COLIN MCCRONE: Good afternoon, guys. My name is Colin McCrone. I'm with Safdie Architects in Boston. This had two speakers today, who was also one of the principals from Safdie. And he was planning to come to AU until the last minute. So it's just me today.

I've been the Director of Design Technology at Safdie Architects for about the last seven months now, I think. I'm still pretty new. And before that, I worked on the Dynamo team at Autodesk. And so my job used to be to go around and teach people Dynamo.

This class was inspired by the fact that over the two and a half years I worked at Autodesk, I had questions all the time on how do you handle services with Dynamo. And there really weren't very good resources on that.

And traditionally, when there are very complex problems to solve, of course, people tend to turn to panelling tools in Rhino or Grasshopper. And the equivalents don't necessarily exist in Dynamo. At least not in the same packaged format. But it doesn't mean that the software isn't capable of it. In fact, quite the opposite.

And with access to really cool Safdie projects, what I did for this class is to take some of the logic that was part of making the Marina Bay Sands, and also Project Jewel, which I'll show you-- I'll give the background on this-- and remade them in Dynamo as a teaching example.

I have set up this class so that I want to work from something that's a little simpler, that's more comparable to the type of paneling tools that already exist in Revit, and then to move into something that's progressively more complex, that reflects real design intent issues.

And I should also note that when the projects were originally done, it was a series of something like 20 different Grasshopper files, all stitched together, that did this. And it was work of many hands. So the work I'm showing you today are not the files that created the buildings themselves. But it represents similar logic. I remade the logic in some places where I thought it was clearer now that the design intent is done.

In reality, of course, all of these designs went through many more iterations than I'll show you here. But I really want this to be a back and forth, an instructional example.

I'm going to have to-- because I wanted to show you lots of cool stuff-- or I hope it's cool-- I wanted to make sure that there was enough to really talk about and show. But if at any point

there are questions that you have about, what is that node? Why are you doing that?

Please ask me. It'd be great. I know we're right at the post-lunch hour. You guys, everyone is going to sort of crash in 45 minutes, right? I'll try not to. And it's certainly a small enough class that we can have that back and forth. So please feel free to ask me any questions.

All right. So just the class summary, just to reiterate. This is the only thing I'll read to you.

But with examples from two projects-- one built and one under construction-- see how Dynamo is used with Revit as part of a design process to rationalize and document a complex facade with flat panels. This instructional demo intends to show how design constraints might be expressed through scripting, and, ultimately, through architecture.

OK. So there are four things that I really want to get across. The first is to compare a situation that you could normally solve with some fairly simple out-of-the-box Revit tools. And we'll work on that with the equivalent is in Dynamo. And then we'll make it more complex in the next section, when we talk about various different edge conditions and constraints.

Then how to control-- our strategy is basically for how to control multiple Revit family instances. Because that's really the goal, if you're going to do this in Dynamo in the first place. Of course, you're probably trying to get it into Revit. And that's the kind of interaction you'd like.

And then I also want to talk about, at least how at Safdie Architects, with projects like this, I'll add a few things about how we do documentation.

So the order of the day will be to go just real briefly through the projects, paneling Revit, paneling in Dynamo, complex surfaces, placing families, and then talking about all the good data you can get out of it then.

So, the projects. There are two projects that we'll be talking about. One is Marina Bay Sands, which is, of course, belongs to the same organization as the Venetian in Singapore. And this is-- I believe it was completed five, six years ago? Something like that?

And it's a casino. It's in downtown. There are three hotel towers. And notably, on top, the SkyPark, which, at the time, was debatably the world's longest cantilever. I think the CCTV building in Beijing took it over a few months later.

But the particular paneling exercise I want to look at is not the glass side-- that's easy-- but the underside of the SkyPark. So this is in-- it's an arc. I think at one point it was an elliptical arc. And then it was a circular arc. And now it's something in between. So it's a complex sweep section.

And then you have a variable cross-section. Through the middle part, it's mostly the same. But then you have sort of a nose at one end along the cantilever, and then you have a different condition at the end.

Because I want to step up the complexity of what we're looking at, for this particular example, I'm going to focus on the majority part of the problem, which is the middle 85% of the surface, specifically because you can represent it as kind of a bent rectangle. There's the same number of panels across and on both edges. That kind of thing.

And it's something that Revit can actually deal with pretty well. Revit can't necessarily deal so well with parameterizing surfaces that are swept to a point. I'm not even sure you can do that in Revit. But this is again an example to look to that kind of thing.

The second project is called Jewel Changi Airport. And also in Singapore. So we're very Singapore-centric today. And it's obviously the thing that's lit here. It's a multi-use facility and will be the new centerpiece of the airport.

And the airport at Changi-- they are, for at least the last four or five years, are consistently ranked the top airport in the world. And they are very serious about expanding their amenities and things like that.

This Project Jewel actually sits in the middle of the airport, next to this control tower, in this ridiculous site. It's in the middle of everything. But nobody ever built there because it's so complex. They wanted-- there was a parking lot, there's a train that goes through the middle of the site that they didn't want to move, and it had only a certain particular height.

And the concept for this particular project is that it's basically a gigantic glass donut-- or bagel, or something like that-- with an oculus in the middle. It's elliptical in plan. And it went through a couple different form finding exercises. Certainly the structural engineers at [INAUDIBLE] [? at that ?] time were also a big part of that. And it's also, of course, asymmetrical, because the train literally runs through the center of the building.

And the oculus is open. Or at least it is the origin of a gigantic indoor waterfall. It's 40 meters

tall. It will be the largest indoor waterfall in the world. And it didn't make sense to have that fall on top of the train. Because then people would get wet. So it falls into this five story tall forest valley.

