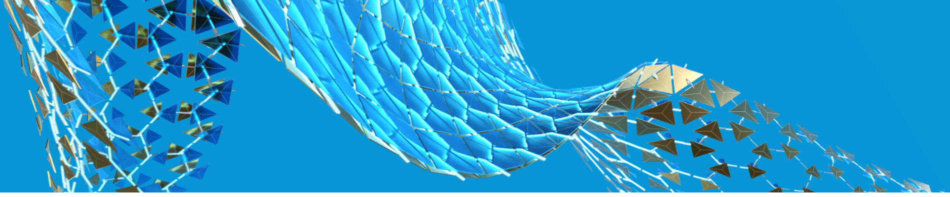


CP473669

Debugging your Fusion Design: Let's get rid of those red and yellow features

Jeff Strater
Autodesk, Inc.

Phil Eichmiller
Autodesk, Inc.

Learning Objectives

- Identify common errors in a Fusion design, their underlying causes, and ways to fix them
- Internalize which design practices will produce the most stable designs
- Understand how to edit a Fusion design in such a way that you do not introduce errors into your design
- Explore some "debugging" tools and techniques available in Fusion

Description

Even the best Fusion 360 designers sometimes end up with errors and warnings in their designs. Those errors and warnings are there for a reason.

The purpose of this class is to help you fix your Fusion 360 design when it breaks, and how to prevent those errors from occurring in the first place. We'll explore the tools and techniques that are available in Fusion to help you repair a wide variety of fusion errors and warnings that can happen when working with a Fusion design.

We'll expose you to enough of the internals of Fusion so that you can understand what the root cause of these errors are. We believe this understanding will help you both to fix the errors when they happen and also how to design/edit in such a way that you avoid ever seeing the errors in the first place.

This class will only cover the Design workspace and tools, it does not cover CAM/Manufacture, Simulation, or Drawings.

Topics include – parametric part modeling errors, assembly/joints errors, sketch errors, and more.

Speaker(s)

Jeff Strater is a Senior Software Architect in the Fusion 360 team. I've been with Fusion since the very beginning. My focus is on general modeling/sketching. Before that, I was a developer and architect on Inventor, also before R1. So, I'm a long-time CAD guy. When not working with Fusion or its customers, I like to run, cycle, hike, and read science fiction.

Phil Eichmiller is a Senior Software Quality Assurance Engineer for Autodesk, on the Fusion 360 team. In addition to testing and designing with Fusion 360, he enjoys working with the Fusion online community, presenting, and teaching Fusion 360. He used Inventor software as a product designer for 16 years. He enjoys sharing his knowledge by teaching Fusion 360 for the CAD program at Portland Community College. Roller Derby is his favorite pastime, especially watching his daughters, who are both derby stars in Portland, Oregon

Table of Contents

1 Goals and Starting Principles/Guidelines 4

1.1 Goals of this class 4

1.2 A Few Starting Principles and Tips 4

1.2.1 Fix it when you first see it 4

1.2.2 Warnings are serious, too 5

1.2.3 Compute All is your friend 5

1.2.4 Editing while rolled back in the Timeline can hide errors until later 5

2 Errors and Warnings and Parametric Design intro 5

2.1 Errors and Warnings 5

2.1.1 The Difference Between an Error and a Warning in Fusion 6

2.2 A Brief Introduction to Parametric Feature Design 7

2.2.1 Feature Dependencies 7

2.2.2 Parametric Naming and Matching 9

3 What Causes Fusion Design Errors? 10

3.1 Geometry Errors 10

3.2 Dependency Errors 10

3.2.1 Types of Dependency Errors 10

3.2.2 There are Two Different Kinds of Geometry Dependency 11

3.2.3 Each Feature May Handle Matching Errors Differently 12

3.2.4 Compute Early 13

3.3 CAD Naming and Matching 17

3.3.1 Problem description 18

3.3.2 How does this help me in Fusion? 23

3.3.3 Fusion Naming – the first half of Naming/Matching 23

4 Error Debugging/Fixing Strategies 28

4.1 General Strategies 28

4.1.1 Fix Errors/Warnings When They Happen 28

4.1.2 Fix the First Error in the Timeline First 28

4.2 Debugging Sketch Errors 28

4.2.1 Debugging Sketch Plane Failures 29

4.2.2 Debugging Sketch Solve Errors 29

4.2.3 Debugging Projected Geometry Failures 30

4.3 Debugging Modeling Failures 31

4.3.1 Geometry Modeling Errors 32

4.3.1.1 Fillet Geometry Errors 32

4.3.1.2 Shell/Offset Geometry Errors 35
4.3.1.3 Loft Geometry Errors 43
4.3.1.4 Combine/Boolean/Join/Cut/Intersect Errors 46
4.3.1.5 Sweep Geometry Errors 47
4.3.2 Dependency Modeling Errors 49

4.4 Debugging Assembly Errors 50
4.4.1 Joint Failures 50
4.4.2 Understanding Ground in Fusion 52
4.4.3 Understanding Component Positioning Model in Fusion 53

5 Preventing Errors in Your Fusion Design 57

5.1 Dependency Management 57
5.1.1 Minimize Unnecessary Dependencies 57
5.1.2 Choose the Most Stable Dependency 58
5.1.2.1 BRep dependencies 59
5.1.2.2 Sketch dependencies 59
5.1.3 Be Careful When You Edit Sketches 59
5.1.4 Using Parameters Instead of Dependencies 60

6 Misc Topics 60

6.1 Design Iterations 60

6.2 Names and Groups 61
6.2.1 Name everything 61
6.2.2 Group features from a component together 61

6.3 Debugging Appearances 61

6.4 Cannot Delete a Design 62

1 Goals and Starting Principles/Guidelines

1.1 Goals of this class

Our goal in this class is simple: Helping you to fix design errors in your model, when they happen, by understanding why design errors happen and the tools and techniques available in Fusion to help you understand and fix those errors.

But, also, and maybe more importantly, we want to help you to prevent introducing those errors in the first place! Again, we will approach this by helping you understand what those errors are trying to tell you.

We will spend a bit of time "under the hood" in Fusion's implementation. With any system, it is much easier to fix problems if you understand how the machine functions. For example, say your problem is "my car won't start". It's easier to diagnose and repair this problem if you know the subsystems involved:

- Battery
- Generator/alternator
- Fuel system
- Cooling system
- Lubrication system

You could always just call a tow truck, but if you know that the car is out of gasoline, it may be easier and cheaper to just get a gas can and put gas in the car.

This class is focused on the Design workspace (because that is our area of expertise). Further, this class is centered around Parametric designs. Direct Modeling designs can also have errors, but the nature of those errors are much different, and are not our focus today. The two of us can't help you very much with CAM or SIM or Drawings errors, although many of the principles are similar, and can be applied there as well

1.2 A Few Starting Principles and Tips

There are just a couple of things we want to lead with. Call them tips, or recommendations, or whatever. They are useful, we think, to keep in the front of your mind when doing parametric design.

1.2.1 Fix it when you first see it

This may be the most important thing you will hear in this class. It's really easy to just ignore problems in your model, especially warnings. "it's just a warning, so I can ignore it, right? I can fix it later, if I have time". If you let errors pile up, they get harder to fix, for a number of reasons. One of those reasons is: The more time that has passed since the error appeared, the less you will remember what design change you made that caused the error. Barring some Fusion bug, design errors <u>always</u> happen as a result of a change that has been made to the design. When you make that change, and an error occurs, stop, understand the error, and fix it right away.

Another reason to fix errors promptly is to prevent future problems. If you let an error/warning go unfixed, you may create references or dependencies to bad geometry.

Then, if you do fix the problem later, the fix may actually cause further errors from downstream features.

1.2.2 Warnings are serious, too

Another critical thing to remember: Warnings are bad, too. Just because a Warning seems less serious than an Error, you should NOT ignore Warnings. These are indications that your design is no longer functioning correctly. There are breaks in the parametric dependency that are very important to understand and fix. Just because it is a Warning, and not an Error does not mean that you get to ignore it.

1.2.3 Compute All is your friend

Think of Compute All (CTRL-B) as a kind of a crude “design checker” - if a Compute All works, your model is in good shape. Combine that with the above tip – frequent Compute All checks when rolled to the end of your timeline are good to do. If you have a large design that may take a few minutes to compute, start a Compute All before a coffee break… After a successful Compute All is also a very good time to do a Save. During Save, I usually put a comment in the version description about “clean model” or something, so I know I can get back to a good state if I need to.

1.2.4 Editing while rolled back in the Timeline can hide errors until later

The one caveat to the statement above: “design errors always happen as a result of a change that has been made to the design” is: while those errors technically happen at the time of the change, they may not reveal themselves until much later. If you are rolled back in the Timeline, features that are to the right of the design marker will not compute (that is essentially what the design marker does). But, later, if you roll to the end, errors that are caused by an edit will be revealed. So, the tip here is: if you are editing when rolled back, be careful to roll forward frequently so as to catch errors as early as possible.

2 Errors and Warnings and Parametric Design intro

This section will dive right into some of the internals of Fusion to help understand the nature of Errors and Warnings in Fusion.

