PROFESSOR: All right, let's get started. We are continuing our theme of folding polygons into convex polyhedra. Let's do a quick reminder, we're talking about gluing up the boundary of a polygon to itself.

And we were representing that with gluing trees last time. So I want to do an actual example of a gluing tree. So this is how you make a cube out of a cross, my favorite example.

And a valid gluing, it's an Alexandroff gluing because it has no crossing, so its topologically a sphere. Everything is glued to everything. And it never glues more than 360 degrees of material to any point. In fact, it's going to be 270 at every point, because that's the curvature of every vertex of a cube.

And the gluing tree was, if we sort of turned this inside out and think of the polygon as being on the outside instead of the inside, it's really hard to see here how to convert that into a gluing tree, though you can do it. It's the way that these arrows nest. But it's a little easier to think of on the sphere or on the polyhedron, because then turning inside out is just upside down. It's not a big deal.

So if we look at where the cut are on the cube, it looks something like this. That should be a cross. This is the top of the cross, this is the bottom of the cross. It goes around the back side.

So then the gluing tree is essentially this doubled. If we walk around the outside of the cuts, that is the gluing tree. So I can see, there's this segment, then this segment, then that segment. Then we're back here, then we go up one, two, and then we turn around.

That's a leaf, and so it's going to look like this weird h shape. And these are vertices. That's right, all of these should be vertices.

And locally you can see, ah, this is a 90 degree angle. This is actually the top of the cross, remember the polygon is now out here. That's going to be the top across.

This is where it turns, it gets a little confusing because of the way I've drawn it.

Here's the bottom of the cross. Those parts are easy to see. The rest is little awkward.

But for algorithms, this is a great way to think about things. And also for proving bounds and how many different ways to fold, it's really useful to think about gluing trees. Of course, all of these views are equivalent, but a lot of things will be easier to analyze here. So, clear?

Let me first tell you what we're going to prove today. We're going to do combinatorial bounds and algorithms for all of this. And let's see. So I want general gluings, general situation, edge-to-edge gluings, and a particular kind of polygon called bounded sharpness. And then I want two columns.

The first question is, how many gluings are there? And whenever we get a bound on the number of gluings, we're also going to get an enumeration algorithm. Remember, there are three goals.

The first goal was decision. Is there any folding? Second goal was enumeration, give me all the foldings. And third goal was counting, I think. How many different solutions are there? For [INAUDIBLE], this is mainly about decision and enumeration. And this number of gluings, that's the combinatorial problem.

So it turns out, for everything the we'll do, these two have the same answer. So I'm going to put it in one column. But sometimes for a decision, you can do it a little faster than enumerating them. In particular, because the number of gluings can be exponential.

We saw last time that it could actually be infinite. I'm going to fix that, don't worry. We'll make it finite. And then the right bound is exponential. There's upper and lower bounds of two to the theta n.

And also here, bounded sharpness is interesting because it's polynomial. And so this is also polynomial. The one result here that is not tight is this one. It's an open

question whether for the general setup, the most interesting scenario, just give you a polygon, is there any gluing?

The best algorithm we know is exponential, even to decide whether there's a gluing. So I put a star there to say that. It is not tight. The open problem is, can you do it in polynomial? I suspect yes, but we worked on it a long time ago and failed.

For the special case of edge-to-edge gluings, which is when you only glue whole edges of the polygons to other whole edges, there is a n to the order one algorithm. You can think of this as an edge-to-edge gluing if you imagine this as being two edges. If I actually draw a vertex there, then this is an edge-to-edge gluing. And then there's a polynomial algorithm to tell you, is there any folding?

If you want to list all of the gluings though, you need exponential time because there can be exponentially many. But everything except this one result is the best you could hope for. Bounded sharpness is a natural sense in which this exponential is kind of cheating. Let's see, let me tell you about bounded sharpness.

I want to bound how sharp an angle can be in a polygon. This is a really sharp angle. If every angle-- is that actually the sharpness I want? Actually, no. Sorry, I'm getting inside out already.

Actually, what I care about are reflex vertices. I don't want to have a super big angles here. So if every angle is at least 360 minus some epsilon-- for a constant epsilon-- then I call my polygon bounded sharpness. If your polygon has bounded sharpness, we get a polynomial bound on the number of gluings and a polynomial algorithm to list all the gluings.

This is really the sense in which this is all practical. Give me a polygon that's going to have some sharpest angle, call that the bound. If epsilon gets really tiny, if your reflex angles get really close to 360, then this bound will go up.

But as long as you're not too extreme, you're all set. I mean less than or equal to, yes, thank you. Yeah, that would be a weird polygon if all the angles were so big. I think that doesn't exist.

So for example, convex polygons always have bounded sharpness. Epsilon is 180 in that situation. So, that's good. I told you about edge-to-edge gluings. Just glue whole edges to whole edges.

What I haven't really told you is how we made this finite, so let me go to that. And when I say gluings here, I really mean a combinatorial type of gluing, because we had rolling belts. So there were infinitely many actual gluings, but I'm going to think of most of those as being the same thing and just distinguish essentially what the gluing tree looks like combinatorially. So let me define that.

The first part is what I call the abstract gluing tree. So this is just what the picture looks like in general. So for example, the one over there-- the cube-- combinatorially the tree looks like this. It has four leaves and a kind of wishbone configuration. So I'm not trying to measure the lengths or anything, just that's the abstract picture, no metric.

And then the second part is I'm going to specify which polygon vertices and edges are where. And I'm going to specify that at things called junctions. So in particular, at leaves I want to say, oh this is vertex one.

At places like this, where three edges come together, I want to specify for each one, oh this is vertex two, this is edge eight, this is vertex 10, whatever. I'm making up these numbers. They don't necessarily correspond, but they should appear in order because this is the polygon after all, just warped inside out.