So this one particular surface, the glass shell grade here, is interesting for several reasons. It's because it's not a rectangular surface. It's asymmetrical. And I'll talk more about the design constraints when we get there. But also the fact that you had to have particular levels that were plainer, things like that.

And then you had different types of panels that had to do different things. And we're all familiar with that kind of thing, too. Some might be vents. In this case, many were solid. Some need to be glazed. Some had different types of glazing, depending on what the program was underneath. And that all is certainly part of a computational design process. So that will be the second project that we'll look at.

OK. So this is the design surface that we're going to look at for the first section. So it's, again, basically the majority condition underneath Marina Bay Sands here. The geometry for these things-- I'm not generating the geometry. I'm going to take that as a given because it's really all just about paneling. In this case, I just stuck it in an SAT file and read it into Dynamo. And that kind of thing.

But I want to talk about how Revit does divided its surfaces. In particular, what I want to look at is this is a section. And if you can make out the detail that the panels on the underside, here, they wrap around. They kind of become part of the handrail.

And then you have this very-- it's extraordinarily evenly spaced. I mean, these rows here are the same width all as it goes along. So that as you geometrically-- the cord lengths are equal. And that's actually a bit of a challenge in Revit. Or at least with the divided surface tools.

So for example, this is from Zach Kron's blog, because it's a great image of how a divided surface works in Revit. And also the paneling tools in Rhino Grasshopper, too. And some of the LunchBox plug-ins. It's a very common way to divide surfaces, which is something that's called by parameter space.

So parameter space would be that you have these sort of evenish divisions along what are called isocurves, that sort of go through. And what the nice thing is, is that it looks kind of cool. You have these panels that get larger or smaller based on what happens with the surface.

But then one of the issues is you can end up with a result like that, where you have-- when all those panels are flattened out, that they are many, many different sizes. Now, in reality, that happened for Project Jewel. There are only two panels on that entire thing that are the same.

But at least for Marina Bay Sands, that was part of the reducing cost, was making sure that the panels were as similarly sized as possible. And I believe the number is 85% were of some normal type. And then the rest were sort of special cases, when you had the pinching, or that kind of thing, going on. So this is what happens if you normally do a divided surface in Revit.

And here's the issue with that. So here is in a one dimensional sense-- I showed you a surface-- here's how parameter spaces works. And Revit, by the way, calls this a normalized curve parameter. And Dynamo is-- I basically drew two curves and used the node called point at parameter. value.

And there are 60 points on each of these curves. And they're evenly spaced in parameter space. And what that means is I'll take the circular arc on the left and the start point is, say, at this bottom. Let's say the end point is up there. And this is parameter zero. That's parameter one. And then everything in the middle.

So if I have an evenly spaced-- 0.5 is exactly in the middle. 0.25 is exactly one quarter. And everything's perfectly evenly spaced. And that works really well when you have well-behaved curves, like circular arcs and lines.

If you don't have that-- say you have a spline, or some other type of-- or even an elliptical arc, parameters tend to bunch up where the curve is derivative is highest. Or where it's kinkiest, basically. So in actual space, the points are very far apart. But as far as the mathematical definition, the curve, that's what's evenly spaced.

The problem is we want paneling that looks like this and not like that.

OK. So I want to talk about-- I'm going to go backwards and forwards with Dynamo, and then also diagram. So my goal is to talk about the logic behind this. And then translate that into what that looks like in Dynamo, and to step through it.

But please, again, if you have any questions, absolutely let me know.

This is the strategy that I took to do something to approach the problem. Here's the design

surface, looking down from the top. It's cupped like this. And we know that the parameter space is not evenly spaced, certainly along this direction.

But it's easiest to find curves to work through, like when you're extracting a curve from a surface to specify a particular parameter. Or, basically, it's called an isocurve. What that means is it's a two dimensional analog of when I said point on a curve before.

If I go up to this slide here, this particular point is at parameter zero. And then, in this case, when I'm looking at a line in the surface, the whole first line is at the first zero parameter in one dimension of the surface. And so it's easy to find these.

So I have these cross ribs, basically, underneath the ship type thing. And those are curves that we can extract from the surface using this very easy parametric space type thing. Just a single node.

And then from those, once you have curves, then we can talk about making sure that they're evenly spaced. So paying attention to, say, how wide the panels must be each row. Then it would be a-- we're translating the problem into something that is evenly spaced points along a curve. And that's super easy to do.

From there, we sort of stitch these other curves together. And that will set the rows of the panels. So think of these as construction lines that helped us get these evenly spaced curves.

Next what we'll do is-- that's not quite enough if you're going to place a panel in Revit. Say, if we're thinking something like an adaptive component. Those are placed by-- the panel doesn't know what else is next to it. It's literally [? paced ?] with like, three, four points, or something like that.

Then what we're going to do is we're going to go through, and then we're going to reorganize all of this data about these evenly spaced points right here into groups of four, because each one corresponds to a particular panel.

All right. So I'm going to go that far in Dynamo. Any questions so far? Yes.

AUDIENCE: How did you get the isocurve to be uniform? Or are you going to get to that now?

COLIN MCCRONE: Yep. I'll actually try and show you in Dynamo. And then if I haven't answered fully or answered your question, please ask me again.

So I'll be going back and forth to this PowerPoint here. All right.

So, by the way, I totally forgot to ask this. Who's used Dynamo before? OK. Good. Who has not used Dynamo before? OK.

And so I do want to say that, again, if there is anything that's totally foreign, please let me know. I can also answer some questions quickly. There's no such thing as a stupid question, especially when, you know, I've never seen this before.

OK. So I'm going to zoom out for a second and show you what this graph looks like altogether. It's actually, for what this does, this is not a terribly large graph. I mean, a lot of graphs that I make, even in an afternoon, can be five times this size. But it's not necessarily about the complexity. And I tried to make this as simple as possible.

