 root 2x to the antipodal point which is the center of the opposite face. Question?

AUDIENCE: Sorry. Can you explain where the 2x came from [INAUDIBLE]?

PROFESSOR: I wave my hands. So I'm thinking about an arbitrary point on the surface of the cube. Here, it should be clear it's 2x. There's x right here. And there's 1/2x here. And there's 1/2x on the back. And I looked at another situation which is when it was at a corner that bigger than 2x.

And I claim if you interpolate in the middle, you'll get something in the middle, in between 2x and root 5x. For example, if take a point here that's closer to the corner, then that point-- you should probably also think about the edge case. But you check all of them, and they're at least 2x. That's what I'm claiming.

So I didn't really prove that formally. But claim is 2x is the smallest antepodal pair you could get.

AUDIENCE: What does antipodal mean?

PROFESSOR: Antipodal simple means on the other side. The anti pode, the opposite pole, like from North Pole to South Pole.

AUDIENCE: [INAUDIBLE].

PROFESSOR: Right now, we know we're taking whatever point is the center of the square. It maps somewhere in the cube. I take the antipode from there. I know that has to be at least 2x away. And if you look at the distance map here, the farthest away point in the squared from the center is the corner point.

So I know that that distance can only get smaller. And other distances only get smaller. So if I have to make a 2x distance from there, this is my best chance for doing it. And that gives you a lower bound. Doesn't mean it can be achieved. But this shows you that you can achieve it. This is a result by Catalano, Johnson, and Lobe in 2001.

It's like the only optimality result we have for folding 3D shapes. That's why I mention it. Tons of fun open problems like you don't want to make a square-- a cube. Maybe you want to make a triangle. If you want to cover a triangle on both sides, that's probably open. If you want to make regular tetrahedron, that's probably open.

Pretty much any problem you pose here is open. It would make fun project. You can also think about, instead of starting from square, you start with a rectangle of some given aspect ratio. What's the biggest cube you can make? That's kind of fun because in the limit for a super long rectangle, you should do strip wrapping.

For a square, we have the right answer. What's the right answer in between? Who knows. The next thing I wanted to talk about where there's been some recent progress is checkerboard folding.

In lecture one, I showed you this model which I never go anywhere without, the four by four checkerboard folded from one square paper, white on one side and red on the other. And so I think this is probably the most efficient way to fold a four by four checkerboard.

You start with a square of one size, and you shrink both dimensions by two. And

you get a four by four checkerboard. But we don't know if this is the best way to fold a checkerboard. Be nice to know. And this has been studied for a while. And this is not the standard method for folding a checkerboard. But it's actually pretty efficient which is kind of crazy.

So you take a square, white on one side and brown on the other. You do this accordion pleat. You get a bunch of nice color reversals, bunch of squares. And then, you just need to make a square of squares from that. So general problem is, I want to fold an n by n checkerboard from the smallest possible square.

How big does it have to be as a function of n? And the standard approach is-- well, this is the first method that does it for all n in a general simple way. But the practical foldings people have designed, like four by four and there are a bunch of eight by eights out there, are little more efficient than this.

But they have the same asymptotics which is the perimeter of the square you start with has to be about twice n squared to make an n by n checkerboard. And the reason that is, is if you look at the checkerboard pattern, we're trying to get color reversals along all of these lines between the red and the white, brown and white.

And the way we're doing that here is we're taking the boundary of the paper, and we're mapping it along all the color reversals. And if you work out how much color reversal is there in an n by n thing, it's about twice n squared. And so either your perimeter has to be at least that large if you're going to cover the color reversals with perimeter.

And for a long time, we thought that was the best we could do was to cover color reversals with a perimeter of paper. Of course, know that you can take a square of paper make the perimeter arbitrarily large. So this was never really a lower bound. We never really knew that you needed 2n squared.

The four by four achieves 2n squared. We think it's the best for four by four, but we proved last year-- this is with Marty and [INAUDIBLE] and Robert Lang-- that you can do better and get perimeter about n squared. Now, there are some lower order

terms there, the order n parts. So this is really only practical for large n.

I think-- I'll elaborate on that a little more in a moment. But here's the idea. Instead of visiting all the boundaries between red and white squares, I just want to visit the squares themselves. So if I could fold a, in this case a rectangle paper into this shape which has slits down the sides, and it has these flaps hanging out.

Now, you've seen how to make flaps super efficiently. You really don't need to shrink the paper by very much to make this pattern. Then, you take these guys-- and everything is white side up. You take these flaps, fold them over. They become brown.

And these guys fall over. These fall down. These guys fall up. You can actually make any two color pixel pattern from this idea. And it will make white squares on top of the brown surface that you folded. So this is the starting point. You just fold everything over. And you get your checkerboard.

And now, essentially, you're visiting each square only once instead of the boundary edge for all the squares. And so you end up using only n squared instead of twice n squared. And you can do it if you start from a square also. You just need more flaps. And there's a bunch of tabs sticking up here, and a bunch of tabs sticking up there.

You can fold this again super efficiently, using all these standard techniques. And then, you make a checkerboard twice as efficient for large n as we've previously thought possible. Now, we still don't know whether this is optimal. We think it is. But we thought so before also. Who knows?