Also this one. Also, I didn't really draw this picture big enough, but if I have two parts of the gluing tree where a vertex glues against something-- it could be another vertex or it could be an edge-- I also consider those junctions. So junctions are going to be all the vertices-- the only exception is when there's an edge gluing to an edge, I don't consider that a junction because there's going to be fairly many of those. I want there to be finite.

So everything of degree one, everything of degree three or more, for degree two there has to be at least one vertex. So degree not equal to 2, or at least one vertex.

Those are junctions. For each I'm going to mark where they are. That's part of my abstract gluing tree, specifying where the junctures are, how they're configured.

And then for each one, wherever there's part of the polygon coming together, I specify which vertex or edge is coming there. Now when I specify an edge, I don't say what point on the edge, because that could be infinite. There could be a whole range of points, there could be. We'll worry about that later. I just specify that edge eight is there.

So this is what I call combinatorial type of gluing. And in this case, there are only finitely many gluings, finitely many types, but it can be exponential. It's still big, just not quite as big.

So just to get warmed up a little, let's prove this exponential upper bound that is only 2 to the order n. That will in particular convince you that there's only finitely many of these, although I think that's probably obvious that it's finite. It's conceivable it could be more than exponential.

This is not a particularly exciting bound, but we're going to use a lot of the same ideas in the other proofs, in particular this one which is more interesting. I want to look at leaves of my tree. And we have to remember back to the previous lecture, who can be at the leaves? Well, if could be a vertex or it could be an edge.

When it was an edge, we called it a fold point. When we folded in the middle of an edge-- so here's an edge of the polygon and I end up subdividing it and gluing the edge onto itself-- right there is 180 degrees of curvature. Total amount of curvature is 720, therefore I only have-- I can't count, I'm jetlagged-- four of them. At most four fold points.

Everybody else is a vertex. So if I have n vertices, I have at most n plus 4 leaves. That's already feeling good. I pull a rabbit out of a hat and tell you that if I have only n plus 4 leaves, there are 2 to the order n trees on that many leaves.

Actually, I should be a little more careful. If I can bound the number of nodes in the

tree, then I'll get an exponential bound on the number of trees. And that will be part one, the abstract gluing tree. But right now I have a bound on the number of leaves, not the number of nodes.

And there's a difference here, because if you have a tree which has a whole bunch of degree 2 vertices, then that would be bad, because I only have two leaves I have arbitrarily many nodes. But every time I have one of these degree 2 nodes, we already threw away the degree 2 nodes that had no vertices at them, because there are infinitely many of those and those don't count. So once we throw that away, the only degree two junctions are the ones that have a vertex, which is why-- these dots are supposed to be the actual vertices of the polygon. So it might be an edge here, it might be an edge there.

There's at least one vertex at each of them. So the total number of nodes here is indeed order n. I mean, it's probably actually at most n plus 4 again.

But if we're sloppy you say, well the number of leaves is at most n plus 4. The number of degree 2 nodes is at most n. Then there could be some degree three nodes. It's also at most n plus 4 in total, or n is all I care about.

You could be more careful and figure out what the consonant is. But we get from this, this is order n nodes. And once you have that the tree has a linear number of nodes, I'll tell you this is one of the [? Kadhalan ?] problems, that there's only 2 the order n such trees.

But that's only part one of the combinatorial type of the gluing. We also have to worry about which vertices are where. I want to show that that is only exponential.

So imagine at this point you've fixed the combinatorial structure of the tree, but you have no idea what's what. So in particular, at these points-- and maybe also you've defined some degree 2 junctions, which you know are supposed to have at least one vertex-- the first thing I want to specify is, for each of these little dots that come together-- these are the junctions-- is it a vertex, or is it an edge? So that's just binary information for each one. I'll fill it in, say if it's a vertex, I'll leave it open if it's

an edge.

In fact, I know that at most one edge comes together at any point. Otherwise, you'd have too much material glued there. So maybe it looks something like this. It could be a fold point, maybe another fold point. Something like that.

So that coloring, black and white, there's only 2 to the order and such colorings. Call it a vertex edge coloring, because again there's order n dots. Each one, there's two choices, black or white, vertex or edge.

And now what we really care about is where the vertices, because remember, this is the polygon in order. So if I could tell you, let's say which vertex is vertex one? Maybe this vertex is vertex one. Then I know this is vertex two, I know this is vertex three, I know this is edge three or whichever edge connects V3 three and V4.

This is also edge three, and so on. I could just walk around and label them in order. All I needed to do is know where vertex one is. There's only n possibilities for that.

There's n filled circles, which are where the vertices are. One of them is V1, so there's only n choices for it. You multiply all of these things together, n times 2 to the order n, times 2 to the order n, that result is 2 to the some other order n. That's total number of choices.

So that's a rough but fine upper bound. I'm not going to try to tune this constant. I guess it would be an open problem to get a really good bound on the constant. I'm not sure if even one has been worked out, but what I will show you is that this is pretty much tight, so you can't hope to do much better than this because there's a 2 to the omega n lower bound on the number of gluings.

And that is this crazy example. I'll draw it also on the board. It's a little tricky to draw. It's a very spiky star, something like that.

So I have e points, and n spikes, n convex corners. The convex angle, I think, is alpha. Let me match the notation. Yes, convex angle is alpha.

The reflex angle here is beta, but otherwise it's completely symmetric. So all the

reflex angles are beta, all the convex angles are alpha. I have n spikes.

I want these points-- it's hard to draw, but I want them to be very, very close to the center. Take the limit. In the limit, alpha is 0. So alpha is going to be some very tiny amount epsilon.

Beta. What's the limit of beta? 180, 360? No, it's not 360. That's what's important.

360 times 1 minus 1 over n, yeah. It's like 360 minus 360 over n. Believe it or not, this is a big number, in that it doesn't depend on epsilon. So it's going to be, what, a little bit smaller-- going to be some minus epsilon prime.