Also, there are things that are basically grouped. Just like I thought would be, when I was showing the diagrams-- even just from the zoomed out perspective. I said first we're going to do this, then we're going to do this. That this, that thought, is basically like a box. And then inside is the how do you get there.

I generally organize things-- so all the inputs are on the left. And that would be things like panel size, the design surface. And then things on the right would be the actual individual panels. And then in the pink over there-- we'll get back to that, but that's doing cool stuff with data. So I'll be starting at the left and moving right.

And I want to talk about what the inputs are first. So the inputs are first-- reading right here-- is the first thing is I have this SAT file, which I'm reading. And I know that because I made the SAT file, that there are certain things in certain orders. And, therefore, that this particular-- I'm going to run the file here-- that the, if I break this up, that the first design surface, or the first set of design surfaces here, is what we're actually interested in paneling. So it's this guy.

If you're not familiar with this notation, what this looks like is we're getting a file. And this is the file right here. This is now getting the geometry from that file. And I can see I have a list of surfaces, solids, things like that.

These guys, these code blocks, are breaking up that list into pieces. So, for example, I'm calling this list data, just because I can. And I'm asking for item zero through four, which I happen to know are the design surface. And the rest of them I'm sort of pushing down to something that's going to display it and make it look cool.

So this whole gray box here, this display the context, is just taking the rest of the surfaces, giving it a different kind of color, and leaving it there. That's just for visualization.

So the first thing we're interested in is this design surface. First thing. And when we extracted item number two from that list, then we have a simple surface. That white box, if you haven't seen it before, is what's called a watch bubble. And that's like a stethoscope, and looking at your data at that particular point. So it's important to know if it's a list, if it's a single thing. And then, as I'll be talking about quite a bit in a bit here, is that the structure of your data means a lot.

OK. So this is the design surface. When you see a node that is dark colored, like this, it means that it's the geometry that it's containing. If it contains anything, it's being previewed. And if it's sort of a duller color, that means it's kind of turned off. So I'm going to uncheck the preview on this design surface here.

All right. So the first issue, the first thought, is to get these lengthwise guide curves. And so the first thing we need to do is get those isocurves in the cross-wise direction. And then use that, with evenly spaced points, to remake the longer guide curves. And so we're going to do that like this.

One other point if you haven't seen this, is that you'll see these nodes with dotted lines around them. That's the way I've set up the file in order to talk through it. It's a freeze functionality. So, basically, I've actually frozen that node, which means anything downstream of it won't operate. So I'm going to turn these on as I go.

OK. So this guy, I'm going to unfreeze this. And this is in a manual run mode. I'm going to change it to automatic so that it's running continuously, so we'll start to see what's there.

All right. So stepping through. First thing. This node called Surface.Getisoline is exactly the single node that does this. So just like there's a curved up point at parameter that will find a particular point on a curve, you can also find a particular line on a surface. And the way this works is this is asking-- it's giving a surface, and then something called an isodirection that just means, basically, you can think of it as being Y or X, or U or V, in a plane of the surface. So it's 0, 1.

And then finally this list that I-- there's a class going on next door, or over here, called

Dynamo, You Don't Need Any Code Blocks. And I totally understand that reflex. But I love code blocks. And they're so fast sometimes.

So, for example, this is a specific Dynamo syntax. And there are really only three of these. And they're incredibly useful because everything about visual programming is about lists.

So if you're going to make a list of numbers, because what I want is, in this case, what this is doing is it's finding 12 things between zero and one that are evenly spaced. There's a help file about this in Dynamo, too, under Settings-- or, sorry, under Help, and then Samples, and then Core and then Core Range Syntax, because that's really easy to find.

But they really are very useful to know. Kind of like knowing a couple of short cuts in Revit will make your life easier. But what this does is it removes-- as long as I know that syntax, I want, say-- 12 is an arbitrary choice on my part, but I want 12 of these ribs.

And so instead of figuring out what one divided by 12 is, and then starting and making sure that-- no, one divided by 11. It would be one minus, duh, duh, duh, because you don't want the edge. You know, that kind of thing? Instead of doing all that, you can just do zero to one, and I want 12 of them.

So that's exactly what I did here. And that's the reason that there are 12 curves there that show up. So that Surface.GetIsoline.

From there, basically what we want-- and the trick is-- I'm going to show this here. From the isoline, then we're going to get, for each of these curves, we're going to get their lengths. And the reason that this works is that there are 12 curves that come out of this. And then there are 12 lengths that come out of that.

They should actually be all pretty much the same. But there might be some differences in the low decimal points, which is fine. If we had taken more than just the center point of the surface, these really would have been very different.

From that, what we can do is figure out what the longest one is. So what's the longest curve, what's the length of the longest curve. And then divide that by the max panel width.

So I had this number that was an input that's called the max panel width. You see, I do this a lot, too, as a bookkeeping technique, is to use this code block, which you just get by double clicking, of course, but as just a pass through that does nothing, that literally has no purpose,

other than to mark something and to call it something.

So it's a way-- so I know when I'm working right here, I know what this is called. I know where it came from. I don't have to go hunt and find it. So as long as you put in a word, it will give you an input. And then, basically, it's calling it that.

So this max panel width, in this case, three meters. And what this does-- this is a little terse. I apologize. But you can read from inside out. I'm going to do this a couple of times. But this actually-- I like doing math in code blocks, too.

So this is doing-- so the max surface width, say it's three. And then I had the [INAUDIBLE] maximum. So it's going to be 44 something. What we really want is how many times 3 fits into 44. But not too many. You don't want a piece of panel at the end. And you don't want to stretch it, so that you have these panels are slightly larger than they need to be.

So if you know that 44 divided by 3 is-- I hate doing math-- what, 13 and something? No. 14 and something. And then, so if you-- but really we want it to round up. So that math ceiling-- there's a thing, you know, of if you're rounding, ceiling always goes to the next integer up. And then-- yeah. 15. So we want to round up to 15 in this case.

