PROFESSOR: So this lecture was about hyperbolic paraboloids, and the extent to which they don't exist or exist. Here is a regular non-existing hyperbolic paraboloid with the concentric squares, no diagonals, folded here. And so, those are just a few questions about this. What does it mean, other things. These are all asked by [INAUDIBLE] I believe, and they're all open, so-- they're all good questions.

We don't know whether the good triangulation, the alternating one, works for arbitrary, and we only know up to n equals 100, for the various fixed angles that we checked. We don't have a proof technique to do arbitrary in. You could try to do some amount of alternation, but not, somewhere in between the two extremes, and probably you'll get something in between goodness, but I don't know. You could certainly play with that.

And it's very natural to try this with larger than squares. The only trouble is, the center no longer has a single degree of freedom, so the only thing to answer there would be, How do you want to initially fold, like, if you're doing a hexagon, hexagon is probably clear what you might want to do, but for general k-gon, how do you want to arrange that innermost k-gon? From that, you could propagate out, just like we did with the square, if you use an alternating triangulation, say. Probably works well, but we haven't tried it. That could be a cool project to do that, not that hard. So those were open problems.

Next, we have some, sort of, math, general math questions. c1, c2, and semi-creases. These are related questions. So let me do a quick visual review of these terms. I'm going to do it for one-dimensional curves in 2D, because that's a lot easier to think about, then what we really care about, which is two-dimensional services in 3D. But all the ideas carry over, so here is just a function.

Let's say, for example, this is a parabolic arc, than a straight segment. This point is missing, and instead it's up here. These points are present, and here we have another parabolic arc. So, this function I would call piecewise C infinity. Let me

explain what these things mean. So we have, on the one hand, c zero is a term, meaning just continuous. Continuous means no jumps like this, so this is not a continuous function. So this function overall is not even c zero. c1 is a stronger condition, which means that not only are you continuous, but you also have a continuous and existing first derivative.

So whereas here we're thinking about f, here we're thinking of f prime. So, not only can you talk about the function being continuous, c0c1, but you can talk about individual moments in time. So for example, this moment here is going to be c1. It has a nice derivative. That derivative is changing continuously here. Here, however, this is c zero, but not c1, because the tangent on the left is different from the tangent on the right. So this is not a c1 function, because the derivative is jumping at that point.

I'm not going to draw the derivative here, it's a little harder to draw, but you could draw it with the same x-axis, and you'd see a jump from one angle to another. This point is not even c zero. This point is c zero, but not c1. This point is c1, because the tangent here is equal, on the left, is equal to the tangent on the right, if I drew this properly, where this is the bottom of the parabola.

You can ask for more than c1, and we do in class, we talk about c2, and this says you should have a continuous second derivative. So acceleration. And this point, for example, is not c2. Because while the tangents meet, if you take the next derivative, you see that it's suddenly-- here the tangent was completely flat, it has a second derivative of zero, and then suddenly it starts increasing.

Now you can design functions that are even C infinity. C infinity means no matter how many derivatives you take, it's continuous and defined, no jumps. But if you do a parabola, for example, it will not be so well-behaved, I think, yes. Got the x, y equals x squared, you take the derivative, you get 2x, take the derivative of that, you get 2, and then the derivative of that, you get zero. So the second derivative here is 2, all along the curve. So I guess y prime, y double prime. So, in particular here, we had a second derivative of zero, here we have a second derivative of 2.

All right, so that's a quick crash course on individual points being c zero or c1 or c2, or C infinity is when you can go all the way. Creases are points on the paper where you're not c1. That means that you have two different tangent plans coming together, typically. So discontinuity is in the first derivative, that's what we call creases. Semi-creases are discontinuities in the second derivative

So semi-crease is where you're not c2, crease is where you're not c1, according to this particular paper. Semi-crease is not too common a term, but crease is very comm-- this is the usual meaning of crease, semi-crease is maybe a little new to this paper we're talking about, nonexistence of pi parse. And so, for example, this would be a crease, and this would be a semi-crease. Here the derivative changes, that looks like a crease, if you had a one-dimensional piece of paper. This doesn't look like a crease, it's kind of smooth, but not-- it's a little bit smooth, it's c1, but it's not c2. So there's a semi-crease there. And part of that paper is sort of worrying about semi-creases, because we want to deal with c2 parts, because c2 functions are nice. I mean, ideally we have C infinity parts, but we only need c2 parts, and so we subdivide at the semi-creases. And basically, argue most the time semi-creases don't really happen, so you don't have to worry about them. But that's what they are.