But I really don't care about the epsilons. They're kind of irrelevant. Let me just cross them out, but they're really there.

So alpha is basically 0, beta is a chunk less than 360. What this means is in the limit I can take one of these alphas and glue it into one of the betas, and it will still be less than 360. In fact I could glue n of the alpha. I could glue all of these spikes into one of these gaps.

Remember, the limiting picture looks like this. That's maybe more convincing. It doesn't look that way here. This is nothing. Of course, I can fit arbitrarily many of them into this gap and it will still sum to less than 360.

So this is kind of the key to why this example works. I've got tons a room for alphas in the complement of betas. Now, I would like to make this not only an exponential lower bound on the general case, but also on the edge-to-edge case, because that's a little bit stronger to say even edge-to-edge gluings there can be exponentially many.

To do that, I'm going to take the midpoint of one of the edges, call that a vertex, and take the perimeter antipode, which is like here hopefully, more or less. So I want the left parameter and the right perimeter to be equal, also call that a vertex. And now one thing we know how to do is perimeter halving.

Now, we were only supposed to do perimeter halving on convex polygons, but it turns out it will work on this polygon too. So I'm going to glue this half edge to this half edge, then I'm going to glue this edge to this edge, then this one to this one, and so on. So just do that gluing.

And the only thing to worry about is that you glue two betas together. That would be more than 360. Anything else, beta to any number of alpha's is OK. At this point it is really helpful to draw a gluing tree, so let me do that.

So gluing tree for perimeter halving is path. And do I give these guys names, probably x and y? Yeah, that's what I would call them, so naturally that's what I called them.

We have x over here, we have y over here. And then we have the perimeter in between. So let's just check what happens. Here we get an alpha on one side-- I'm going to say the right side is the bottom, let's say. I'm not going to try to think about it.

So there's an alpha, and that meets a beta. And then it just alternates from there. There's an alpha, it meets a beta, but on the other side. And a beta meets an alpha and so on. At the end, it's probably the reverse, like alpha, beta. I've drawn it up here at the very top.

The alpha's red, betas are black so you can distinguish them. But that's clearly an OK gluing because it's just one alpha gluing to one beta. We know we can glue tons of alphas into one beta. That's one gluing.

The fun part is that there are exponentially many of these gluings which look like this, where you take some of the betas and turn them into leaves, kind of squeeze them out. And the result is that you get two alphas gluing to a beta, which is fine. The beta glues to itself, which is fine. I mean, it's a vertex.

It's comforting that it has an angle almost 360. Therefore, there's very little curvature out at that leaf, but we don't really need to check that. It is just necessary for this to work. Of course, beta is less than 360 by itself.

Here's an example where I squeezed two of the betas right next to each other, then three alpha's come together and glue to a beta. But what's nice is we don't have to think about the polygon. We just think about the gluing tree. And we say, well if I pull this down-- so I end up with that-- that's going to be just fine as long as I also pull one on the top.

I have to squeeze the same number of betas on the top and the bottom, otherwise these links won't match up. As long as I do that, I get a valid gluing. So what I'm going to do, I have n betas total. So I have n over 2 betas on the top and n over 2 betas on the bottom. I'm going to squeeze n over 4 on the top and n over 4 on the bottom, so half of them.

That way the lengths will match up, and I happen to know there's a lot of ways to do this. The actual number is n over 2 choose n over 4 for the top and the same for the bottom. So it's this squared, and I'm just going to tell you that is 2 to the theta n.

Actually, I think you can be pretty precise. Can you? I've forgotten.

Isn't n choose n over 2-- it's very close to 2 to the n over 2, I believe. Is that right? But not exactly, yeah.

All right, I'm going to be sloppy. It's 2 to the theta n. You could figure out what the constant is there. I think I have a bound written down, maybe. Oh, it is the power of 2, yeah. It is equal, I believe, according to my notes.

So this is going to be 2 to the n over 4 squared, which is 2 to the n over 2 exactly. Yeah?

AUDIENCE: [INAUDIBLE]?

PROFESSOR: I only am allowed to squeeze betas, not alphas. But I can do whatever pattern I want on the top and separately any pattern I want on the bottom. Because what's key is by squeezing betas I preserve the parity, because when I squeeze away a beta I bring two alphas together. So the parity and who's matching whom on the top

and the bottom is always the same. And the beta on the top will always be matching some number of alpha's on the bottom, and vice versa.

So it doesn't matter what pattern I do on the top, or what pattern I do on the bottom. It just matters they have the same length, so I arbitrarily chose them to both be n over 4. Well, it wasn't arbitrary, but I chose it to be that because I knew there would be a lot of those choices.

There are more, of course. Really I should sum this over all choices of n over 4, but this is good enough. That summation would not improve the bound by much.

So that's sort of the bad news. These are really nasty polygons. They have exponentially many gluings. For fun, we actually did this for small n. It's been a while. This was some time ago that we made all these. I think it actually was before 2002, but that was when the paper appeared.

So here is a four star, and these are I think all of the possible gluings. Maybe just some of them, it's been awhile, I could think about it. Not drawn as gluing trees here, but drawn as actual gulings on the polygon. Here instead of drawing the gluings on the outside, which is how they happen in the gluing three, I've drawn them on the inside, just because it's easier to draw.

And then we built them in the same way that I built one like I showed you last time, where we just took a polygon, started taping edges together, and these are the roughly taped versions. And then you guess where the crease signs are, and then you can draw the crease lines on here. And two of them are drawn sort of worked out where exactly the crease lines have to be, according to getting all the edge links to match.

And then you can make your polygons. That is n equals 4, then we did n equals 8. I guess we want n to be even here for convenience.

And there's more, although still not a huge number. I think these are the top sides and these are the bottom sides. And you can see, in some cases it was a little tricky to tell where the creases are, because they were almost flat or possibly actually

exactly flat. Like here they are flat.