That's a good number, because now we know that's the number of panels that fit this way, according to the specification. Then once you know the number of panels, then it falls to just dividing these same curve lengths into that many pieces that are evenly spaced, and then finding the point at parameter. Or, excuse me. Point at segment length.

So that switch that happened here, in order to make them evenly spaced, was we used the parameter space of the surface to get these ribs, that are basically like construction lines. And then, once we had those, we evaluated them at lengths of the curve itself, not the parameter of the curve.

So that's an important distinction, because that means that it will be evenly spaced.

There's one other thing that's going on here. And this will come up again. Who knows what List@Level is? All right. Not that many. So what I'm going to-- I'm going to explain what this is doing.

The inputs to this particular curve, this node here, called Curve.PointAtSegmentLength look like this. So I'm just going to do two watch nodes. There's a segment length and then here are

the curves.

So, in this case, what's happening is the structure of the data is very important. And so you'll see it gets increasingly important as we go. There were 12 of these curves. But now we have 12 lists of lengths, and there's a list along each curve.

So for every one curve, we have a list of segment lengths. And in order to make Dynamo match that correctly, the thing to really look at is this thing called List@Level. If it doesn't work the way you think at first, you can actually force the node. If you look at this little right pointing arrow, you can check on List@Level, and you can tell it how to match what.

So this is important because it made this node-- basically, I took the concept, find a segment length along the curve, and I used it where I wanted, so that this particular curve at level one-- this says L1, that says L2. I'm matching that at L1, and then this guy at L2. So the whole list, this whole chunk, applies to that curve. And this next whole chunk applies to the next curve.

AUDIENCE: Is this new in 1.2?

COLIN MCCRONE: Yes. It's new in 1.2.

AUDIENCE: Are you going to talk about List.Map?

COLIN MCCRONE: Do you guys know List.Map? Anybody List.Map? This is what-- this mostly replaces List.Map. You still have to use it sometimes. And you may see it fly by on the screen at some point today. But that's functional programming. You take this node, and you can sort of pass it as an instruction down lower in the list. That's still sort of the default way that Dynamo works.

But List@Level is-- I believe it's much easier to understand. And it's actually much more flexible. Especially when, like, these things aren't matching quite the way I want it to. And you can sort of repoint these things.

There's actually, on dynamoprimer.com, there's already a section on List@Level, too, if you're curious about more examples.

All right. Picking up speed here a little bit. Once we have these sort of construction line arcs, then I'm going to unfreeze this guy. And then we can basically do this thing called List.Transpose.

In this case, I had so many-- I had points that are organized along the arcs this way. And then

I wanted to change that. So take them, organize this way, and then flip the list, just like you would do in Excel, like to transpose. Swap the rows and the columns. And that means that all of a sudden with [INAUDIBLE] curves [INAUDIBLE] Then I have these new, basically, construction lines.

And to make this clearer, I'm going to manage what's being previewed. So I'm going to turn off what was there before. All right.

So once we have-- that went a little too far. But we now have these rows over here. I know what happened. Turn that off for just a second.

OK. That's really where we were at that point.

The next part was we wanted to reorganize these points, basically, evenly spaced points, along each of these curves in little groups of four. But now the whole thing is arcing, right? So that means for each, we have to do the same thing we did before, when we were finding these curves.

But now we have to do it in this direction, so that based on the panel length-- and let's say it's something like this-- that we need to divide each of the curves into equal segment lengths that way, so that the lengths on the far row are a little farther apart than they are here, just because the whole thing is bending. But then we can control that with that same List@Level, in the same way, and get those points. And then reorganize them.

So this actually looks pretty simple, relatively speaking, because you have the max panel length, the length of each guide, divided into the same thing as before. It's taking the whole thing, dividing it by the panel, rounding up, and then using point curve at parameter.

And then, finally, this last guy, there are a couple of different nodes like this on the package manager. It's not in Dynamo out of the box. I've no idea why. It's a great concept. But it literally just takes a grid of points and then turns it into little cells of points.

This is in a package called Ampersand, which I write. But there are several others too. Quads from rectangular. Grid is another one on there somewhere. But there are at least five that I knew of at some point.

But, basically, this is an example of a custom node. You can double click on this and look inside if you wanted. This is quite a bit of list logic that you just don't have to figure out

yourself. You can just use somebody else's, which is exactly what this is doing.

AUDIENCE: What's the package called again?

COLIN MCCRONE: Ampersand.

AUDIENCE: [INAUDIBLE]

COLIN MCCRONE: What was that?

AUDIENCE: [INAUDIBLE]

COLIN MCCRONE: Oh. Yeah. So since it's Ampersand, it's like my favorite punctuation symbol. So that's why. And I renamed all my nodes that have the ampersand in them, so you can tell, if you find it in your graph, you know where it came from.

And so basically what this does, this is a quick check. So this is now more hierarchy in the list because now we had-- we had points in this direction, points in that direction, we already had a two dimensional list that way. And now, for each of these things, now we have this set of four corresponding to a panel.

This is an important concept that I'm going to carry through all the rest of the work, is that this list, you can think of as being an object itself. It's containing four points. But for any one of these things, you can figure out more information about the panel based on even the fact that I know that, for instance, this set of four points, I know that this is the second panel in the first row, just because of where it is. And you can use that information, too.

This is just a quick check to make sure everything looks right. But up here we can do, take these sets of points, and this point's input here is just looking for a list of points, and it will make a, basically, a closed curve. And if I turn this on, I can verify that, yep, I have a lot of little rectangles.

So that's most of the way there. However, that's not what Marina Bay Sands looks like. So the next thing to look at is that, really, it's supposed to look like a diagrid, Marina Bay Sands is.