Any questions about smoothness? These are all different kinds of smoothness. Yeah.

AUDIENCE: [INAUDIBLE].

PROFESSOR: All of these things are C infinity, except at these points. So every polynomial C infinity, you can differentiate it all the time, eventually get to zero. Exponentials are C infinity, pretty much all functions you can think of that are smooth are C infinity. Yeah.

All right, so next question is about one of the proofs, which was that-- the proof involving normals basically, the one involving tangent planes, and the other one is involving normals. This is the polygonal implies flat proof, so this was, If you have a region of paper that is bounded by polygonal creases, so they're piecewise straight,

then that region of paper, if it's uncreased, must actually be flat. And I'm just going to, I want to talk about just one part of the proof.

So there was a sequence of steps, but ultimately what we wanted was some segment, bf. This is a boundary of your region, somewhere over here, and we assume that it's straight. So, we took a few steps to get here. But suppose we have a straight region and locally, around this point, there's some ruling. And then we're looking at a point, Q, here. I'm going to define this ruling to be r of q. The rule line from q, and so q is an arbitrary point, it slides along this segment.

And then we're interested in a normal vector here, which we called n of q. That's normal to the surface at this point, and so as q slides, n of q changes. And we were claiming a few things, but so the definition of this guy being normal is it has to be perpendicular to-- normal to the surface is perpendicular to the surface. It has to be perpendicular to this axis of the surface, and it has to be perpendicular to this axis. So we know, just from definition of normal, n of q is perpendicular to bf, and n of q is perpendicular to r of q, the rule line.

So that's the definition but it wasn't quite what we wanted. We wanted to prove, for example, that the derivative of the normal, funny thing, is perpendicular to bf. That was one of the things we wanted, and the other one was the corresponding perpendicular to the rule line. I don't know if I used this notation in lecture, but this just means that they're perpendicular, which is like the dot product of those two vectors is zero.

So, how do we go from here to here? This was actually the part that was most confusing to me. I find this one more confusing, because it uses the fact that it's [INAUDIBLE], but the question was actually about this one, so unless you ask about this one now, I'll just cover this one. It's a little simpler. Both are in the notes.

So why is this true? I guess, first, the intuitive reason. So, you've got all these different normals here. We're going to end up concluding they're all pointed the same way but, in general, they're changing direction somehow. We know that at all times the normal is perpendicular to this segment, so it is pointing away, straight

away from the segments. So you've got a right angle here. So it's kind of spinning around the segment at best, as it moves around.

We claim that the change in the normal, the derivative of the normal, as you walk along this segment, must also be perpendicular to the segment. Intuitively, this is obvious, if you think about it enough. If you were changing in a direction that was not perpendicular to normal, to this segment, for example you're going this way or something, then, while initially you're perpendicular, you're kind of falling over, and then you'll no longer be perpendicular. So if you change in a first order way that is not perpendicular to this thing, then afterwards you will no longer be perpendicular. So that's intuitively why this follows from this.

One way to formalize that is to use Taylor expansion, so I'll just write that quickly. If you look at a point, q, and I'm going to use this notation, q plus epsilon, to mean a point just a little bit over. This is q plus epsilon, this is q. For a very small epsilon, this is going to be approximately n of q, plus epsilon, and the prime of q. In Taylor expansion, then you have epsilon squared, but for small enough epsilon, this is a good approximation. And so, on the one hand, you have-- If you look at n of q plus epsilon, and you compare it to bf, you know that these are perpendicular, and what you can write that as a product being equal to zero. So the dot product of these two vectors should be equal to zero, that is, being perpendicular. We know this is true for all points q, so in particular, it's going to be true for q plus epsilon.