But it's a fun exercise. You can, whenever you have these gluings, reconstruct the polyhedra. Of course now that we have algorithms for Alexandroff's theorem, we could try plugging these into the programs and they should give us exact 3D polyhedra, from which we could figure out where the crease lines are. It would be a fun, small, mini-project, I guess, to do that. This is all from ages ago.

At this point I want to move on to this result, which is the good news. This is sort of bad news that there are exponentially many different gluings. It's not so bad in that at least we can enumerate them in the same amount of time. We'll get to that later.

While we're in the combinatorial streak, I want to prove that there's only a polynomial number of combinatorially distinct gluings for bounded sharpness polygons. So of course, this star polygon does not have bounded sharpness. We set beta to be very, very close to 360.

It may be worth talking about this a little bit, because it wasn't exactly 360. It was bounded away from 360 by this value of 360 divided by n. Now I said that was a big number, because it was much, much bigger than alpha. Bigger than n times alpha even, because I can make alpha very close to 0.

But it is not what we call bounded sharpness. Bounded sharpness was 360 minus a constant-- I really shouldn't have called it epsilon because it's confusing. Let me call it gamma just got another Greek letter. Gamma is not arbitrarily small. It is a constant value, whereas over here it's not constant. It's a constant divided by n, that's sub-constant.

So this is not bounded sharpness, just to check. But if we have all the reflex angles, at most 360 minus an actual constant bigger than 0, then we can get a polynomial bound on the number of combinatorially distinct gluings. So let me show that to you.

Here's the cool thing about bounded sharpness. How many leaves can the gluing tree have? Each leaf is either an edge, like it's a fold point. In that case, the curvature is 180 degrees. Or it's a vertex.

If it's a vertex, the curvature at that point is 360 minus the angle at that point. 360 minus the angle, let's call it alpha. Now, we know that every angle alpha is at most 360 minus gamma. So if this is going to be at least 360 minus gamma, which is just gamma.

So every vertex, every leaf of the tree, has curvature at least gamma. Gamma is a constant. Total curvature is 720, exactly. Therefore, the number of leaves is at most 720 over gamma, which is a constant.

That's going to help. We don't have to worry about trees with n leaves. We only have to worry about trees with 20 leaves, some constant number of leaves. Again, as the bound on sharpness goes down, the number of leaves will go up in this inverse proportional way, but it's a constant.

So at least for part one, we're golden. I mean, the number of gluing trees, I said over here-- where is it? If I have n nodes in my tree, I'll only have 2 to the order n trees. So if I can reduce the number of nodes in my tree, I will reduce the number of trees correspondingly.

If there's only a constant number of nodes, there will be a constant number of trees. It doesn't matter that a constant gets exponentiated, it'll still be constant. So that's good news, except we bounded again the number of leaves, not the number of nodes. So we have to be a little bit careful, because in particular we can have degree 2 junctions, which are real junctions-- they might look like this where a vertex comes to an edge. There's got to be about n of those, because the vertices have to be somewhere.

We're saying there aren't very many at the leaves. And therefore, at the degree three and higher junctions, there can't be very many. Maybe I haven't said this, but if you have a tree with L leaves, you'll only have at most L internal nodes, branching nodes. But degree two nodes, there could be arbitrarily many.

So what we come to is, if I have L leaves in a tree, then I'll only get 2 to the order L

gluings. This is what we need to prove. It will tell us that in particular, if we have a constant number of leaves, as comes from the bounded sharpens case, I will get-- constant number of gluings? That's not right.

There's some polynomial on n times 2 to the order L. Ah, that's why. I should look at my notes occasionally. The right bound to n to the order L.

In fact, what I wrote is an open problem. I think maybe it's n to the order 1 times 2 to the order L. This is what you'd call a fixed parameter tractable bound, but no such upper bound is known. That might be fun to work on in the problem session. It might not be difficult, just at the time I didn't know those bounds were important.

So we still have to prove this, n to the order L. This is the part where my notes are wrong. Here's the idea, we have a small number of leaves, we're worried about these degree 2 junctions. But I claim, actually there isn't a lot going on.

Because one of the great things we can do, over here we said for every dot, was it a vertex or was it an edge? We can't afford that anymore, because there's n dots still. We can't afford 2 to the order n.

Where we can afford it is at the leaves, because there's only L of those, and at the branching nodes. So everything except these degree 2 problems we could specify. And really we could specify anything we want.

So yeah, we could specify, is it a vertex or is it an edge? But we could make life even easier-- hm, I have an idea for solving this open problem. Do I want to solve it right now? Yeah, let's do it.

Let me think for a second. I think that's going to work. OK, cool, cool. I needed to rewrite these notes anyway, so I might as well prove a stronger result.

All right, so let me draw a picture. There's branching nodes, and I surely draw one of these longer so I can say, oh there's some degree two junctions here. Those are annoying, ignore those. For everything else-- the leaves and the branching nodes-- I'm going to specify the same binary thing. Is it a vertex, or is it an edge? But now

there's only order L of them.

There's L leaves, so there's at most L minus 1 internal nodes. Each of them has some degree, but the total number of these dots will be order L if I ignore the degree 2. So that's 2 to the order L vertex edge colorings, just like before except now with L, of leaves and branching nodes.

Oh, I see the problem. Darn it. What I'd like to do is mimic the same proof and say, well choose where V1 is and then just label them around. There's a problem with that, though. Say V1 is here.

Where's V2? Well, maybe this is an edge, maybe this is an edge. This, I have no idea. Is it an edge? Is it a vertex?

I can't tell. At degree 2 junctions, I have a problem. That's kind of annoying.

To fix that-- and so this is still going to be open, alas. To fix that I need to-- I really want to know which dot is which vertex for reasons to be determined. But in particular, that is part of number two. So I'm just going to give up and say, well there's 2n different things that they could be, n vertices and edges.