And so you have these diagonal seams between triangular panels that are actually wider than the seams here. And in general it looks like that. But that's not necessarily how you have to find it logically. Logically, it's really just a checkerboard of different conditions. So I've false colored them blue or green here. And then, zooming in farther, there's a light green set of

panels and a dark green, light blue, dark blue.

And the rest of it is really a list manipulation issue to make sure that you are treating every other row the same way, right? And every other column the same way. So what the point of this is, is if you're looking at the data, the first row in the first-- the first panel in the first column, we have to look at that. We have to divide each one of these into panels. I mean, into triangles. But we can actually figure out how to divide into triangles based on where it is.

All right. So the way that that works-- I'm going to turn off the preview for this guy. Next thing is I'm going to turn this on. Unfreeze that.

OK. So this is basically doing like I was kind of describing. There's a node that I made, also, that just literally splits odd and even. So you can do this whole list in. And all the odds go one place, all the evens go one place.

So the first thing you need to do is you split every row, so here are all the odd rows, here are all the even rows. So these are all the green rows up here. And here are all the blue rows down here. And then you split odd and even basically every column.

So, diagramatically, I've colored the green group here, [INAUDIBLE] the green set of panels in the diagram, the blue in the blue panels. And then this is just a quick check right here, just sort of off to the side, to make sure that once you have these sets of points that are sorted correctly, filtered out correctly, that you're dealing with the correct set. So [INAUDIBLE] quickly, you can make a surface by corner points.

The next problem is that now you have that, that this looks-- please don't be scared. I'm not going to go through this part, because I just-- what we need is actually a variably offset triangle. Because according to the diagonal seam has to be wider than the other seams. So why not, if I'm somebody who's managing these kinds of libraries of things, why not figure that out once and make a generic, so we can use it again.

So what I did was I made a node where you give it the corner points, and then what the offsets are. And then that kind of thing. So, basically, what I'm telling you is we're going to find these corner points, reducing the problem to finding-- there's a four-pointed thing here. There's a missing point over there. And we're going to give it a set of three. And then tell it what the offset should be, basically the diagonal offset, or the [INAUDIBLE].

This is what it's doing inside. I know it looks crazy. But you could just use it. So it's in the

Ampersand. And it just makes a variably offset triangle.

Well we'll do then, for each of these rectangles, is basically take the corner point. So this is zero, one, two, three. Then blue gets connected this way. Or that way. And then green's that way.

There's also a strategy here, too, and why that particular direction. A couple of reasons. And this is a really good strategy, especially if you're dealing with adaptive components, is that the way you specify adaptive components in Revit is it by a particular ordered set of points.

And so if it's a triangular panel, for example, then if it's winding one direction, you'll have it facing this way. And if it's winding the other direction, it'll be facing the other way. So you should be consistent about how you're picking the points and how they're winding.

The other issue is that, remember, I had this variably offset triangle. So in each of these cases, I'm going to pick the points so that the diagonal is last. In all of the cases. So that way I can just use one piece of logic to do everything after I've picked the points.

So this is an example where it literally helps to go off to the side of a piece of paper, draw a diagram with, like, what do I want to do? And then stitch them together from there.

So in Dynamo, what this looks like is each of-- this is the light green set, the dark green set. You can see the same thing is happening, except that I'm choosing a different set of points here. So I'm starting with number three, then-- basically, which is the fourth one-- then the first. Then one. In another case, it's two, three, zero. Zero, one two. Those numbers come from that diagram of what order to connect those points in.

But that's literally the hardest part of this whole graph. And once that's done, then you stick it all in one list and then do this variably offset triangle you can see showing up right there.

OK. Let me turn this guy off. And show you what this looks like at the end.

OK. So we have these surfaces. And we can look at, then, what that looks like here. And I'm going to go back and also turn off wherever those length curves are. And, again, you can find these just by whether or not the node is light or dark. So I'm just going to uncheck preview there.

So this is what the surfaces now look like in Dynamo. So you can sort of check it out this way,

and verify that that does indeed look like what it's supposed to look like.

Any questions there, so far? OK.

There's quite a bit more to get to here, so I'll try to do my best. Once you have something like this-- I'll talk about this again at the very end-- but once you have the surfaces, there's a lot of really interesting data here, a lot of pieces of information. And, for example, you could actually color it by size.

So I'm going to just turn on this coloring here, basically to each of these surfaces, and then looked at how big it is. And then here is colored by-- this is colored by whether or not it's a special cut, because this is not just perfectly plain underneath. It's actually supported by something. And the buildings come in at this particular spot.

So I also took that geometry from before, cut that out. And then you can actually ask for each one of these panels-- I mean, those are all the nodes, that's the whole thing right there-- once the surfaces are done, to actually identify where the special cut panels are.

In this case, it's coloring them differently. But then, also, that could just be a piece of information that you could write. You could actually separate them out in a branch, and then treat them with special design documentation and flatten it out, graphically, that way, too.

AUDIENCE: [INAUDIBLE]

COLIN MCCRONE: I always use Grasshopper. Yeah.

AUDIENCE: Any chance you would share that with us?

COLIN MCCRONE: Give me your email. Yeah. Yeah.

AUDIENCE: --dissect it a little bit? I think there's a lot of things that you could learn from--

COLIN MCCRONE: Yeah. Absolutely. There wasn't really a way for me to do it, I don't think, through-- I mean, maybe I can look again. But I have no problem sharing any of that.

This is the panels in real size, what they look like. This is a mockup. And then this is the construction. One thing I did forget to mention was that part of the design constraint, one of the reasons that these longitudinal lines had to be very clear, is because that's how the panels were installed, is by these sort of rigs that would move up and down and crawl up and down

underneath.

And that's the finished version. That is also the end case right here, which is a little different, and I didn't talk about that yet. And that's the point of this next section, because there are similar processes about how to pick that apart.