So big open problem is [INAUDIBLE] for anything. In terms of actual values of n, for n bigger than 16, this method is better than the standard approach. Although if you look just at seamless-- so seamless, I didn't mention, but we're going to talk about it more in a moment.

When I make a square of a checkerboard, I'd really like this to be a single panel of paper not divided into little panels. And like in this checkerboard, this white square

has a bunch of seems on it. It's made out of three smaller triangles. And that's not so nice.

This method is seamless. You get whole panels making each of your squares, so it looks a little prettier. If you look at the best eight by eight seamless folding, this beats the best seamless eight by eight folding. Although it's rather difficult to fold. Hasn't yet been folded. That would be a good project also. Build an actual checkerboard with this method.

Questions? Now, I want to move to the general case. So I talked a little bit about checkerboards and about cubes. Let's think about arbitrary polyhedra. And this is the Origamizer. So Origamizer's actually two things. It's a computer program for Windows that you can download, and it's an algorithm.

And they're not quite the same. So there's original computer program and Tomohiro Tachi wrote a couple papers about it. That program does not always work. Doesn't make every polyhedron. It need some finesse to make it work, but it's super efficient.

And it's practical. He's made lots of models with it like the bunny you've seen on the poster. There's the algorithm, which we developed together. And we know it's similar. And we know it always works. But it's a little bit less practical. So it's-- theory's always a little behind practice, let's say.

So there's a practical thing here. There's also a theoretically guaranteed thing here. They're not quite the same, but they're very similar. I'm going to tell you a little. I'll show you both, basically. But the idea is, a practical algorithm to fold any polyhedron.

And practical here is a bit vague. We don't-- that's not a theorem. We don't know how to define practical in mathematics anyway. It has some fun features, though, mathematically. One is that it's seamless. So for it to be seamless, I need to assume convex faces.

So faces are the sides of the polyhedron. So like in a cube, every face is a square.

Those are convex. And provided all the faces are convex, if they're not, you have to cut them up into convex pieces. My folding will be seamless in that it will be covered by an entire piece of paper. There maybe other things hidden underneath.

But there won't be any visible seems on the top side. So that's nice. It's also water tight. And for this, I have an illustration. This is a feature missed by the strip method. And if you've always felt like the strip method of making anything is cheating, here's a nice formal sense in which it's cheating.

We didn't realize it until we start talking about Origamizer which does not have this cheating sense. So here I'm trying to make a 3D surface, looks like a saddle surface, by a strip. If I just visited the guys in this nice zigzag order, which I know is possible, I get all these slits down the sides.

This thing would not hold water. If you poured water on it, it would fall through all the cracks. And if I fold it right, like in this picture, there should be no seems in here. The square, the boundary of the squares is what's drawn in red. So here the boundary of your piece of paper gets mapped all over the place. So it's lots of holes.

Here, I want the boundary of the paper to be the same as the boundary of the surface. So the only place the water to run off is at the edge. I mean, obviously, this thing is not a closed solid. But if you actually made a cube, you're still going to get some edge because the boundary paper has to go somewhere.

But if you then sewed up the edge, it would totally hold water. So that is the informal version of watertight. The formal version is the boundary of the paper maps within some tiny distance epsilon of the boundary of the surface, boundary of the polyhedron.

And here, when I say polyhedron, I really means something that's topologically a disk. Brief topology. This is a disk. This is a disk. This is not a disk. Cube is not a disk. It's a sphere. This is a disk. A piece of paper is a disk. So really the only things you could fold topologically in a pure sense in a water tight sense are disks.

You can't glue things together. That's not in the rules. So I can't make-- I could fold this. But I'd have to have an extra seem somewhere in order to make this thing just be a disk. I could fold a cube. But I have to have some seem somewhere. Here-- the top-- a square gets cut into four pieces in order to make it into a disk.

Any higher topology can be cut down and made a disk. So this is still universal. But in terms of the water tightness, you have to think about the disk version.

AUDIENCE: Erik?

PROFESSOR: Yes.

AUDIENCE: Even when you go back and forth with the script method, you could argue that that was topologically a disk.

PROFESSOR: Oh, yeah. Anything, any folding you make is still topologically a disk. This is making a disk. But it's doesn't preserve the boundary.

AUDIENCE: It what?

PROFESSOR: It's not preserving the boundary. So yeah. Any origami, you're still disk like, and you're watertight for some disk surface. But I want to make this disk surface with that boundary. And watertightness is supposed to match the given boundary. But that boundary must form a disk. That's the point. I can't say, oh, there's no boundary in a cube. So you have-- so the boundary of paper goes nowhere. That's not allowed.

So you get to specify it, but it has to be a disk. I'm going to wave my hand some more. There's another feature which you can see in this picture. This is a schematic of what Origamizer would produce which is that there's some extra stuff underneath.

It's slightly lighter because it's on the bottom side there. But you can see along every edge, these are the edges of the actual polyhedron. And then, there's these extra little tabs, extra flaps on the underside. This is actually necessary. If you want watertightness, you can't fold exactly that polyhedron.