Do I want to do it that way? Maybe n choose order L. I'm just going to write down wherever I have a filled dot-- I guess this one is not filled. Maybe this one's filled, filled. I just going to write down, what is the vertex number there?

This is maybe V3, V5, V20, V21, whatever, for each of the filled dots among the vertices that I can see, ignoring again the degree two junctions. There's order L of them. I just going to for each one pick out one of the labels. Again, the order is determined, so this is the right number of them.

And this is n to the order L at most. So this is where I'm being a bit wasteful. It would be interesting to try to get around that. This was actual vertex labelings.

So now, among the vertices I can see-- the degree three junctions, and the leaves, and higher degree junctions-- I know which vertices are glued there. I can also figure out which edges are glued there by similar labeling, although I don't think I

actually care. No, I probably care, because it's part of number 2. So I do have to also label the open circles which edge is there. But a similar bound holds.

What's left? What's left are the degree 2 vertices. I don't know anything about them. This is particularly tricky when I have some edge here.

But let's first think about the case where I have a vertex, add a leaf, and I have these degree two junctions. And I don't know which edge is here, which vertex is here. In this case, because everything's small, I might be able to figure out.

But here, where's the V3 to V4 transition? Could be here, or it could be here. It could be here. I can't afford to figure it out.

Fortunately, I don't have to figure it out. It's not so obvious. From this picture you can't see what's going on, but this came from a polygon. This is an actual vertex, this is an actual vertex. So this edge length is the total perimeter from V3 to V5 along the polygon. I know how long that is.

So also if that was going clockwise from V3 around the polygon-- actually, that's probably counter-clockwise in the polygon because everything's inside out. If I go the other way-- let's be relative-- from V3, I can measure out the same length. So in fact, I have some polygon-- maybe it looks like this, whatever.

I go from V3 to V5, and I measure out that length. And I go the other way from V3, and I know this is getting glued to that. So I know where the vertices are from the lengths. I can figure out exactly what this picture must be when I have-- in this case, I happen to maybe get a vertex going to another vertex. In general, probably get some vertex gluing to some edge, and maybe vice versa.

But if I can figure out exactly what that pattern is, I can figure out the label of these guys, because I know exactly what's happening there. So that's good. If it's a vertex to vertex edge in the gluing tree like this, I can figure out everything between that pair.

Is that enough? Almost. So if I have a leaf where this is a vertex, which I was usually

denoting by a filled circle, I have various degree 2 junctions which I ignore until I get to a degree three or higher junction. Now at this place, even just looking at this dot and this dot, I know at least one of them must be a vertex. Because I can't glue two edges together plus other stuff. I can glue at most one edge.

So maybe this one's an edge, like in the picture that we did before. But one of them has to be a vertex. And then I can figure out exactly what's happening there, because I have this vertex to vertex thing.

And I know the labels. I know this is V5 and this is V13, whatever. I know exactly what's happening on the right side, I measure backwards, I know exactly what's happening on the left side and who's going to what. So that determines everything.

The problem is when I have an edge here, because then I don't know where it is along the edge. I can't measure lengths anymore, somewhere in the middle of the edge. And indeed, if I have something like this, where all of these angles-- I mean this for example is in a convex polygon, this could be a perimeter halving-- and it's not determined what's happening here because this is a rolling belt. I'll show you how undetermined it is.

Suppose you have a bunch of vertices like this, and then some vertices like this. Depending on where I bend around for E5, these guys could be here. They could be here, or here, or here. They could be in the middle straddling this.

In fact, there are about n squared different places where these guys could be, relative to these guys. Because there's going to be n transitions from-- if this is n and this is n-- n transitions from this versus that guy, then n transitions from these versus that guy, n transitions versus that guy, total of about n squared. Actually if this is n and this is n, it's exactly n squared.

So we're not quite done, but I claim that there's really only order n squared left. It's always from the rolling belts. We know from last time there's at most three rolling belts, so at most quadratic per belt. So total outcome is polynomial for what's left, what we haven't specified, for how the degree 2 vertices behave.

These are all degree 2 junctions. It's here, or they're here, we don't know. So if you want to get the full specification of what's where, you do all this work. And then for the degree two junctions, you have to deal with the rolling belts.

The way to prove this is say, well, if I have a leaf that's a vertex, I can get rid of it and induct. In the end, I'll have at most four leaves because there's at most four fold points. So I have a constant number of leaves. You look at each belt, each of them is quadratic and polynomial.

If you work out the bounds here-- we've tried to be a little more precise-- you could prove an overall bound of n to the 2L minus 2 for L leaves. And we have slightly better bounds for L equals 4 and L equals 3, but that's pretty good. But the proof I gave is at least n to the order L. We could n to the order L count different things here, and then some N to the constant at the end.

This ends the combinatorial part. Now we're going to get to algorithms. So we've proved all of these things. Exponential upper and lower bounds for the general case and edge-to-edge case, and polynomial for bounded sharpness, or when you have a reasonable number of leaves in your tree.

I'm going to start with this edge-to-edge gluing algorithm. This is going to use a technique called dynamic programming, which some of you have seen before, some of you haven't. It's a very simple idea. You take your problem that you want to solve-- which is I have a polygon-- and you split it up into sub-problems.

You solve each of the sub-problems, and one of the sub-problems is actually the whole problem, and then you're done. In our case, a sub-problem is going to be some sub-chain from one vertex to another vertex of the polygon. So I have some polygon, and I just look at some interval of the boundary from VI to VJ.

And the idea is, well suppose that VI is glued to VJ. How many different ways, or how should I glue the part in between? How should I glue this into itself? Because there's no crossings, there's going to be no gluings from inside to outside.

So that's the sub-problem. And of course if I say from the interval from V1 to V1,

that is the entire problem. Of course V1 is glued to V1. So in particular, the V1 to V1 sub-problem is what we want to solve.