So that's what we know, but now we can expand this thing, and we get these two terms. So we get n of q dot bf, plus epsilon and prime of q, dot bf. That's just distributing that sum over this product, dot products work OK that way. Now this thing is also zero, because this is just another point q dotted with bf, and they should always be perpendicular. And this is the thing we care about. Now if this whole sum is equal to zero, and this term is equal to zero, then this better be equal to zero. So implies this thing equals zero, and that's what we wanted. And prime of q is perpendicular to bf. So that's the algebra way to see it. But if you think about it enough, and you believe things are linear to the first order, as we say, than that has to be true. You can get second derivatives and things as well, but first derivative

better hold in, particular.

Any questions about that? You can do the same thing with this property, but it's messier. You need to use the [INAUDIBLE] property, which we didn't really go into much detail on, so I think I'll skip it.

Last question is, what does it all mean? If we're proving things are impossible, and yet here I have physical models of them existing, what's going on? What's the difference between mathematical paper and real paper? And, of course, there's no real mathematical answer to that question, but there-- we have a couple competing theories for what might be happening in this model.

One is that perhaps the paper is not being isometric. Perhaps it's stretching or shearing. So normally we think of paper as unstretchable and not shearable, the only way to shear is to fold it. And that is almost perfectly true but, of course, no material is perfect. So it might be shearing or changing the geometry, the intrinsic geometry just enough to somehow make this possible.

Our theory only says that it's impossible if you're exactly perfect, so maybe-- it would be interesting to measure how exact paper is. Question?

AUDIENCE: If it's possible to imitate a shear, if it compresses the paper but I fold it, then--

PROFESSOR: This is not a shear.

AUDIENCE: I know, but if you can fold it, so that it, like, imitates a shear--

PROFESSOR: Well, you can add creases to make it, to simulate a shear.

AUDIENCE: Right.

PROFESSOR: But only by adding creases, that's what we proved in lecture. So yeah, you can kind of fake a shear, but only by adding creases.

Now, it doesn't look like there's extra creases here. Theory number 2 is there are lots of really tiny creases here that are so small. I mean, here you could detect the

crease, because it was a big change in angle, but if they're super, super tiny here and, you know, we never fold these things completely perfect, it could be lots of little tiny things, maybe you look under a microscope and you'll see that, I don't know. It could be an interesting project to try to investigate what really happens with paper in these kinds of models in real life, and how it differs from mathematics.

But the two theories are maybe some stretching or maybe extra creases. We know by adding the diagonals, it folds. Now here, you could pretty clearly see the diagonals are not being added, but there might be-- there might be effectively, there might be enough added creases that are together so tiny, you can't see them.

AUDIENCE: It's not semi-creases--

PROFESSOR: Semi-creases are not enough. The theory says that if you have these creases and possibly any number of semi-creases, you can't fold it all. So you need to add actual creases. Yeah, good question.

All right, that's it for old material. Now I wanted to show you some cool new material, which is based on--

AUDIENCE: [INAUDIBLE].

PROFESSOR: Yeah, question?

AUDIENCE: [INAUDIBLE] don't really look like straight lines.

PROFESSOR: Right. The creases here do not look like straight lines, and yet we prove they must be straight lines, so something's going on. Now, we didn't know initially that straight creases had to stay straight in 3D, but once we proved that, we were pretty sure this didn't exist. It's not obvious that straight creases stay straight, but they do. It's evidence there's a problem.

OK, I wanted to talk about this paper, because it's kind of related. It's about how to efficiently make pleat folding happen, and it's written by a bunch of people, Jean Cardinal, Marty, our cameraman, Shinji Imahori, Yoshi Ito, Messashi Kiomi, Stefan Langerman, Ryuhei Uehara, who's here on sabbatical, and Tachiachi Uno. Good

practice.

So, here's the kind of setting. It's going to be familiar, in that we have a one-D piece of paper, so it's a segment, and we're also going to assume that it's uniformly creased, so each of these is spacing of 1. And now usually we think of preassigning mountains and valleys here, and then you could only fold mountains on the mountains, you can only fold valleys in the valleys. The model here is that you're allowed to fold and unfold repeatedly on the same segments. So you can do some layers, simple folds, although, I think, we'll just need all layers here today. But you can also unfold any previous folds.