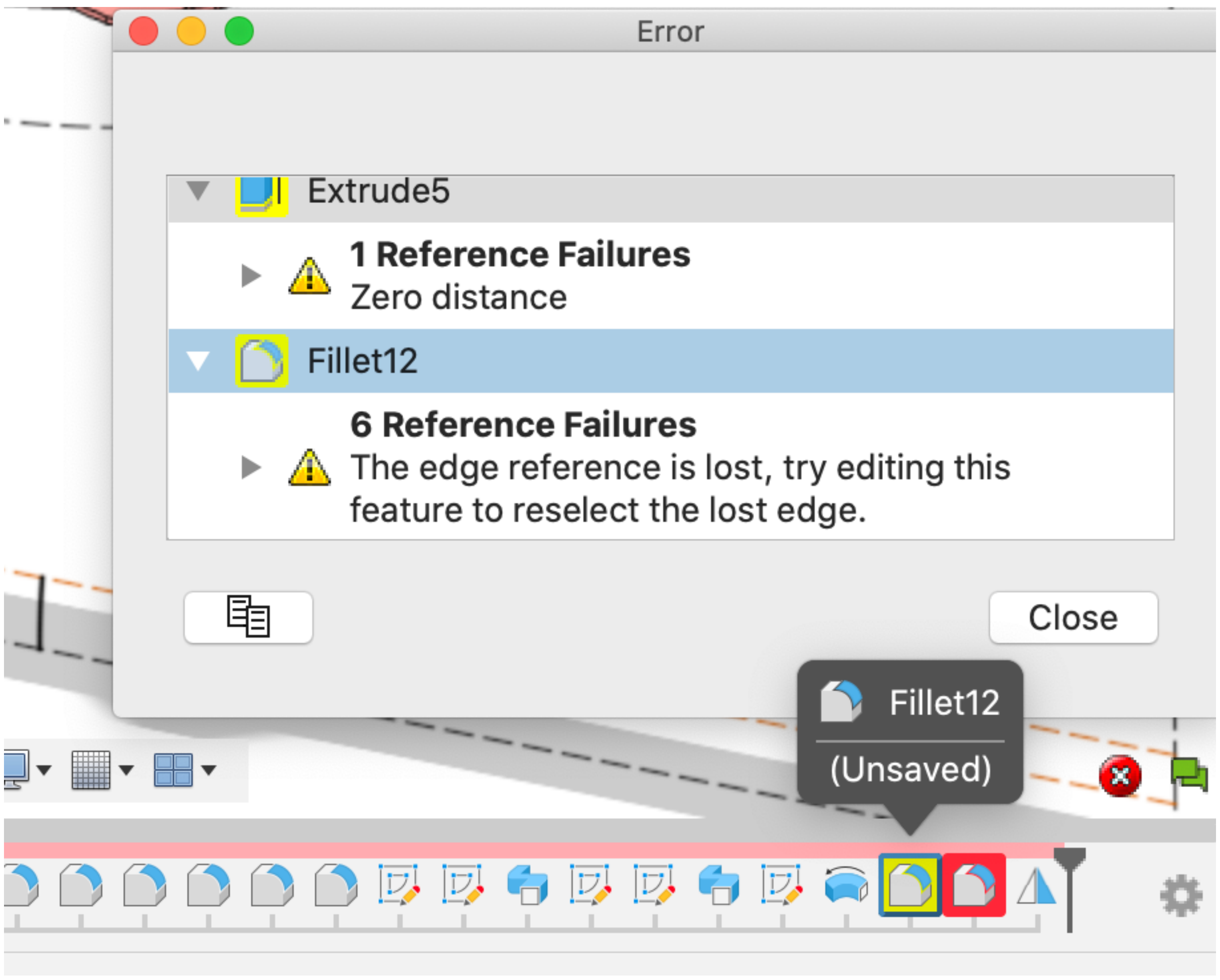
2.1 Errors and Warnings

Both of these states indicate a design *failure*. What constitutes an Error or a Warning in Fusion? What is the difference between the two states? What causes these design faults to occur? How is the rest of the design affected by Errors/Warnings? In this section, we’ll explore some fundamentals and Fusion internals from the area of feature status to help us understand errors and warnings more completely.

2.1.1 The Difference Between an Error and a Warning in Fusion

A Timeline *Error* is a compute failure that is so severe that the feature in question cannot successfully compute at all. A Timeline *Warning* is a failure which is not severe enough to prevent that feature from computing some result. A second major difference is the condition's effect on downstream features in the Timeline. An Error will block compute of any dependent downstream features, while a Warning will not. We'll explore this in more detail below.

Another difference is the propagation of the error status to downstream features. A feature with an Error will propagate that Error status to dependent features, while a Warning status allows downstream features to remain in a healthy status.



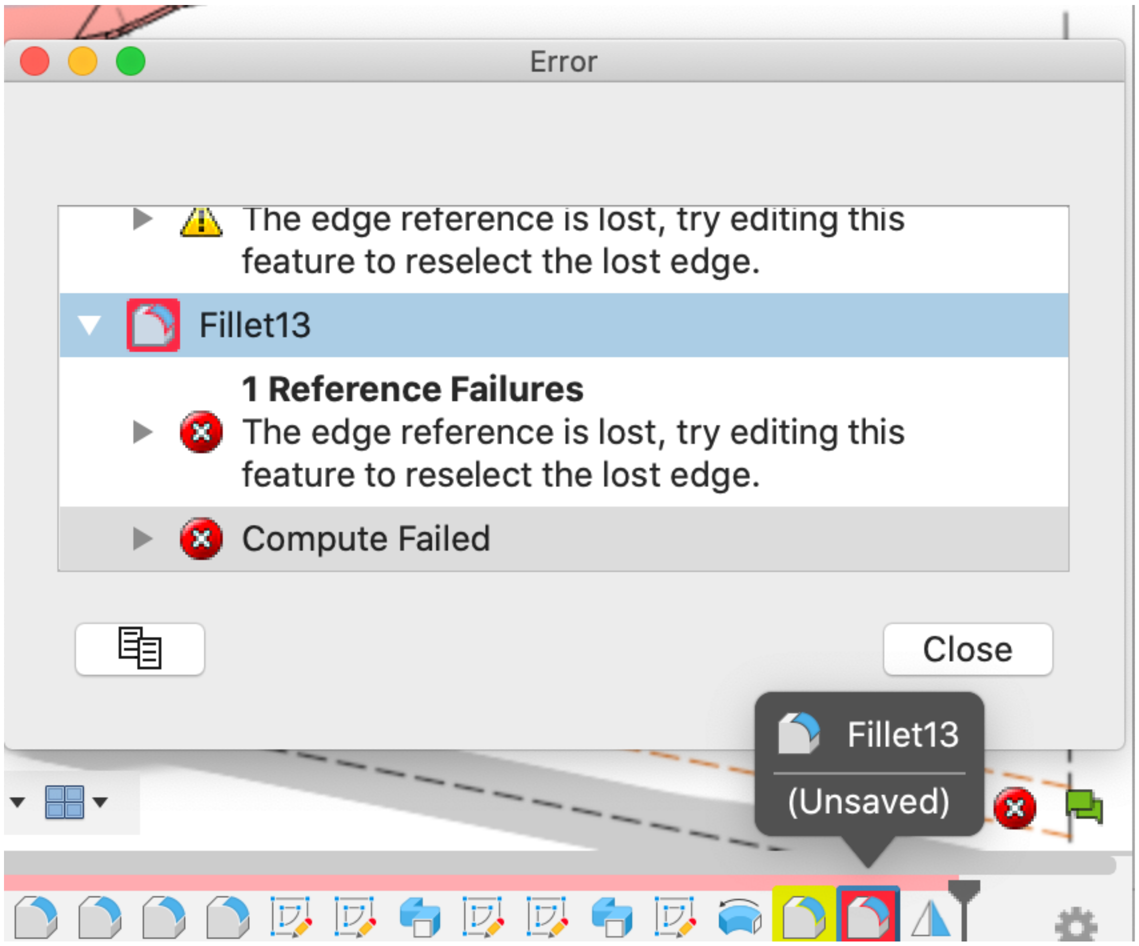
Error

Extrude5

1 Reference Failures
Zero distance

Fillet12

6 Reference Failures
The edge reference is lost, try editing this feature to reselect the lost edge.



Error

The edge reference is lost, try editing this feature to reselect the lost edge.

Fillet13

1 Reference Failures
The edge reference is lost, try editing this feature to reselect the lost edge.

Compute Failed

2.2 A Brief Introduction to Parametric Feature Design

A parametric design is a sequence of “features” (timeline entries) that are computed in a specified order. That order is determined by two factors: Creation order, and dependencies between features. This dependency is critical to our understanding of Fusion error handling. We’ll explore the nature of these inter-feature dependencies next.