So this is our goal. So I'm going to tell you how to solve all of these sub-problems, and therefore how to solve what we actually want to solve. That's dynamic programming. So let's try to solve one of these sub-problems, VI to VJ. How are we going to solve it?

So I'm going to draw a picture. It looks like this. This is now thinking about the gluing tree, so the polygon's on the outside here. That's one of the main reasons gluing trees were developed, just to describe this algorithm. This algorthim's actually quite old, 1996 by [INAUDIBLE] and [INAUDIBLE].

The edge-to-edge case was before I joined this group. And then later together we did the general case, unbounded sharpness, which is what we'll do next. So this is at the heart of everything.

So we have VO to VJ glued together. We want to know how to glue together the rest. Well, maybe some vertices get glued in here in addition to VI and VJ, maybe not. Let's think about what gets glued to this edge.

There's the edge from VI to VI plus 1. And this is an edge-to-edge gluing, so what gets glued there is an actual edge. What edge could it be? Some edge over here, I don't know which one. The power of the dynamic programming is you don't need to know.

It's some edge, VK, VK plus 1 gets glued there. There's only one of them, it's out there somewhere. It has to have matching edge length, but there could be many choices, all of which have matching edge length. Maybe all the edge lengths are the same, but there's at most n choices for that edge. It's actually at most J minus I choices plus 1. But there's at most n edges in here, so there's at most n different possibilities. Just try them all. Think about all of them.

I haven't said here whether I'm doing-- or I have? I guess I've been pointing here. I

really want to do both of these algorithms at once, but I'll start with this one, which is just find all the gluings in exponential time, which is optimal in the worst case because there can be exponentially many outputs. So try all n choices for VK VK plus 1.

What happens when I do one of them? I might choose one of them. Well, the new picture will be-- so the topological picture is I still have VI glued to VJ.

Then I also have here VK and VI plus 1 glued together. And I have VI also glued to VK plus 1. Now it's possible actually VK plus 1 and VJ are the same thing, in which case there's no loop here.

And that's the case when no extra vertex got glued in there. But this is the generic situation. If this is link 0, then these guys are the same.

So really what I have now are two different sub-problems. There's this thing, which is VI plus 1, dot, dot, VK. And there's this one which is VK plus 1 dot, dot, VJ. The idea with the sub-problems is to solve the sub-problems in order by increasing chain length.

So start with very short ones. And I saw longer and longer ones using the previous results. If I've already computed the answers to VK plus 1 to VJ because that is smaller than the original-- look, there's two edges fewer at least. This thing is at least 2 smaller than the original chain because here's two edges.

So I wanted to solve VI to VJ. I guessed what VK to VK plus 1 was, meaning I tried all the possibilities for each one. I say, well what are all the possibilities for this? What are all the possibilities for this? Take the cross-product.

Just multiply those sets together. Those are all the possible gluings. And then I sum that. I take all the options for all the different values of K.

And that's it. That output is all the possible gluings. So this should actually give another way to prove that the number of different gluings is only exponential, but this is just for the edge-to-edge case. Maybe I should write that down.

So we have for each choice of K output-- let's say, and for each solution to the smaller sub-problem VK plus 1 to VJ. And then for each solution to VK plus 1 to VK output. This is very simple. This is just a product over various things. There's only order n choices here.

This thing, it depends how many solutions there are to those sub-problems, but it's the product of the two. And then that's the total output. Believe me, this runs in exponential time. That's not very hard to prove.

What's interesting is we can make the same algorithm with a little bit of tweaking get a polynomial time decision algorithm. That's cool. How do we make it a polynomial decision algorithm?

I can't afford to store every solution for every sub-problem because there can be exponentially many. So I only get to store one solution for every sub-problem. So this is the great idea. Let's say for each sub-problem, let's say VI to VK only store the solution that minimizes the amount of stuff glued into VI VJ.

So the sum of the angles glued at VI, which is the same thing as VJ because they are already glued together. That's what we were told. So I had many choices here.

I had all the choices for K. I had all the choices for up here. I had all the choices for up here.

Well actually now I'm just going to think about one choice up here, which is what is the choice that minimizes the angle glued here? And what is the choice over here that minimizes the angle glued there? Why the minimum? Because some of these are wrong.

I should actually say output if it's Alexandroff. We only want to output Alexandroff gluings. We only want to output gluings where the total sum of the angles is less than or equal to 360. I need to check that for every one.

If I want to have the most chance of it being Alexandroff, if I got the least possible angle glued here and glued here, that's my best hope of it working. Now I still have

flexibility, which is I get to choose k. There's still n choices for k, but everything else is determined at that point. Once I fix K I see, well does this work, yes or no?

If it works, that gives me one candidate for how much material gets glued here. It could be a lot, could be a little. I try for all K, I take the smallest one. So it's the same loop, except there's no four loops here. It's just for every choice of K, I look at the solution to VK plus 1 to VJ, the solution for VI plus 1 to VK.

If it's Alexandroff, I don't output it. But I check, is it the best solution so far in terms of the total angle glued of VI and VJ. I keep looping until I find the very best one, and I output that one. That would be the solution that I store.

The result is, the running time is actually polynomial, because the number of sub-problems-- a sub-problem is determined by two vertices. There's only n of these, n of these. So the number of sub-problems is n squared. For each sub-problem I need to look at order n choices for K. And so the total running time is n squared times n, which is n cubed.

And that will tell you if there's any hope of anything working, if there's any gluing this will find one, because it's always trying to minimize the angles that get glued together That's is edge-to-edge. Now we can generalize. It's a little bit messy, but we can generalize to arbitrary gluings.

Let me sketch a little bit how that goes. So we've seen now this result for algorithms and this result for algorithms. And now I'm going to tell you about this result. I wish there was a result here to tell you about, but there isn't.