And I'll say that unfolds are free, you don't pay for the unfolding operations, because it's just a constant factor. You can only unfold things that have previously been folded. So we'll count the number of folds you make. Each time you do a fold, it's a mountain or a valley, say, through all the layers and through some of the layers. But our goal is in, the end, to have our segment creased in a particular pattern. And the pleat pattern would be mv, mv, mv. So we want the last time each crease to be folded, to be a particular pattern, but along the way we could fold it in lots of-- it could be, this one could be mountain for a while, as long as the last time we crease this thing, it's a valley, we're happy.

So we end up with mountain valley strings, and we want to know how many folds, how many folding operations do you need to make, in order to achieve a particular mountain valley string? And, in particular, we care about strings like mv mv mv, because, as we'll learn later today when we fold these, pleating takes a lot of time, it's tedious. It would be nice if you could do things faster.

Now, why do we think we can do things faster? Because I can take a piece of paper and, imagine this is one-dimensional, and fold it in half, and then fold it in half again. Boom, I got how many creases? Two creases for the price of one operation. Now I fold again, Boom! I get four creases for the price of one. I fold it again. Wow, I get eight creases for the price of one.

So the number of folding operations you do here may be much, much smaller than

n. n is going to be the number of the creases, the length of your string.

So if I fold a piece of paper like this-- I actually tried this-- and then I unfold to see what mountain valley assignment I got-- unfolding is free-- I get this weird shape. And I didn't do it perfectly. It's hard to do perfectly because of creep. I get a weird thing. Anyone know what this thing is called? I think that's pretty much-- if I unfold them to 90 degrees.

AUDIENCE: Dragon curve.

PROFESSOR: Dragon curve. Yeah. Dragon curve looks like this. This is the Wikipedia drawing, and the more you fold in half, that you keep adding this iteration, it's a nice fractal. It's kind of cool, it doesn't have-- you can prove it doesn't self intersect. It touches itself at vertices, but it doesn't properly cross itself. You take the limit, it looks something like this if you fill it in, very cool. Wikipedia calls it the, I don't want to spoil the surprise, but there's actually a book where the section headings are iterations of this fractal? Anyone know what this book is?

AUDIENCE: Jurassic Park.

PROFESSOR: Jurassic Park, yes, Ian Malcolm right there. So, pretty cool. It's called the Jurassic Park fractal, to some, at least on Wikipedia. And so that's one particular mountain valley assignment you can get. And let me tell you what is. It's pretty easy to figure out, and you see why it's a fractal. So the first time we make a fold-- let's say I always make mountains, just for simplicity. You have a little bit of choice here, but not very much.

So first I make a mountain fold, so now I've got things folded in half. Now let's say I make a mountain fold over here. Of course, it's already folded in half, which means I get a valley over here. So I'm going to-- Yeah. So now I make another mountain fold, maybe I should have done it the other way, anyway, so it is. So I make another mountain fold here, and on the left I get this, on the right I get the reflection of that. And then I make another mountain fold, and I get this on the left, and I get the reflection of that on the right. And when I reflect, I'm reading backwards and also

inverting everything. vv and so on. So you see the kind of fractal nature here, and that's what gives the cool curve. Keep repeating that.

You have, essentially, n choices, because they're log n folds that you make, each could be a mountain or valley. So there are n different mv strings you can get in essentially log n folds. But sadly, none of them is the one we want, mvmvmv, the pleat fold. So that leaves us with the question of how many operations do you need to get a pleat fold?

This is where things get cool. So-- why don't I tell you some answers. So if you want to fold mmmm, or mvmv, these turn out to be the same, because you can do m's on the odd positions, v's on the even positions. Then you can do it in log squared n folds, and you need at least log squared n over log, log n. This is way, way faster than n. There's an open problem between log squared n and log squared over log log, small gap of the log log factor. But somewhere in between here.

If you want to do, let's say, a random pattern, then we can prove you need about n divided by log n. So the obvious way to do it is fold each crease unfolds, that takes n steps. So you can always improve by a log factor, and that's the best you could do for most strings. But these strings are sufficiently special, that you can get a huge factor, basically exponential, when, instead of doing roughly n steps, you're only doing poly log steps.