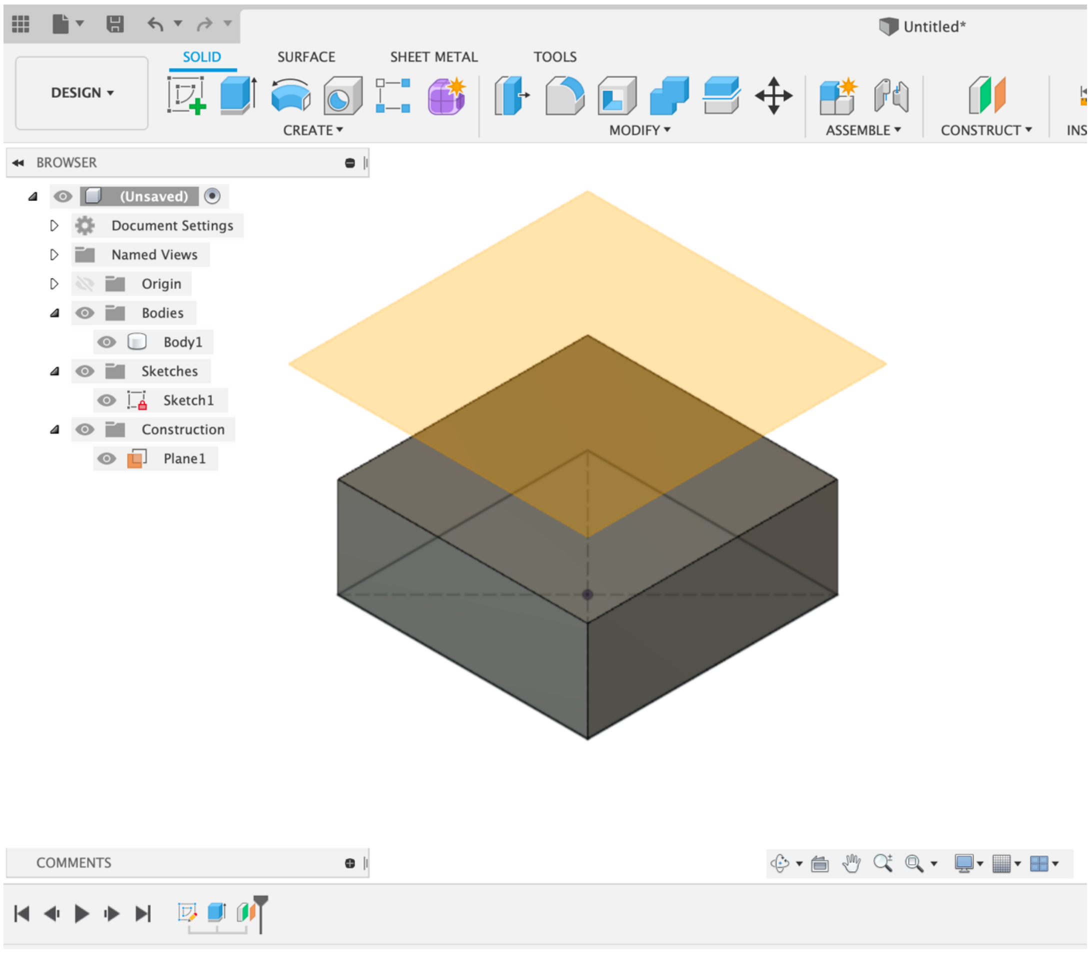
2.2.1 Feature Dependencies

What causes a dependency between features? If Feature B needs geometry or topology produced by Feature A, then we say: Feature B depends on Feature A. In the UI, this is **ALWAYS** because of a geometry selection in a feature creation command. For example: When you select a profile for Extrude, you create a dependency between the Sketch that owns that profile, and the Extrude feature you are creating. Selecting an edge to Fillet creates a Fillet feature which is dependent on the feature which created this edge. If you use a Box feature to create a solid body, then Fillet one edge of that body, you have created a dependency between the Fillet and the Box feature.

Here is a more concrete example:

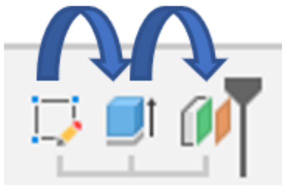
- Sketch1 contains a rectangle

- Extrude1 produces a solid body from that rectangle
- Offset Workplane1 created from the end face of that Extrude



This Timeline creates this set of dependencies:

- Extrude1 depends on Sketch1 (for the profile)
- Plane1 depends on Extrude1 (Extrude1 produces the reference face)



What are the implications of these dependencies?

1. Order of compute. In this case, the timeline order is the same as the dependency order, but there can be cases where features can compute (or partially compute) out of Timeline order

2. Dependencies determine what gets computed when you make a change. If you edit a sketch, every feature which depends on that sketch will get computed. If you edit a feature, every other feature which depends on that feature will get computed.
3. You cannot reorder a feature in the Timeline before one on which it is dependent. In this simple case, that should be obvious (you cannot reorder an Extrude before the sketch containing its profile, or an Offset Workplane before the Extrude that makes the face it uses), but other cases can be a little less obvious.
4. Error propagation. This is the important point for this class. Errors in a feature can propagate to dependent features. We'll look at this in more detail later.

The important things to remember regarding feature dependencies and error handling:

- **The dependency system is the core technology behind parametric design**. It is what makes the design "parametric" in the first place. The dependency between a Sketch and an Extrude is what allows you to edit that Sketch and have the Extrude update when you exit that sketch. This is the core distillation of the power of parametric design. However, as with all sources of power, it comes with dangers as well…
- **Any dependency is a potential source of an error.** Almost all feature errors you encounter will be caused by a dependency failure. We'll cover this in more depth below.
- **Be careful and intentional when creating dependencies.** In some sense, this is the entire moral of this class: More dependencies means more potential failures. Sometimes you need to create dependencies, but there are plenty of times when one dependency or another is NOT needed, so avoid those if you can.
  - Sadly, Fusion makes this problem worse with some of its default settings and behaviors. We'll show you what options to set to avoid the worst of these. These settings were chosen, ironically, to make Fusion appear to be easier to use. And, to some extent, that goal is achieved. However, that ease of use comes at a cost of creating a brittle design. So, it's debatable whether these ease-of-use settings are really an advantage or not…

2.2.2 Parametric Naming and Matching

Another important topic to understand in trying to understand design failures in Fusion is a subsystem that we internally call "Naming and Matching". This directly relates to the Dependency topic above, as most of the dependencies in the system are handled here.

This system can be described at its most basic as: Naming and Matching is the way in which Fusion tries to remember what you selected in creating a feature (Naming), and making sure that the same entity is found again at the time a feature is computed (Matching). Most dependency failures (and therefore most feature failures) are caused when a Fusion Match fails to find the right dependent object at compute time. We'll dive into this subsystem in a bit more detail later in the class.

3 What Causes Fusion Design Errors?

This section will cover the most common causes of design Errors/Warnings in Fusion. As stated above, nearly all Errors or Warnings are caused by edits to the design. Or, to say it another way: Any design fault is caused by… you. A design which does not have any Errors or Warnings will not spontaneously create Errors. If you've done a Compute All, and have not made any design changes, that design will continue to be healthy the next time you access it. In all cases, some design change is the underlying cause of Errors or Warnings.

Not counting data management errors, or other non-Design errors, there are two main categories of design errors:

- Geometry errors
- Dependency errors

3.1 Geometry Errors

This type of error seems to be a less common error, but let's discuss it first, so we can spend more time on the more common error type. Geometry errors include failures that are purely because the operation cannot be completed. For instance, if a Fillet fails with the error "Fillet could not be created at the requested size", it is because the modeling kernel itself cannot produce the fillet faces. These kinds of errors occur in features like Shell, Fillet, Face Draft, Loft, and other complex modeling features.

3.2 Dependency Errors

This type of error accounts for a larger percentage of all design errors. Reference errors are caused when a dependency (see above) for a feature is not resolved (or, the wrong number of them is found) at the time a feature computes. For example, in the case of an Offset Workplane from a body face, if the face that was referenced cannot be found, then we call that a matching error. Any dependency to geometry that is selected in creating a feature is a potential matching failure.

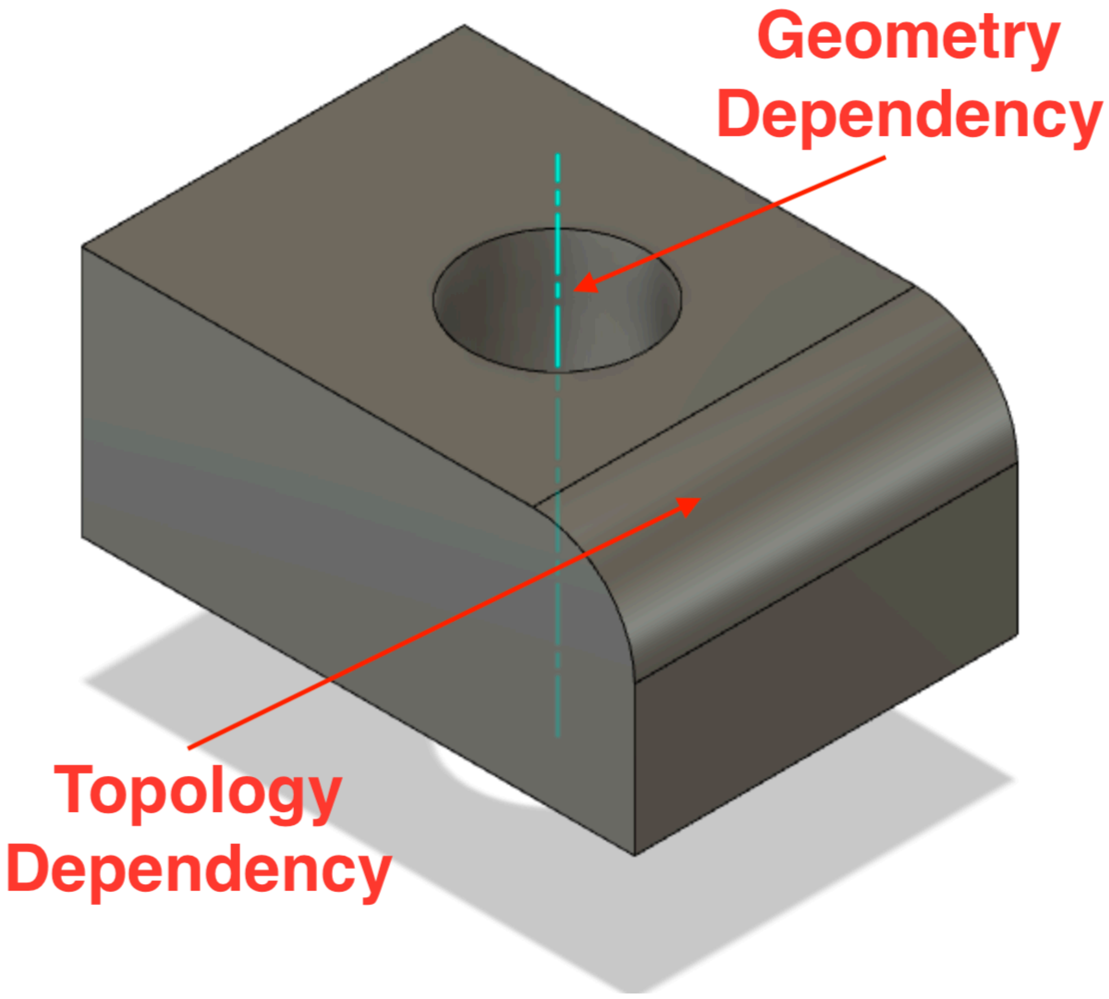
3.2.1 Types of Dependency Errors

There are three basic ways in which a Matching Error can occur:

1. **No Match Found**. If no match at all is found for a feature input, this is a direct failure. For example, Work Axis Through Edge has only one geometry reference – a linear edge. If the edge cannot be found, an error occurs.
2. **Too Many Candidates Found**. This kind of Matching failure happens if a feature is expecting N matches, but N+1 or more matches are found. This usually happens with face splitting or edge splitting. A reference that was a single entity at the time the feature was created becomes 2 or more entities due to an edit. As we'll see below, different features can handle this case differently.
3. **Mixed Results**. In this type of Matching failure, some references get resolved, some do not. Obviously, this cannot happen if a feature only has one reference, but features like Fillet or Chamfer, or Sweep Path, where multiple objects can be selected, can fall into this category.

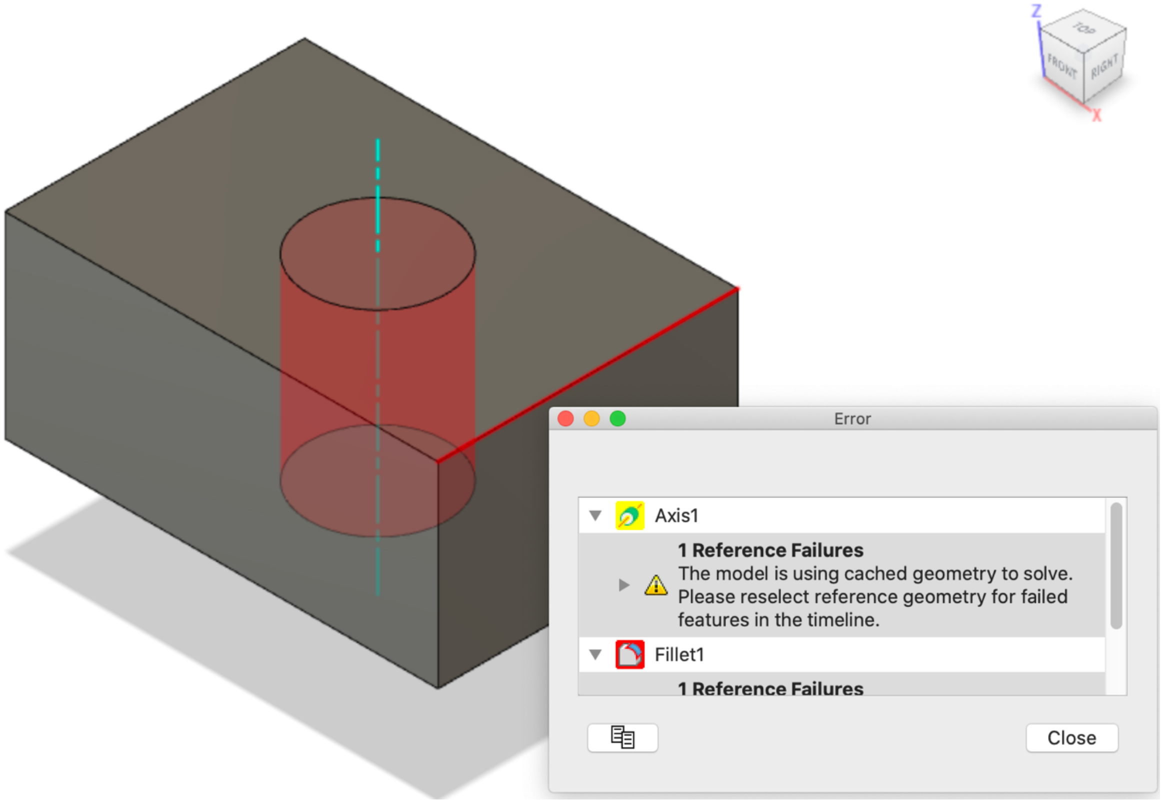
3.2.2 There are Two Different Kinds of Geometry Dependency

Fusion has two basic types of geometry reference or dependency: "Geometry" dependencies, and "Topology" dependencies. Geometry dependencies mean that the consuming feature is reliant on the geometry of the selected entity. Examples of this include: Offset Workplane (needs the plane of the selected face, not the face itself), Project Edge in Sketch (needs the underlying curve of the selected Edge). Topology references, on the other hand, need the actual Edge, Face, Body, Vertex, etc to succeed. Features such as Fillet, Shell, etc do not operate on the geometry of the item, but directly on the item itself. You cannot Fillet a line or an arc – you must fillet a model edge directly, because the edge will be consumed by the Fillet, adjacent faces will be modified, etc.



Geometry reference failures result in Warnings, Topology reference failures result in Errors. Herein lies a fundamental concept in understanding the difference between an Errors and Warnings. A failure in a geometry reference will result in a Warning, while a failure in a topology reference will result in an Error.

Why the difference? Why is one type of failure a Warning, and the other is an Error? Because a geometry reference automatically caches the geometry on each successful compute. Then, when a Matching failure happens, that cache can be used to compute the feature. So, for example, if a geometry reference for an Offset Workplane fails, that feature can cache the plane definition used on the last successful compute to continue to produce some result. It is important to realize that, though a result has been computed, it might be an <u>incorrect</u> result, in that it is no longer connected to the geometry. However, Fusion cannot cache a Topology reference. You need the edge itself in order to compute a Fillet. You need the Face/Body to shell a body. So, there is no way to recover from this kind of failure, hence, an Error is reported.



Axis1

1 Reference Failures
The model is using cached geometry to solve. Please reselect reference geometry for failed features in the timeline.

Fillet1

1 Reference Failures

3.2.3 Each Feature May Handle Matching Errors Differently

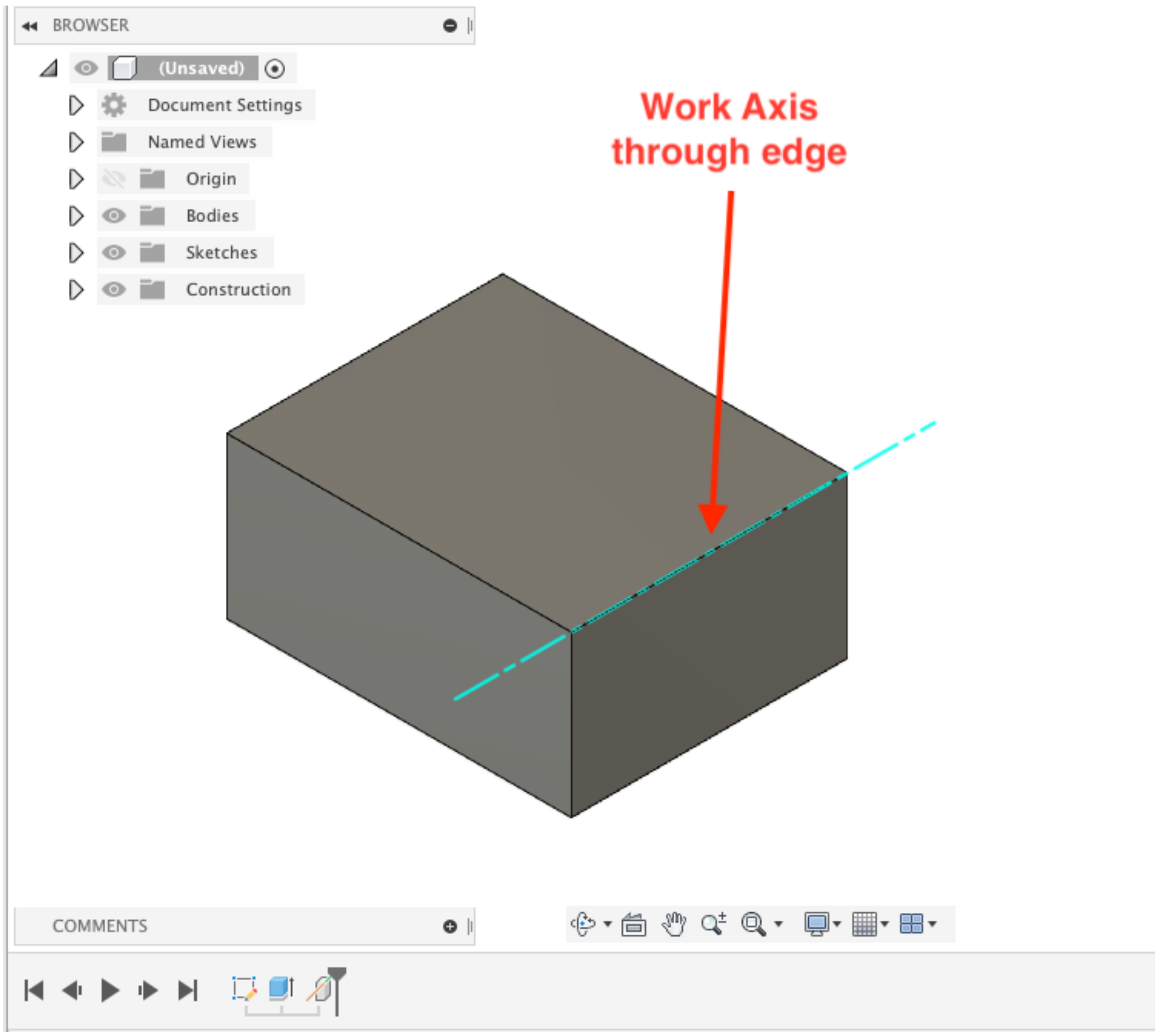
When a Dependency Error happens, the effect on your design will depend on the feature involved. While most features react pretty predictably to a Matching failure (a Geometry reference failure causes a Warning, a Topology failure causes an Error), there are some important exceptions that are worth discussing. One is Fillet/Chamfer. These are examples, as we discussed above, where the feature tracks multiple entities. However, if only <u>some</u> of these references fail, the feature can continue to compute those edges that do match correctly. So, even though these are Topology references, a failure does not mean an Error will result – as long as there are <u>some</u> edges to Fillet, a matching failure will only result in a Warning.