So this last part that I did where the decision algorithm is very efficient, we don't know how to generalize that to the general case, arbitrary gluings. This only works for edge-to-edge. We assumed a whole edge, VK VK plus 1 was glued to VI VI plus 1.

But the enumeration part where we said, oh just give me all the solutions, that we can get to work. It's just slightly more complicated. Let me tell you how it gets more complicated.

So one difference is that there are more kinds of sub-problems, this is the heart of the problem. We have the VI to VJ sub-problem just like before, that's easy. And then we have the VI to EJ sub-problem.

So this one was VI was glued to VJ, very clear what that means. Here we have VI is glued to an edge EJ. The trouble with that is we don't know where on the edge it's glued. It's somewhere on the edge.

This is where things get trickier. And there's also the symmetric case, of course, where it's EI to VJ. But we don't really need to worry about that, it's symmetric. So what does a solution look like?

So before, a solution was an entire gluing. Now, because the gluings can be infinite. There can be infinitely many different gluings. I can't actually list every one explicitly. I've actually got to explain ranges of gluings if I have any hope of capturing all of them.

So what a solution is going to look like is the combinatorial type of the gluing, which was this stuff. You have an abstract gluing tree, you get which vertices glue to what edges. So you have pretty much the whole picture, just not exactly where each vertex is glued to which edge.

Then I also will tell you a total angle of material glued at VI so far. In both these cases, I want to know how much stuff in my gluing at VI so I can tell whether it's Alexandroff. And then in the case where I glue a vertex to an edge, what I'm going to do is give you an interval.

The Interval's always going to be starting at 0, and going to some value x. x is less than or equal to the length of the edge, VJ. And this is the interval along the edge where VI can glue. So I'm going to tell you an actual interval here from the edge EJ.

As long as you glue VI to somewhere between there and there, this stuff works. Whatever I specified works over there. You can't go beyond some point. That will turn out to be enough.

So while there's the VI to VJ case, it's less interesting. Let's think about now if I want to actually solve the VI to EJ case. What could happen?

Before there was only one-- we looked at what glues VI to VI plus 1 and it was one edge. It was a little easier to think about. Let's not think about the whole edge, but let's think about locally what happens just after VI. Something gets glued there.

Now it could be EJ actually gets glued there. It could be we continue gluing. That's kind of like zipping, although not exactly what we were calling zipping before. Here I call it zipping, good choice.

So one option is that you zip for a little while. Now, how long would you zip? A couple options. You zip until you hit a vertex, you can't stop in the middle of two edges.

So maybe you hit a vertex here first, maybe you hit a vertex here first, and the next vertex here is farther away. So there's two pictures. Let me draw the two pictures. It could be hit you hit VI plus 1 first.

So here's EJ. It could be you hit the vertex of EJ first. In this case, you get a sub-problem which is VI plus 1 to EJ. These are glued to each other. In this case, you get this vertex, which I think is VJ gets glued to the edge EI, which is just the symmetric case I was talking about here.

Edge EI is glued to Vertex VJ. So those are smaller sub-problems. You solve those, and then from that you will see whether this actually gives you a solution to the original sub-problem you wanted to solve. That's the zip case.

There's another case which we call tug. In the tug case-- so that was supposing, we'll maybe nothing else gets glued in here. But maybe another vertex gets glued in there, that's the tug. To pick a vertex, I pull it in.

So in that case we get a picture like this. So here's EJ, here's VI, and now there's some other vertex VK that gets glued in there. Which vertex? I don't know, try them all.

Just like before, before we were choosing an edge that gets glued in. Now it's not necessarily a whole edge, but if there's any vertex that glues in, pick who's the next one. And now I get a sub-problem here which is VI to VK. I get a sub-problem here which is VK to EJ.

I solve each of those, I take the cross-product, I do that for all values of K just like I did here, except the labels have changed a little bit, and I find all the outputs. And again, you can show that it's exponential time at most. So that's an algorithm.

Now, it's exponential time, but really it's not so bad because it's at most cubic like we had before. But we're doing extra work, which is something like the number of gluings dealt with at each stage per sub-problem. Now, it's not just the number of actual solutions, because we generate all possible solutions and then check whether they're Alexandroff.

So if you have some example that happens to be unique in the way that it glues up, this bound will not actually be very good. It's still going to consider lots of options. But in the case of bounded sharpness, where the number of gluings is only polynomial-- and it's polynomial for a strong reason that the trees can't get very complicated-- then we can actually turn this into a polynomial algorithm.

This will be n cubed times that bound. As long as your polygon is bounded sharpness, you can show that at every stage of course the number of gluings will also be polynomial. And so then even the number of gluings you have to consider and check will be polynomial. So the whole thing will be polynomial time, that is this result and this result.

So if your polygon is vaguely reasonable, this is actually a really good algorithm. This algorithm has been implemented by two groups, one [INAUDIBLE] in mathematica and another [INAUDIBLE], which sadly is not-- his implementation used to be on the web, I can't find it anymore. But you get either implementation, plug-in your polygon, and it just lists all the possible gluings.

So that's all the theory for today. Now I want to show you a whole bunch of gluings for fun. Once you have all the stuff, we have these programs, you get to play.

So our first play is not that. That was the last play. First thing, we saw this before. I'll just remind you what it looks like and tell you a little bit about what's going on here.

So this was the cube. And then it's eventually going to fold into a cross. The real study here is for the Latin Cross-- which is the picture I keep drawing at the beginning of lecture and which will now be unfolded-- what other convex polyhedra can you glue that polygon into? So it's like you're not told what the creases are. You just want to look at gluings, the resulting creases.

This is one convex polyhedron you can make. This is something that can come from Alexandroff's theorem that you actually get a flat, doubly-covered convex polygon. It's technically a polyhedron, according to Alexandroff. These are all edge-to-edge gluings if the length two edges are actually subdivided into two length one edges. And this video only enumerates the edge-to-edge gluings, because at this point we only had the edge-to-edge algorithm.