So, there's four results here. Which ones would you like to see? How many people vote for log squared n upper bound? Four. How many people vote for log squared over log log lower bound? Completely different set of four. How many people vote for the n over log n story, upper bound first and lower bound? OK. I think the upper bound here wins, just curious. They all these similar ideas. I'm going to briefly sketch, because I want to get to folding stuff, how you achieve n over log n upper bound, maybe then I'll do log squared upper bound here. Lower bounds are, well, they're bounds. That doesn't say anything.

So, if you have an mv string, to get this bound, arbitrary string, I'm going to consider epsilon-- sorry, 1 minus epsilon log n, consecutive letters, and split into, basically, n

over log n chunks, each of size roughly log n, a little bit smaller than log n. Why do I do that? Because the number of different chunk values is 2 to the 1 minus epsilon, times log n, right, each can be mountain or valley.

Now 2 the log n is n, so this is n to the 1 minus epsilon, possible different chunk values. These are chunks. Why is that interesting? Because I have basically n over log n chunks, but there's only n to the 1 minus epsilon different chunk values. This is much, much smaller than this. Think of epsilon being a half, as the square root of n, this is n over log n. So there's actually most-- many the chunks have to be repeated.

So this means that an average chunk is repeated, What is it? It's n over log n of them, is n over log n, times n to the 1 minus epsilon times, which is n to the epsilon over log n. This is the number of chunks here, and we're dividing by the number of different ones, so typical repetition is going to be n to the epsilon over log n times. I'll just assume all chunks are repeated this many times. It's all linear, so it doesn't matter whether some are more common and some are less common.

OK, cool. Now what? So, if I look at one of these chunks and all of it is repetitions, I would like to fold them all somehow more efficiently. And there's one key idea in this paper on how to do that, which is-- so here's my strip, maybe here's one instance of the chunk, here's another one. They can be kind of spaced out arbitrarily. So what I'm going to do is fold here, fold here, fold here. I think that's right, all the bisectors. So I end up with one chunk like this, than another chunk like this, than another chunk something like this. Then, this one is very tiny. Can I get my four repetitions of the chunk all on top of each other?

What I then do, so this is going to take how many repetitions is, n to the epsilon over log n, so this is about n to the epsilon over log n steps, to do all these folds. Now I fold here, fold here, fold here, fold here simultaneously, getting them all correct. That's going to take 1 minus epsilon times basically log n steps, then I unfold. And then I discover, Darn it, half of them were upside down. Because I got this one right and this one right, but these two are flipped, so that's so good, so then I recurse on the remaining half. But this turns out to be a geometric series, so if I do all four of

these, and then I do two of them, and then I do one of them, I'll be done, and it will only cost me another factor of 2 in time.

So I get log n plus n to the epsilon over log n. I have to do this for each of the different chunk values, so I need to do n to the 1 minus epsilon, times log n plus n to the epsilon over log n, times 2 because of the geometric series. And this comes out to n over log n. Ta-da! Magic.

The log n is basically coming from the chunk size being log n and 1 minus epsilon log n as big as we could make it, because we needed this to be a lot smaller than n. So that's how you save a factor of log n, little crazy, but it works in theory.

That's the upper bound, the lower bound is actually pretty obvious, because if you only use k folds, you're doing-- there's n to the k or maybe 2n to the k, different things you can do, because there's n different places you could make a fold, could be a mountain or a valley. And this better be, at least, 2 to the n, because they're 2 the n different mountain valley patterns you could make, and you have to somehow make them with k folds. These are all the things you could make with k folds. And you work that out and k has to be, at least, about n over log n.

So that's why it's optimal, and because this is an information theoretic argument, this works in the average case as well. Take a random example. You need at least n over log n. Most examples, high probability, will need n over log n. So that's those lower bounds.

Let me briefly tell you about this log squared n upper bound. It uses the same idea, but because everything is repeated in the mmmm string, you don't-- it's a lot easier to do this kind of folding. Oh sorry, there's one step I left out here. So great, you make these folds, you line things up, you fold these things. Then you unfold. So in recurse, eventually you fix these guys, but you also destroyed this crease and this crease and this crease, so you kind of messed things up a little. You've got to do this repeatedly for every chunk. You don't want to mess up previous chunks that you've done. So you have to go back and fix this fold, unfold, fix this fold, unfold, but that, again, it only costs n to the epsilon over log n, so not too bad. So another

factor of 2 here. But I'm ignoring constant factors.