You fold a slightly thickened version. But you can keep all those flaps on one side. So if you're making something like a cube, you can put all the garbage on the inside where no one can see it. So there's another feature here is we get a little extra.

And that's necessary if you want watertightness. And it's sort of the trick that makes all of this possible and efficient and so on. So the high level idea of Origamizer is we're going to say, there's all these faces that we need to make. So just plop them down on the piece of paper somewhere. And then, fold away the excess. Get rid of it by tucking.

And that excess paper is going to get mapped into these little chunks here. And maybe I'll show you a demo. So you takes something like-- this is similar to the thing I was showing. And I forgot a mouse. There are all the faces in the plane.

And they've conveniently already been arranged. I can't zoom out because I lack scroll wheel. But there's a square that makes the-- yeah, or a multi-touch trackpad. Sorry. And all the faces are just there. And then there's this extra stuff. And now I say-- I should probably do this, too. Maybe not. Well, all right.

And then I say, crease pattern. Boom. That folds away the excess. And then, just the white faces, these guys which correspond faces over there, are used. And then, you just fold away the extra junk. Easy. You want to make a bunny. This is actually an example where it will not work by itself.

Because as I said, this algorithm is not quite guaranteed to work. So I'm going to change the boundary little bit by cutting to the ears. And so this is still-- it was a disk before. It's a disk after because there's this boundary here. But it turns out, now the algorithm will work, assuming I didn't mess up.

It's bouncing around a little bit. You can see it's pretty efficient here. There's very tiny gaps between the triangles. There's actually a little bit a violation here. What's happening, which you can see here, is, on the inside are all this extra structure, these flaps. And sometimes they're so big that they actually penetrate the surface.

But that can be dealt with by just a little bit of cutting, maybe a little more cutting. Not cutting in the literal sense. But we just subdivided these panels into lots of smaller panels. And now, it is valid. This not the design that you've seen on the poster. The design on poster's little more efficient.

I'm not so expert. I'm not so pro that I can make exactly that design. So it's a little inefficient on the sides here. But you can use this tool to make super complicated 3D models. Let me quickly tell you what goes into the algorithm. So the first part is to figure out where all these tucks are going to go.

They lie essentially along angular bisectors on one side of every edge. But at the vertices, things are super complicated. And in general, if you have a non-convex surface with tons of material coming together, what I'd like to do is add lots of little flaps on the side. So that when I open it up-- so let me draw you a generic picture.

So we have two faces coming together. What I'd like to do is add a flap here and a corresponding one just behind it. So that's sort of a tab. And I can unfold that and think of some other surface that I'm trying to fold. So I really wanted just those two polygons. But now, I've made some other thing which is still a disk. You can add those faces in such a way that you will have, at most, 360 degrees of material everywhere.

So even though in the original thing, you had maybe tons of material coming together which you cannot make with real paper, you add in a bunch of tabs along the edges and a few more at the vertices. You can fix things so that the thing is actually makeable from one sheet of paper. That's the high level idea. Doing that is a bit detailed. The paper isn't even published. These are some figures from it.

But this is some-- way this is how you resolve a vertex with some triangulation stuff. Each of these corresponds to a flap in the original thing. And then, this is where we're a little impractical. It doesn't quite match what we do in practice in the computer program. But the idea is, you imagine, you have your faces which you want to bring together somehow.

They're distributed in the piece of paper somewhere. But you'd really like to connect them together when they have matching edges. So this edge might be glued to some edge of some other triangle. And I need to just navigate these little river-like strips to visit one edge from the other. And we proved that if these guys are sufficiently tiny, you can always do that.

And of course, in reality, you want to arrange things, so you can do it efficiently with little wastage. But we proved at least it's possible with this kind of wiggly path stuff. So now our picture, the thing we're trying to make looks something like that. Where we have-- originally it was four triangles, the gray triangles. We added in these extra flaps so that it's nice and well-behaved.

And each of those flaps we're covering from both sides. And if you think the red diagram-- I'll get this right. The red diagram corresponds to that. It's just been kind of squashed. So there are four triangles, which correspond to the four top flaps. And there's this outer chunk which corresponds to that flap. And then, you look at the dual which is the blue diagram.

You take that picture, and that's how you set up the crease pattern essentially. So these are the original four triangles. And then. There's all this stuff that represents the structure of that thing that we want to make. And then, you just have to fill in the creases in the middle. And you do that just with something-- this is how you do it guaranteed correct. And we saw I had to do lots of subdivision here. What I called, accidentally, cutting.

But just lots of pleats there. Because, essentially, this is the edge. And we want that edge to lie in a tiny little tab. So it's got to go up and down and up and down and up and down, accordion style. And if you do it right, all those things will collapse down to a little tab attached to that edge which is also attach that edge. And they will get joined up. Then, you've got to get rid of all this stuff in the middle.

And rough idea is, if you pack it with-- or I guess you cover it with disks so that everything is again very tiny. And you fold what's called the [INAUDIBLE] with those points. And it works. It's a complicated but very cool. And the paper hopefully will be

released later this year finally. And that's Origamizer. And that's efficient origami design.