In this case, there's a nose. There's basically a special nose cone at the end, because it came to a point. But there's also-- it's hard to tell in this particular picture-- but as the panels were getting narrower and narrower, some were basically pared out, or picked out, and sort of combined into larger panels. And so we're going to look at that for the next example-- or for Project Jewel, also.

So I want to talk about Project Jewel. It's, again, [INAUDIBLE] this asymmetrical surface.

AUDIENCE: Before you get too--

COLIN MCCRONE: Yes.

AUDIENCE: You mentioned 85% of the panels were similar--

COLIN MCCRONE: Yep.

AUDIENCE: --or the same. Was that part of a script, too? [INAUDIBLE] just naturally occured [INAUDIBLE]

COLIN MCCRONE: Yeah. So it was both. There was-- because you can also-- since there are gaps between them, you could actually flex them a little bit, and get a little more bang for the buck that way.

But because of the logic of the script, and setting up the rows to be perfectly evenly spaced like that, all the panels, I mean, all of the light green shaded panels in a particular row are exactly the same. And all the dark green shaded panels are exactly the same type thing. Yeah. Thank you for-- that was part of me shading that, too. I forgot to mention that.

All right. So this surface is much weirder. It's no longer-- you can't flatten it like a rectangle anymore. It has these conditions where there are these datums, so along the footprint, along the waist, which is kind of at a particular spot where it has to-- it's a structural reason it has to be perfectly planar and level to the ground.

And which also means the division between the panels has to do the same thing, because the division between the panels is the structure.

And, finally, it has to be perfectly planar at the top, based on airport regulations with radar, or something like that.

And, finally, the oculus must be circular, and planar, and also parallel to the ground at that point, because there's water coming from it. And they wanted to make sure that hydrologically worked out well.

All of that's a problem. At least, a computational problem, because then you basically have-- it's like this whole thing is three separate paneling projects. There's everything between the two, this thing, and then the next one. And because of the way it's set up, you have to make sure that the panels' edges touch those datums perfectly every single time. And then that there's an evenly spaced set of panels here, even though this distance is far smaller than this distance.

OK. And just a little bit of vocabulary that I'll use. This is a few different ways of dividing up the surface.

On the left, I'm going to call these radials. They basically are these curves that follow a design surface from the outside toward the center of the oculus. And the center. These are called biases that go on a diagonal, basically, and they sort of spiral in. And then on the right are-- I'm going to call them hoops. And, basically, the spirals take place at where the hoops and the radios intersect and just go on diagonals.

The final built version, as you probably saw from the image from before, looks quite a bit like that. I'll come back to this one in just a second.

OK. So I'm going to open up that guy. And I'm going to show you how to panel this two ways, both because I want to step up in the level of detail that we're getting into, but also because that's exactly how the design process worked, that the whole original design was that you have these spiraling panels that went all the way in. And then we'll see in a second that that actually causes quite a bit of visual problem in the center. And then I'll talk about how to fix it.

So, again, there's a CAD input here that I have, which is that I've basically ripped the design surface from the [INAUDIBLE] model, that I'm stuck in a SAT file, and then looking at that. So here is, indeed, the design surface. You can see how it's slightly asymmetrical.

OK. And I can [INAUDIBLE] that it is the design surface. This guy, I'm going to turn that off.

The first thing to do here is probably not a surprise. It's really similar to what we were doing there before, but in a radial sense, as opposed to a rectangular sense. We want to get the radial guidelines.

And in order to do this, I'm going to-- so unfreeze that. So the first thing to do is identify the maximum width of the panel. Because where those radials are, you can't just have them circularly symmetrical. That doesn't work. Because the building itself is not circularly symmetrical. And you don't want the panels on the wide side of the building to be too big.

So the next thing to do is really to identify the widest point. And this was something that was done as part of the geometric construction in the Rhino file. So it's also information that I stole. So this is from the original file. So you can see that's basically the widest point. This is not the structural waist. It's just the widest point.

OK. From there, what we have to do is find, basically, just like before, find the length, divide by the number of panels, or divided by the panel width, and then round up. Although this time we're not just rounding up, because-- oh, no. We are this time. I take it back. We can just round up this time, by one.

OK. The next thing to do is to find those-- so here are the evenly spaced points. I'll unfreeze this. Can't unfreeze that. Why not? OK. So I'll unfreeze that. And you can see here, your evenly spaced points. Right there. Around the back. Let me zoom in a little bit.

Again, this is nothing quite that new. And then, finally, using these points and from the center point of the oculus, and making these lines between them. So I'm going to do this.

And as just a bookkeeping trick, what I'll do here is always make sure that the lines start from the oculus and go out. That way, I will always, for the rest of all this, work from the inside out. It will be important for the next thing. Just trust me. It'll make everything easier to do it that way.

And this is about-- the logic has nothing to do with Dynamo. In this case, I'm making these lines between all these individual points right here to the middle. And then extruding a surface up, doing geometry that intersect, which is a really great node. And then the curves that result are going to be these radial guidelines. And they're all different. But it's useful to keep track of them.

So once we're there, I'm just going to turn the preview off from here. And from here. And then

we have our radial guidelines. So, again, like by analogy, this is just like those lengthwise guide curves we had from before.

The next thing to do is to find points along those hoops that are evenly spaced. And this is an interesting challenge, because, again, this is kind of like three different design surfaces.

So this guy is what we actually want to do. This, again, this looks really complex, but let me explain it. Yep?

AUDIENCE: I'm just curious, how did you draw the surfaces? [INAUDIBLE]

COLIN MCCRONE: Yeah. So they were the points they I found along the widest point. And then, actually, what I did is projected them down to the X-Y plane, because basically I knew that was the bottom of the building. That's how I set it up. And then used the origin of the oculus, also projected down, made a line between them, and then extruded vertically.