I forget, this video is '98, I think. This algorithm was in '96, so they had just implemented that. And then that's when I joined. I was just starting out as a grad student and said, OK let's make a video. That would be cool.

So I think there's one more. And this is the only one where we actually have the animations. This octahedron, this was tricky to find, beceause at this point there were no algorithms for Alexandroff's theorem. So we found it by taking a piece of cardboard, gluing it up, getting a protractor, and measuring all the dihedral angles, and then typing them in. And Lo and behold, in that animation they don't perfectly close up, but it's very close, up to the measurement error of the protractor.

So for edge-to-edge gluings, there are exactly five convex polyhedra you can make from that cross. But if you allow non-edge-to-edge gluings-- and so some years later when we came up with that algorithm and implemented it, we discovered there are 85 gluings of the cross. This is only six of them. We're going to do it starting with

the cube, and then we're going to go in increasing order of faces.

So there's two flat, doubly-covered quadrilaterals. They're different. Then there is a bunch of tetrahedra that you can make. None of them are regular. One of them was edge-to-edge, I think it might be this one. I think there's some more tetrahedra at the top there. The blue labels here say what they are.

For each one of these we taped up the-- we knew what the gluing was. We taped it up, and then we played with it until we figured out, this must be the creases. Then we actually drew in that create pattern, folded it up, and verified that is the right answer. Because again, at this point no algorithm for Alexandroff's theorem.

More tetrahedra. Here we get to the five-sided polyhedra. I think this one is edge-to-edge and is in the video. It's a little hard to tell from the-- actually, you can tell from the pictures. Here there's two sevens. That means seven glued to seven.

That means this is a fold point, this edge is glued to that edge. Over here, every edge has only one label, where again this is considered two edges. So this was the edge-to-edge gluing. I think a couple more slides.

We get another pentahedron and then starting to get hexahedra other than the cube, lots of them. Then we start getting octahedra. The last one in the video was octahedron. It's not this one, this one has some fold points. It must be-- these are all octahedra, so it's one of these.

This guy. Of course, it's hard to tell from a static image, but yeah, obviously that was the octahedron from the video. And that is all. So actually, there's only 23 distinct polyhedra you can make, but you can make them out of 85 gluings.

So 85 gluings. If one of them is symmetric, the cube is symmetric. So that's one. And then the other 84, there's everything and its reflectional inverse, because this polygon is symmetric. So it's 84 divided by 2, which is 42, a good number. 42 actual honest gluings plus the cube.

But those 42 gluings, a lot of them generate the same polyhedron, because there's

only 23 distinct polyhedra in the end. I don't have it here, it's on my web page. You could look up all 85, and it says which ones are the same as which ones. And it was a real surprise when we were building them.

It was like, wait this looks identical to this other polyhedron, and it's glued in a very different way. Sort of an extrasymmetry of that polyhedron, as opposed to the cross itself. It was pretty cool.

An interesting thing to note here is that we only bisect the edges, right? There's a lot of edge-to-edge stuff, and then some of the edges get cut in half. And in this case actually, they look mostly like fold points, but that's not always the case.

It turns out, if you have a polygon were all the edge links are the same like this and you look at non-edge-to-edge gluings, and there are no rolling belts-- because if there's rolling belts, it's infinite because you're not going to have anything like this. Then it suffices to subdivide every edge in half. And then every non-edge-to-edge gluing will become an edge-to-edge gluing.

This is called the [INAUDIBLE] half-length theorem in our book. It was proved by this guy [INAUDIBLE], who was one of the implementers of the algorithms for enumerating all these gluings. And he proved it afterward. The code doesn't actually exploit that fact, but it's kind of neat. All you have to worry about are half edges in this case where all the edge links are the same.

All right, that's a cross. And it's kind of maybe even surprising that there's only finitely many different gluings, but there's no way to get a rolling belt here. Let's look at-- oh, here are all the polyhedra in a pretty 3D form. Here is a polygon which is convex. It is the equilateral triangle.

And so, no surprise, there a rolling belts here. But here's the picture of how all the rolling belts fit together. In principle, you can extract this picture from this algorithm. Although combined with an algorithm for Alexandroff's theorem, that's the tricky part.

So you have some very simple gluings, like could could fold the triangle in half, get

this doubly-covered triangle, you can make a regular tetrahedron, or you can fold the triangle into a flat doubly-covered rectangle like at the top. And then there are rolling belts in between. So this is one rolling belt, this is another possible rolling belt, another possible rolling belt each with a gluing tree. They're not all drawn here.

And this is sort of the topology of all of the possible gluings you could make, which is nice. So a triangle is relatively simple. The square is a little messier. There's again a bunch of particularly nice gluings like the [INAUDIBLE], the folding in half one way, folding in half the other way, making a letter, and some tetrahedra that are just drawn here. They're in the middle.

And then there's these crazy rolling belts for how they fit together. So this is in unit square all the different gluings. And one of the fun things, some of them are octahedra, I have a list here. But I don't know actually which one of these belts has the octahedra. It might be a couple of them.

And it turns out, for octahedra they actually figured out an algorithm just for octahedra to solve Alexandroff's theorem. This is again before Alexandroff theorem algorithms. And they wanted to compute, well if I take all these polyhdra, which one has the maximum volume?

So if I take a square paper, what's the maximum volume shape you can make? And there it is. It's kind of fun. One of those octahedra.

And this is a prettier picture of-- so here they actually computed all the polyhedra. Obviously not all infinitely many of them, but various snapshots along each rolling belt. And each of these rolling belts corresponds to one of the circles over here.

And that's it. Any questions? So that's the end of fun with gluings. We'll look at other kinds of gluing problems next class.