OK, that was n over log n, upper bound. Let's do log squared upper bound for mmmmm. So I need to look this up, it's a little tricky. We're going to use the trick of the dragon curve, essentially, which is repeatedly fold in half, and we're going to keep doing that until we are left with three creases, which haven't yet been folded. So this is my folded bundle. There's many, many, many layers here. And then I'm going to fold mmm on those three creases.

Instead of going all the way to dragon curve, which would just be, Do m, fold here first, I'm going to fold this, unfold, fold this, unfold, fold this, unfold, so I get this pattern. When I unfold this, it actually turns out to be kind of nice and alternating. It will be something mmm, something, valley valley valley, something, mountain mountain mountain, something, valley valley valley, and so on.

OK great, half of my things, roughly, are mountains. I just need to fix these valleys. So I use the same trick, which is I'm going to fold, let me use red maybe-- fold in the middle of these mountain segments. If I do that, I line up all the valley segments. So I end up with, after folding, this is going to look like, you almost need a computer to work all this out, but we have implemented this, it works. It's going to look like this, I mean, with many layers like this. All the values are on top of each other. These question marks are still question marks. This mountain is that mountain, this mountain is that mountain, my mountain is your mountain, I don't know.

So now we're going to fold mountain mountain mountain mountain, on all five of these things, and these two remain mountains. And then unfold, and what I end up with is twice as good. I end up with question mark, and then, I think, sorry, not quite-- because this part also gets messed up. So I end up with valley, mountain, question mark, then seven mountains, then question mark, then some not so pretty stuff, m v v v v m, question mark, and then this repeats until we get to the end of the string. And then the end looks kind of like this.

So the point is, I had three m's in a row, now I have seven m's in a row. Each time I do this, I roughly double the number of m's in a row. By the end, almost everything

will be m's, after I do this log n times. I have pretty much all m's and then I can finish it off. And each of these steps took log n steps, because I had to fold along all these things. How did I fold along them? Not one at a time, of course, I fold in half and fold in half, and fold in half, roughly. I choose the middlemost red line to fold first, and about log n steps later, everything will be piled up. So I get to use this efficiency of dragon curve things. I'm messing up these creases, of course, but it's OK, because I get a big chunk that all gets correct at once. Half of it gets wrong, half of it's correct, but the half that's getting correct, gets longer and longer each step, doubling. So only log squared steps.

Pretty sure this is optimal, but the best lower bound we can prove is log squared over log log.

All right, finally, let's fold some stuff. I thought we would fold some hyperbolic paraboloids and maybe put them together into structures like this. So, you have your squares. If you don't have squares, a bunch of squares up here. You can start folding as many as you like. We're going to put them together. I will demonstrate up here.

So the first thing we want to do is fold along the diagonals. These aren't perfect squares, but it's OK. So we fold along on diagonal, and the other diagonal. This is mainly so we know where the center is, but also because we need to fold a diagonal, so we kill two birds with one stone. Whereas we just saw how to kill, in this case, log n birds with one stone. Say log n, here we're only killing two.

So now you've got your diagonals and your square, we want to fold-- it's kind of cool, I colored the edges with my chalk hands. Now everything's going to be parallel to the sides of the square. So the first thing we're going to do is construct a square that's half as big, centered along that point. So this involves folding the bottom edge, usually do this valley fold. Fold the bottom edge to line up with that center, and also line up the edges. Now, very important. When you make this fold, don't fold it all the way, just fold the middle half, between the two diagonals. That's really key, otherwise this will not work. It's the only thing you have to be careful of. Don't fall all

the way, just in the middle.

OK, that's one quarter. Do that four times. So we go here, line up at the center, fold the middle. Once you've got the inner square, you're going to make two more. There's going to be this square at the one quarter mark, and an inner square at the 3/4 mark. To make the outer square, you fold the edge to the thing that you just folded. Keep careful note of which crease you're folding to, should be from the previous square, not from the current square. To fold the inner square, you go all the way up to the opposite side of the first square.