In this case, this is moving from the oculus out. Each of these-- let me just explain this diagram. I haven't thought of a better way to do this. Each of these dotted lines right here is one of those datums that has to-- where the panel width, or the panel seam, has to line up there.

And then, this is kind of exaggerated, but the darker tick marks, then, are where the individual-- the interstitial seams would have to line up. And they'd be evenly spaced. So here they're evenly spaced. Here they're evenly spaced, here they're evenly spaced. They might be differently evenly spaced. And on a large enough surface, you can't tell the difference at all.

And so, in this example, as if this were the length of the curve-- this is laid out from the oculus, then the apex, then the waist, then the footprint. If this is the length in meters, something like that. If the footprint is all the way to 100 meters, there are five things in between, this is what the series would look like in that design script syntax.

Because, in this case, we are starting at zero. And we're going to-- we want four things. So we're going to get these first four ticks right here. And then the width between them would be 33 divided by 4. So that's this section.

Again, I know this looks really complex. But it's a whole lot more complex to figure out all the math than to just know this.

Then the next thing is over here, we're starting at-- this should say 33-- we want three of them. And then now the difference is 70 minus 30, divided by that number.

Then, over here, it's same thing, where we're starting at 70-- or, in this case, sorry about that, that should say 70-- we want 5 plus 1, in this case, because we also want the end in that condition. And, again, dividing by the width there.

This is the kind of thing, again, you would have to go and draw out a diagram on a piece of paper. But if you do this, then the whole graph, everything you're doing, is simpler because you don't have to actually divide this into three different panelling problems. You just did it. Just by writing this out, you've divided this into three different panelling problems and solved all of it.

So this is what that looks like in Dynamo. And I actually describe this quite a bit more in the handout. And the handout is really quite long and extensive, but even talking about the going backwards [INAUDIBLE] by analogy, this is what that code block looks like. I know it looks horrendous. I apologize for that.

But what I just described to you is literally those last three lines. You start at zero. You have a certain number of things. Then you move up that way. So, again, hopefully that's useful.

Once we've done that, I'm going to unfreeze this. Then, basically, we're using those segment lengths to find our new grid of points.

Now, these new grid of points, I know from here on out that all these grids-- you have this certain level here that lines up perfectly with that datum just like it is. I don't have to worry about that anymore. This is just my new grid. So whatever this is.

And this is, again, a two dimensional piece of information, just like on Marina Bay Sands.

All right. So we're here. The next thing to do is we have to get this triangular pattern. And it works a little differently now than it did before. Right now, the way that the data is organized is like this. It's along the radials, and you have the perfect grid.

But Jewel here is actually paneled by triangles. And the way that this particular piece of logic works is that every other point is basically thrown out of every other row in an offset pattern, so that the panel on the very end should be the triangle here, triangle there, triangle there. And then and then moving up.

So similar to before, you do this [? list.splitoddeven. ?] You get rid of every other point in every other row. And [? then ?] sort of offset. And then these are the points by which we will create the biases.

So you can kind of see, even the line up here, that these are going to become those angular-looking spiraled features of the end panelling.

All right. So that guy looks like this. It's not even that many nodes to do it. But you basically take the points which are-- the way I've been calling these things is points by hoops, because that's how they're organized.

And then, from this, we're going to take every nth item-- this is a little too detailed probably to explain all the way through, but, again, it's in the handout-- and to basically take this list, and then get rid of them.

The trick here that I tended to use was the modular operator. I really like this guy. The [? modular ?] operator gives you the remainder after a division. And so if you take each of the index of the thing, what you really want is to make this every other thing.

You really want a list of zero and one here because you're going to start at a different spot when you're taking this every other pattern. You're going to take this first one, next one you take the odds, and then you take the evens, and the odds and the evens. So you want this list of zeros and ones to do this in the first place.

You can take the index and then just divide it by two, and see what the remainder is, which is either zero or one. So that's a nice little trick there. It makes the logic a lot simpler. But what you have at the end of the day, then, is an every other set of nodes. So if I turn this guy off, you can see that we've actually now gotten rid of-- we basically have an offset pattern.

The next part is I've actually simplified a bit into just a custom node. And I'm going to talk about it tomorrow in the handout, but I can give you this file, too, about what's inside of this. But to take that and then basically stitch them together, it has to do with shifting the list a little bit and making triangles again. You kind of already saw that. So I'm just going to turn that. Just skip that part. But it's inside that custom node.

And at this point, we can check to make sure that this is right. This should actually give us sets of three points, each describing a triangle. And we can check to make sure that that's right.

And we'll see that, hopefully.

There you go. You can see the individual-- it's kind of hard to see, but it's a little faceted. But we're not done yet, because the issue is that it gets too dense in the middle.

So if you look at what we just did, and what the original design was in the competition was something like this. But then it gets so dense if you're looking at actual panel widths for the size of this building that you have to do something to more strategically deal with the fact that the surface is pinching in the middle.

And there were many tens, I think, of design iterations going through. There was a lot of form finding that went back and forth, especially with the structural engineers. And in the end, the design that was chosen was to, at certain rows, to have the number of panels. And there's a particular way that this will work to do that.

So this is the end result, is to privilege the visibility through the surface, as opposed to the biases, those diagonals, are not perfectly planar anymore. They'll skip a little bit, but in order to preserve the panel size.

So, diagrammatically, this is the difference. So from what we just did, the difference now is that for each of those new sections of the panels, picture that these light rows right here are the radials. And we basically-- instead of skipping every other point, we have to skip every, basically, power of two.

Here's a zoom in on the left of what that means. So, say this is-- I was zooming in at one of these particular boundaries, we're repairing the number of panels. So coming from the footprint up, we have the maximum number of panels. And then we need to halve it.