And you can just look visually, are these evenly spaced. If yes, you did it right, if not, you're in trouble. You're probably OK if you make a few extra creases, but don't try to make as few extra creases as possible. So then repeat that four times and you'll get your, I guess, four squares, if you count the outer boundary.

It's a little hard to see, once you've got that, those squares, looks like I did one of them incorrectly. Got your nice concentric squares, four of them. Now you're halfway down, you flip it over, and do valley folds on the other side, because you want these to be alternating mountain valley, mountain valley, and just fill in all the squares in between those squares. And you can figure out what the reference markers are, they're all the mountain folds, actually every other mountain fold will be one of your references.

Don't go to one of your new valley folds, always go to the mountain ones. And just check that you're always filling in halfway in between two of your mountain folds. Don't forget to flip over, very important. And skip every other guy, otherwise you will make a refold and just increase the wrong way. This is what it looks like after one quarter of those.

And once you've done all this crazy pre-creasing, concentric squares, alternating mountain valley, then the fun part begins. This is actually, literally fun. It's, I guess, also harder, but much more exciting than all that pre-creasing. So, in this case, you have to fold all the creases at once, and the easy way to do that is start at the outside and make the outermost ring, fold it, just go around once or twice. Make

sure everything's folded, including that diagonal. So here I have valley folded everything, including the outer diagonal.

And then you want to mountain fold the next one, and get them to be against each other. You're aiming for a kind of x shape, which is drawn over here. It's going to look like that ideally. Just keep collapsing, square by square, making sure you alternating mountain valley, it should just fall into place pretty much. But paper likes to misfold a little bit, so you've got to fix it all. Here I've done two squares, little more. It gets easier and easier, because the squares get smaller and smaller. We've got three squares done, so I'm, like, halfway to an x.

Fourth square. The center is a little bit tricky, try to get it so it alternates, so now it's nice and thin. It's like repeated sync folds of a water bomb base, and it's like an x. And then this is your opportunity to recrease all your creases really hard, just give it a good squeeze on each of the three, four legs of the starfish. And now you've got your hyperbolic paraboloid. To make it, you want to take two midpoints here of the squares, pull them apart until it's a little bit open, and then give it a twist, and then open it up a little, and you've got your hyperbolic paraboloid.

If we make enough of these, we can assemble it into some cool shapes. How many people have folded one? At least one. A few people. I will show you-- you can fold more. We're going to assemble something, might need a bunch of people.

This is the algorithm we use for converting a polyhedron into a bunch of hyperbolic paraboloids. It's in this paper for '99, and we take each face of the polygons, of the polyhedron, sorry. So if you have a cube, you've got a bunch of squares. For each square, we will make what we call a four hat, which is four hyperbolic paraboloids, joined in a cycle, like this. And you've got to be careful the join it the right way, but then, these, the tips of the high parts that are not joined-- I mean, one tip of each of the high parts comes together at the center. Than these tips are going to represent the edge of the polygon, these red dots.

And so these two sides are the sides of-- these two sides are the sides of [INAUDIBLE]. They're going to join to an adjacent [INAUDIBLE] over here. So that's

the idea. So I've got-- I've already folded a bunch of these already. I'll show you, maybe, what a hat looks like.

So take two of them, join them along those edges. We're going to use tape or staples to join them, I don't have a fancy lock, I'm afraid. And join these edges together, and you get-- this is a three-hat and you can put on your head, whatever, and that would represent a triangle. And then we're going to join along these two sides to an adjacent triangle or whatever.

So we could make a platonic solid. I guess the simplest one would be a tetrahedron has six edges, so it needs 12 [INAUDIBLE]. I was thinking we could make this solid, it's never been made before. It should be cool. It's the simplest Archimedean solid, in terms of number of edges, the truncated tetrahedron. This requires 36 parts, so if we're ambitious, we can go for it, but I think we're low on time. So a tetrahedron might be an easier bet.

If people want to come up and start assembling. Finish taping that. Hold these, and then this goes here. Do you want to hold this one? Virtual assembly here, so maybe we'll finish assembling next class.