And where those arrows are pointing, you can see that there are actually three panels in that row, not four, like above, or two below. And the difference is this particular kind of strut right here that doesn't exist in any other row, because it's sort of matching these two triangles together. And you can see that the size of the triangle jumps there.

But then, basically, that's what we need it to look like. This was the means by which that was achieved. And then the number of times you need to skip rows, the next one would be you'd skip every other. And this way, you'd skip every 16, for example. So the number of changes that need to happen in the file are these.

So, in scheme one, which we just showed you, and then in scheme two, the number of panels per row in scheme one is the same as the fewest number of panels to fit around the widest ring. That's it. It's constant the whole time.

In scheme two, it's the fewest [INAUDIBLE] to fit around the widest ring, rounded up to the nearest power of two, because we need to have-- if we're going to have four different regions, and you have to skip 16 at some point, your total number of panels has to be divisible by 16. If you had five divisions, you need 32. So that's actually part of the script, too.

Then selecting the node points. We're no longer selecting every other. It depends on what region you're in now. Now you have to select every power of two to whatever your region that you're in. Are you the first one, it's every two. If you're second one, every four. It's the next one, every eight.

Then, finally, when we create the panel sets for, basically, that one node that I wrapped it all up into, we take three points and make a triangle. Three points make a triangle. That still works everywhere. Except for on those boundary rows, which are now in new condition.

So these are the major changes. Basically, you can take what we just did and make a few tweaks. So this is what that looks like. An

Open this guy. Repaired panelling. It gets a little more complex. But the same flow of the logic to getting the radials, to getting the points, getting rid of some points, reorganizing them into triangles-- happens in the same order. It just gets a little more complex.

The first thing that happens is there is now a new set of parameters. At what row numbers, moving out from the oculus, are you now introducing on one of these rows? And this is actually then parametric. You could change the back and forth.

This whole thing's got to run. And then I can walk through it.

While we're doing that, I want to just point out another thing here, too. Nope. I've got to just wait for that.

OK. This is starting off from where we were before, is that these grid of panel points here. And then now that's exactly the same as before.

However, now we want to introduce something where we basically choose by the power of

two. And this gets a little complex, but not too bad. Just select the nodes by region. So this is almost toward the end of the graph, to we have to sort of have [? start ?] adding things. And, at this point, this is what that looks like.

And this is all the complex part right here. We zoom in. You should always read these things from the inside out. That's the series again. You have this power of two thing that I was talking about. And then, basically, what you're really after here is-- let me show you-- this particular thing here.

And this is a list of offsets per row. So if I say I'm going to run this by hitting this guy, and we're sitting-- one, two, four, eight, depending on the row we're in, which is exactly what that piece of logic did. And then we're going to, basically, same thing as before, take every nth item, except that now it's some power of two.

So that's what we started with. That's do, do, do, do do. This is what we started with. And we got rid of a bunch of points. And now it actually looks much more sparse.

After this, what we need to do is just like before. We need to create those sets of points describing panels. And in this case, it should look exactly like before, with this one particular node that did everything.

But now there's a new piece of logic where there are those division rows. So that's added at the bottom there. So it's sort of bifurcated those.

And we're almost out of time here. So I want to make sure I-- while that's running in the background, what else this looks like. There's something else. This is a little complex, so I'll leave that for the handout.

Placing families. There's one other thing. Before I show you that, I should show you. This is what the panelling kind of looks like, is that you have these different family types that are placed. So now, once it's paneled, then you have maybe opaque panels toward the bottom. And then toward the center, where the program changes because there is a tropical forest on the inside that needs sunlight, it needs to become glass. And there's a gradient effect between them.

So the way to do that in Dynamo is just one more piece of logic to figure out what they are. And so the graph here is showing you per row-- these are the rows of panels, moving from the oculus on this side to the footprint on that side-- this is how opaque it should be.

So there's a region toward the bottom-- or toward the center, excuse me, of the oculus, where all the panels are glass. So that means that the percentage opacity is zero for all of them. Then, for these-- on average. Total, on average.

And then there's a transition row, where we're moving from-- the center row is mostly transparent, until all the way to opaque, and then all the rest of the rows are opaque. Basically, the logic is to take that percentage, and then to choose randomly along each of these rows, how many-- making sure that the number of total panels is rounded, basically, makes that percentage.

So you can see on the inside. On average, there is something-- if there are 10 rows here, then they would be 10%. Or 1 in 10 of the panels would be opaque. And you pick randomly what they are. And then that gradually increases.

The method to do that in Dynamo, which I think is useful, is to ascribe a number to that. And then to use either a zero or a one, or a two, if you have more panel types. And then to choose the family type based on that.

The last thing-- I know I'm like just out of time-- I'll show you that guy. This is something you can get. And this is-- I'm using false color in Dynamo because it's a lot easier to see than the primary values of the family instances in Revit. But this is false coloring by panel type. Green is one type. Blue is another. And then, also, the lightness grouped by relative size.

You can also do things like, once this is parametric-- this actually happened as part of the design process, is looking at the energy incidents in the surface. It takes the age of the universe to calculate the solar effects of that many panels. But since it's basically the same shape, you can actually decimate the number of panels, maybe reduce it to a tenth, and it basically gives you the same information.

And this is the case, this would be just faking that by just looking at-- the sun was coming from that particular direction, with the solar and [INAUDIBLE] incidences, on any one of these panels.

And the last thing I'll say is that as a lot of this is-- because this stuff is complex, it's something that Safdie Architects has certainly developed this way to communicate this is how design surface was made, and this is how it was communicated with different contractors, and

consultants, too.

So it's in sort of an open source method. Everybody had the Grasshopper files. Everybody has these diagrams of how these things were made. And it's a way for the architect to continue to control what the process is.

So we're at 2 o'clock. So it's the very end of the time. But I'd love to take any questions if you have them.