Another example of per-feature behavior differences is how multiple candidates are handled. In some cases, if a feature expects a single matched entity, and it discovers more than one exist, an Error will occur. However, other features can handle multiple candidates differently. For instance, in Fillet, if an edge is split, so two candidates are returned where one is expected, Fillet says “so what? I can just Fillet both of those edges”, so this is not even returned as a Warning state. In other cases, it will depend on the nature of the feature. Some features (such as Offset Workplane, or WorkAxis through Edge) will just pick the “first” candidate and ignore any others.

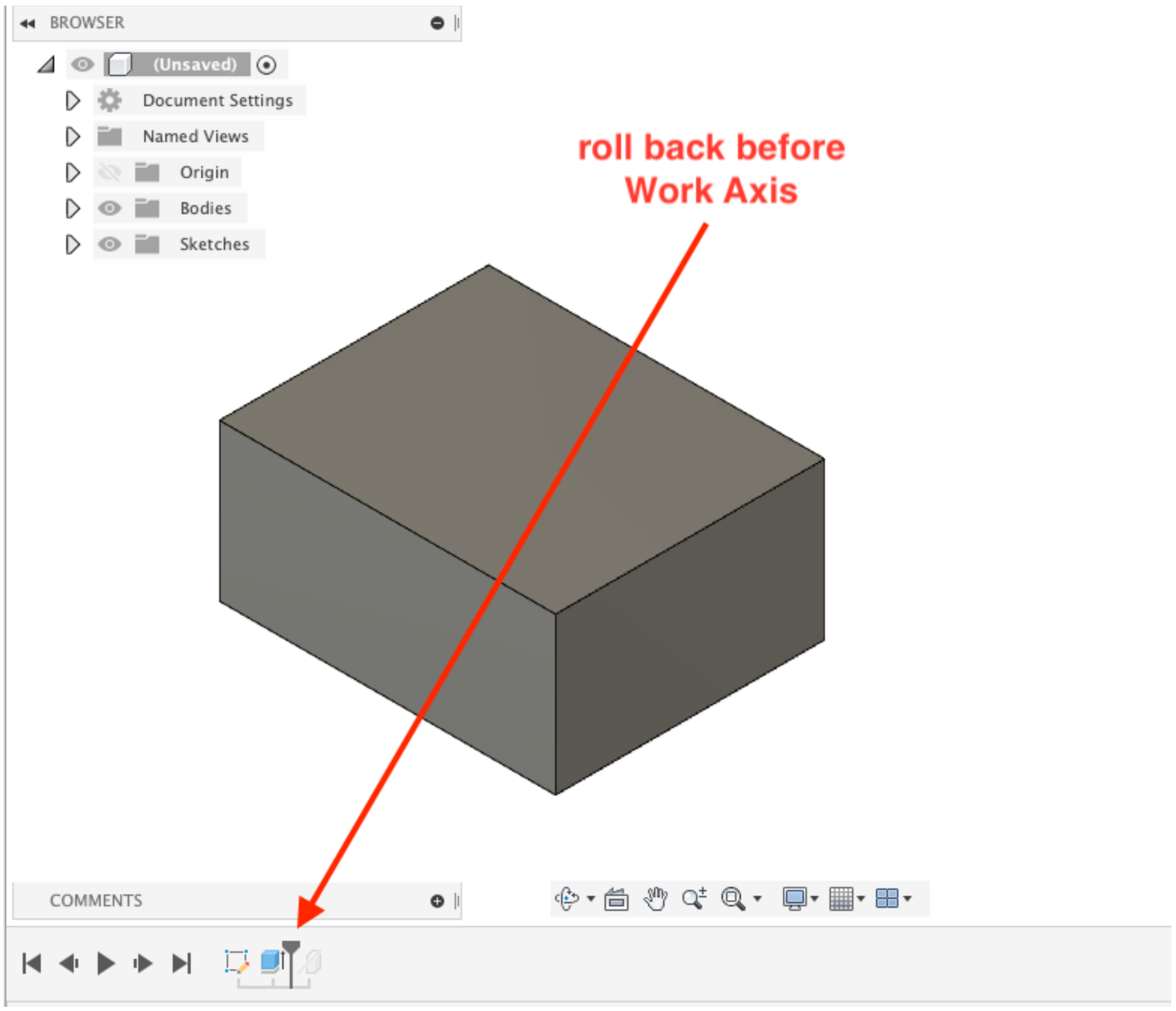
3.2.4 Compute Early

(dependencies compute early to prevent failures)
Why does this sequence of edits not result in an error?

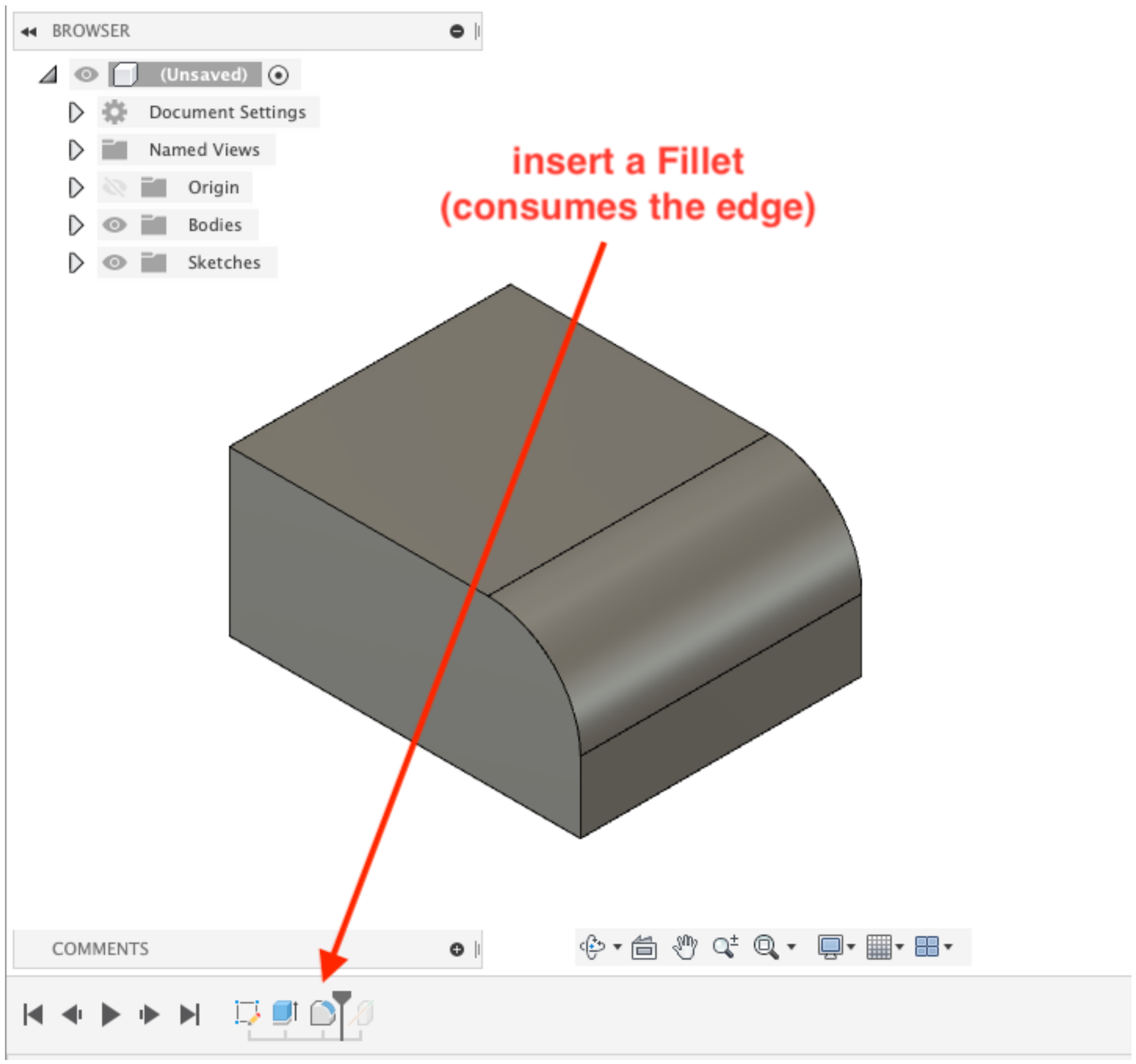
Starting point: Simple design – one sketch, one Extrude, and an Axis Through Edge on one of the edges:



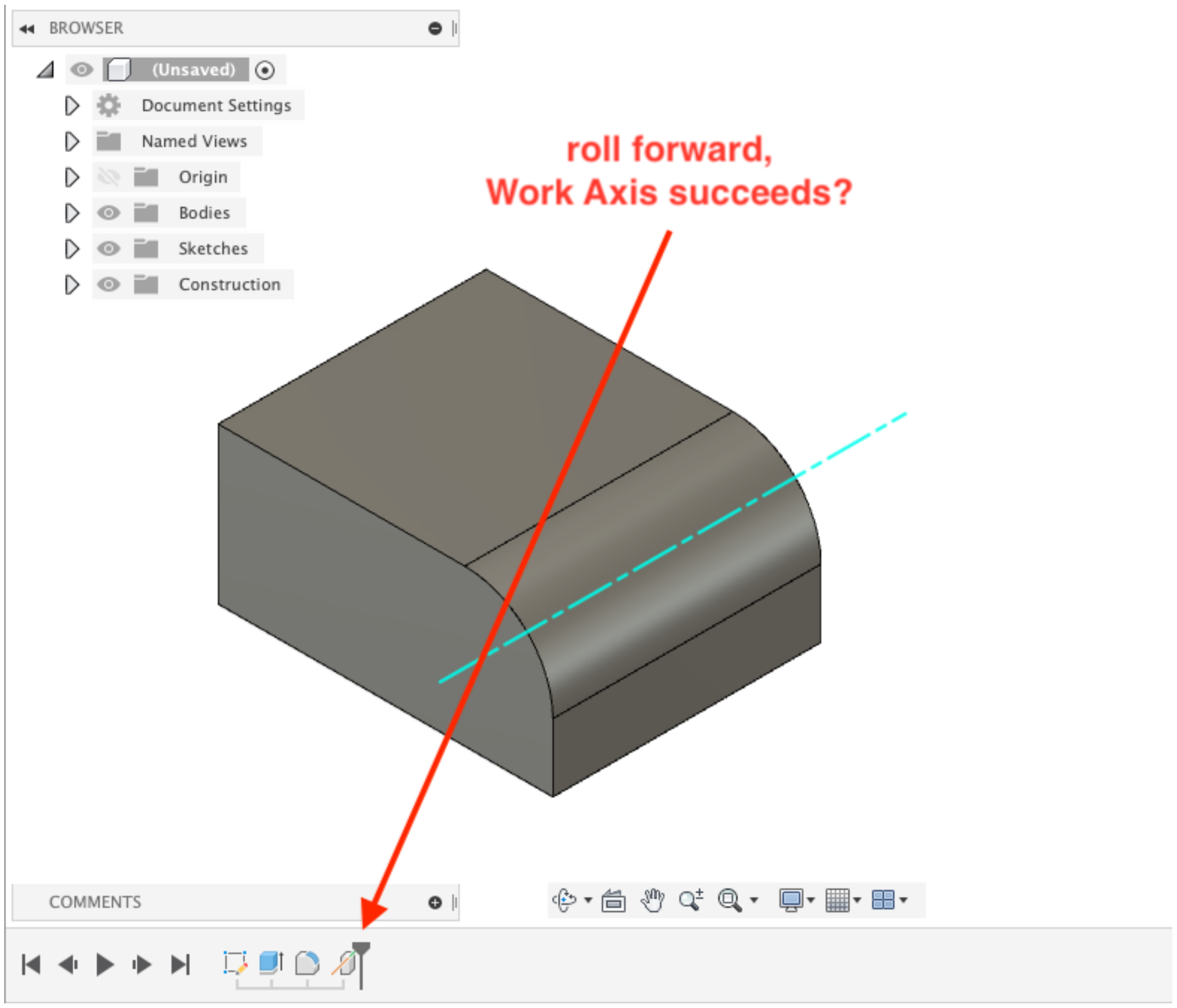
Next, roll back to before the Axis feature:



Insert a Fillet of the edge that the Axis uses:



Roll Forward:



In this example, we've rolled back the timeline to before the Workaxis feature, and then introduced a Fillet that completely removes that edge. The edge is a Geometry reference, so if it fails, we would expect the Workaxis feature to produce a Warning, right?

The reason why this can succeed is through something we call "compute early". For these references, Fusion tries to Match them as early as possible in the Timeline. So, even though we roll back, and insert a Fillet which completely consumes the edge that the Workaxis needs, we don't even get a Warning. The Match is done after the Extrude, but before the Fillet, so even though the Workaxis computes later, it can still succeed!

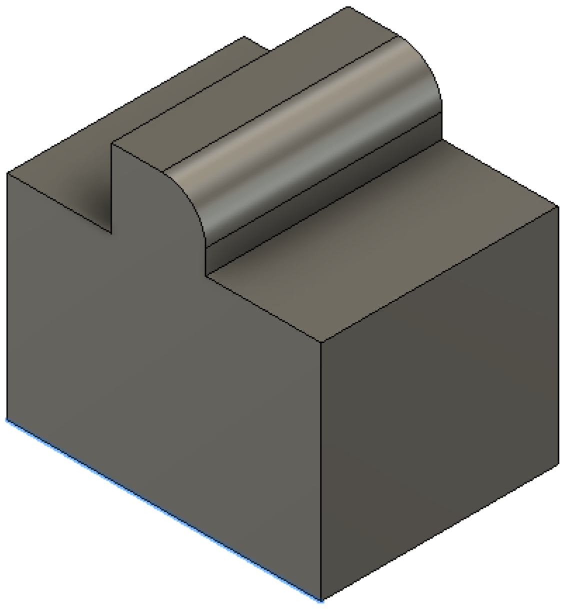
3.3 CAD Naming and Matching

The area of CAD that deals with resolving design references is called "Naming and Matching". This section explores the basics of this area of software, mostly so you have some background as to why you get some of the errors and warnings that you get.

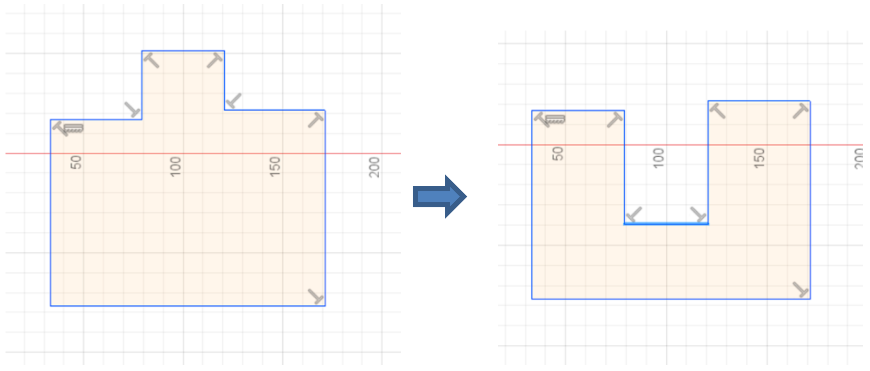
3.3.1 Problem description

When you fillet an edge in Fusion, then go back and make an edit to the design, Fusion needs to make sure it finds the “correct” edge to fillet when the change computes across the model.

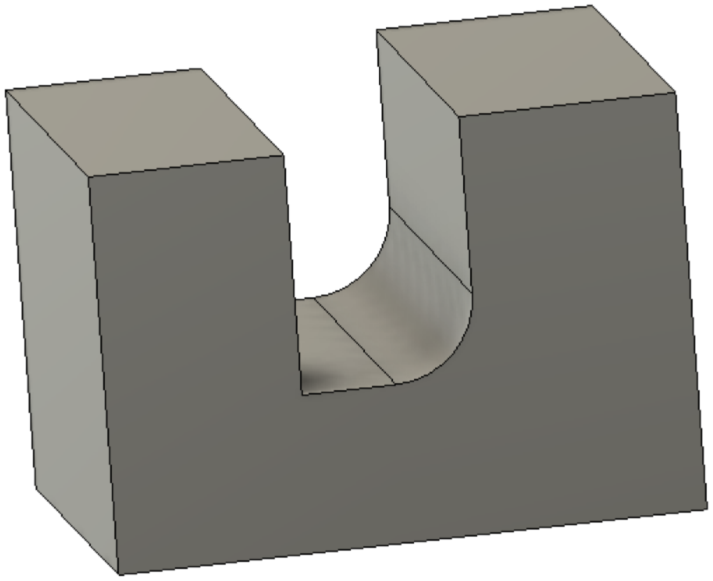
For example:



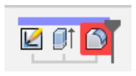
If, after an edit to the sketch for this model:



The correct answer, somewhat surprisingly, is:

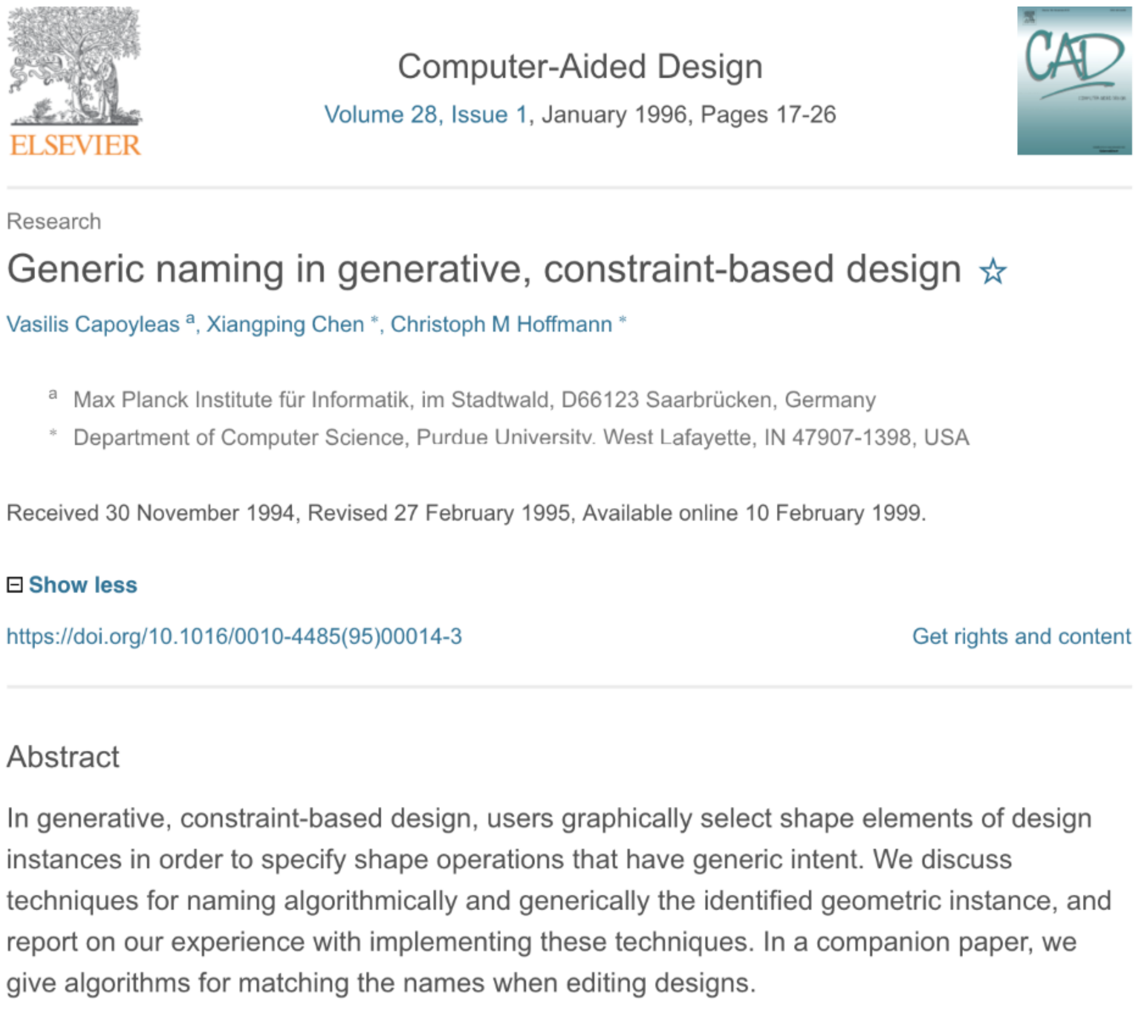


If the edit has changed the edge so that it cannot be identified, it is shown on the timeline icon as reference failure, and depending on the feature, can cause a yellow warning or a red error.



In this example, the fillet has lost its reference

This entire area of CAD: resolving model references and managing failures when the match fails is called "***Naming and Matching"*** in the CAD software world.
All modern CAD systems are based on this research paper from 1996:



Computer-Aided Design

Volume 28, Issue 1, January 1996, Pages 17-26

Research

Generic naming in generative, constraint-based design ☆

Vasilis Capoyleas [a], Xiangping Chen [*], Christoph M Hoffmann [*]

[a] Max Planck Institute für Informatik, im Stadtwald, D66123 Saarbrücken, Germany

[*] Department of Computer Science, Purdue University, West Lafayette, IN 47907-1398, USA

Received 30 November 1994, Revised 27 February 1995, Available online 10 February 1999.

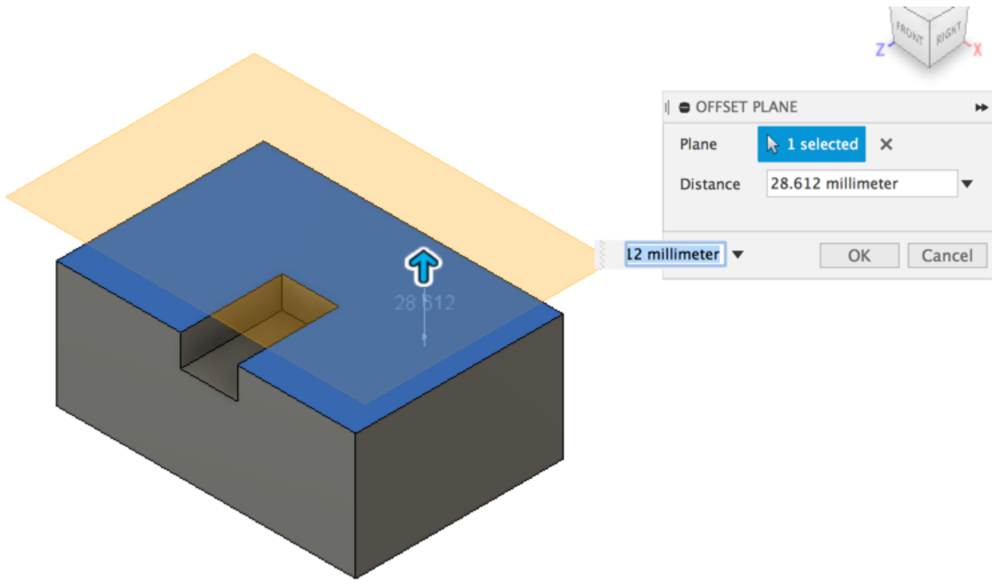
⊟ Show less

https://doi.org/10.1016/0010-4485(95)00014-3 Get rights and content

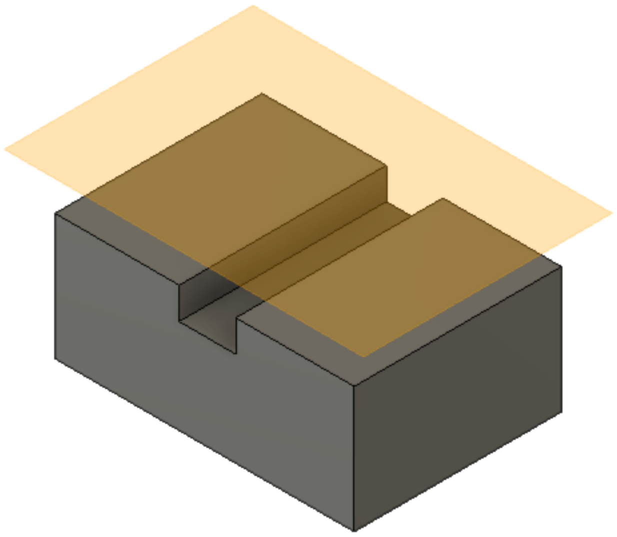
Abstract

In generative, constraint-based design, users graphically select shape elements of design instances in order to specify shape operations that have generic intent. We discuss techniques for naming algorithmically and generically the identified geometric instance, and report on our experience with implementing these techniques. In a companion paper, we give algorithms for matching the names when editing designs.

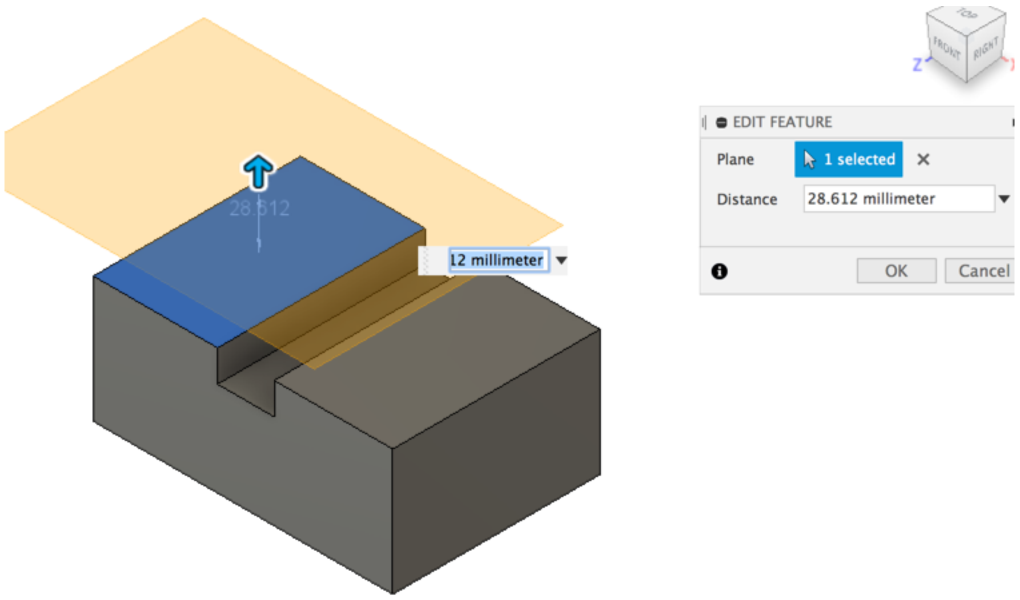
Here's another example. Say you have the design below and an offset plane from the selected face:



Then assume that the slot is edited so that the face is split:



Which face is the “correct” match here? This is the one that is matched:



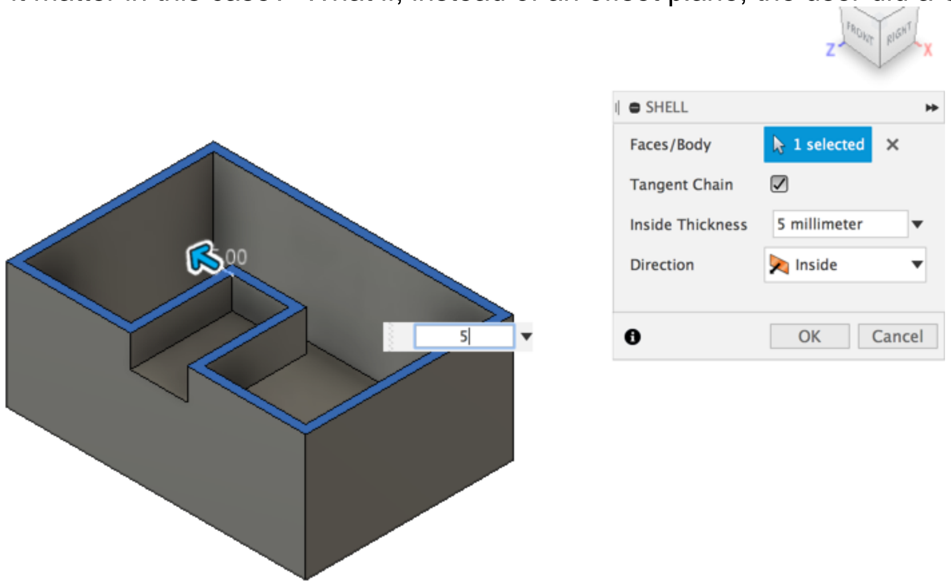
EDIT FEATURE

Plane: 1 selected

Distance: 28.612 millimeter

OK Cancel

Does it matter in this case? What if, instead of an offset plane, the user did a Shell:



SHELL

Faces/Body: 1 selected

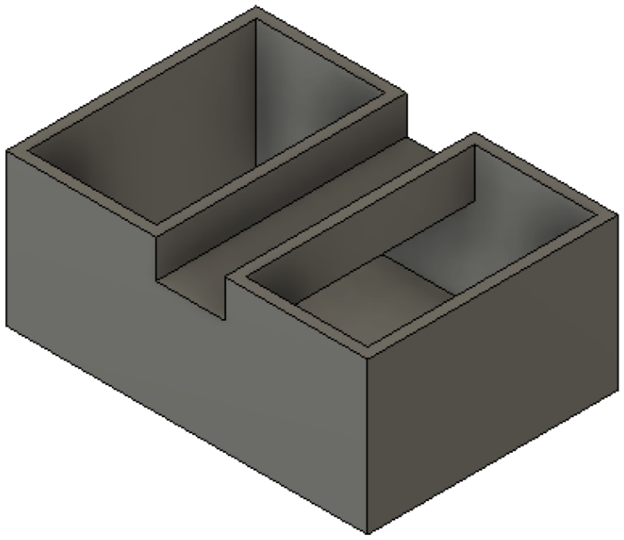
Tangent Chain: ☑

Inside Thickness: 5 millimeter

Direction: Inside

OK Cancel

(Above: The top face is selected as the face to remove for Shell)



The Shell and the Workplane handled this case of multiple match candidates differently, yet both give a “reasonable” result. This flexibility allows Fusion to have each feature respond in a way that is optimal for that feature.

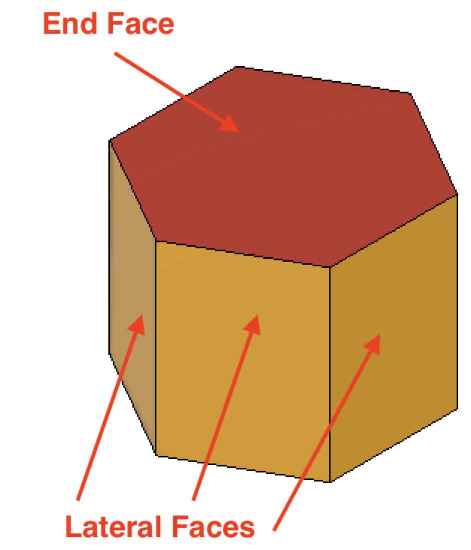
3.3.2 How does this help me in Fusion?

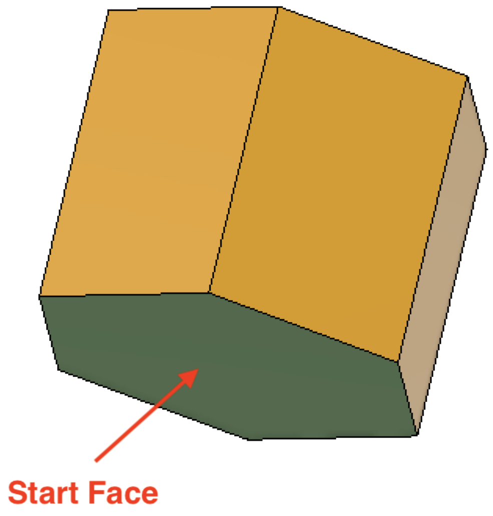
How does Fusion track and identify model references, how should you use that information to get the results you expect, avoid failures, and if failures happen, and how can you fix them?

3.3.3 Fusion Naming – the first half of Naming/Matching

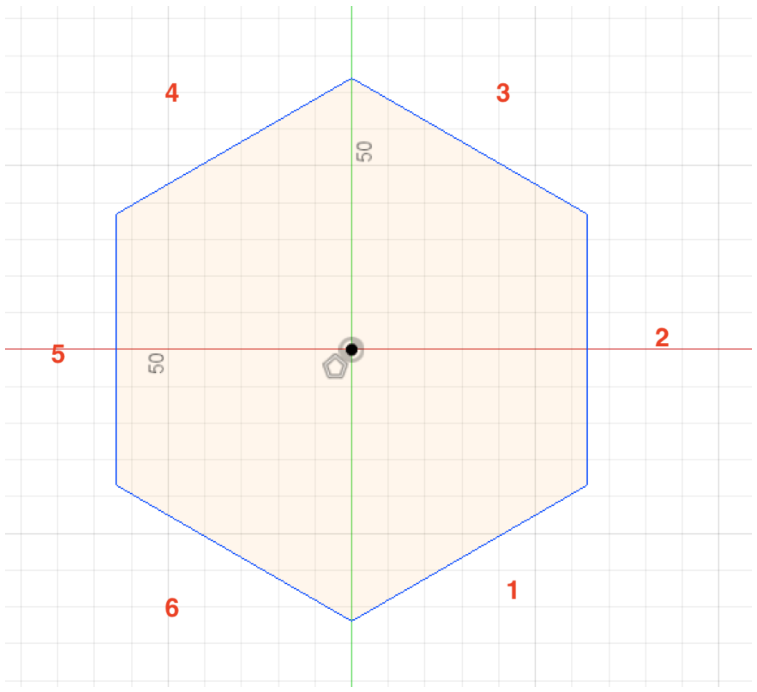
Fusion “tags” model faces with information from the feature that created that face. Tags are attributes on the geometry that you cannot see, but we can.
For example, for Extrude, there are 3 categories:

- Start face
- End face
- Lateral faces

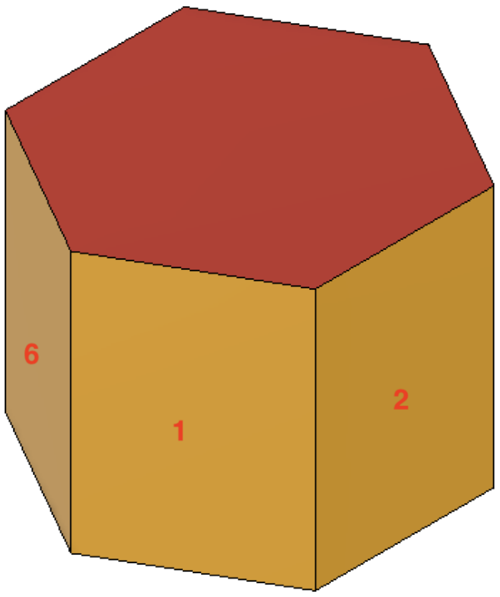




If you're all still awake and following along, you'll no doubt wonder: "OK, I get how you can track the start and end faces, but there are 6 lateral faces here, how can you tell them apart?"
Lateral faces are tagged using an ID of the sketch curve that is the progenitor of that face. So, if we look at the sketch for this polygon model, it's easy enough to see that each line of the polygon in the sketch can have a different ID

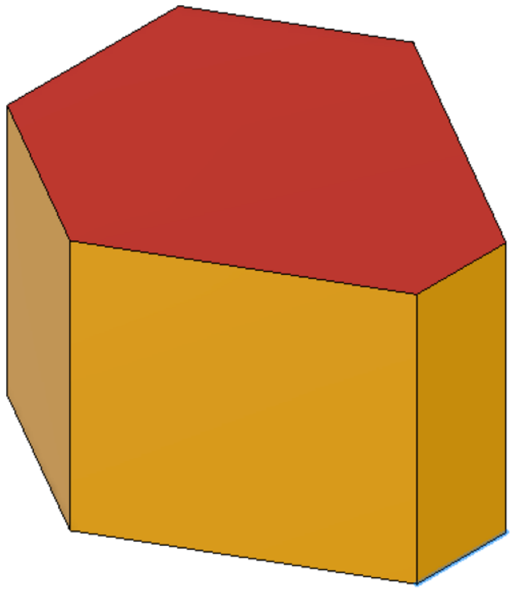


This ID is added to the face tag of the lateral faces of the Extrude:

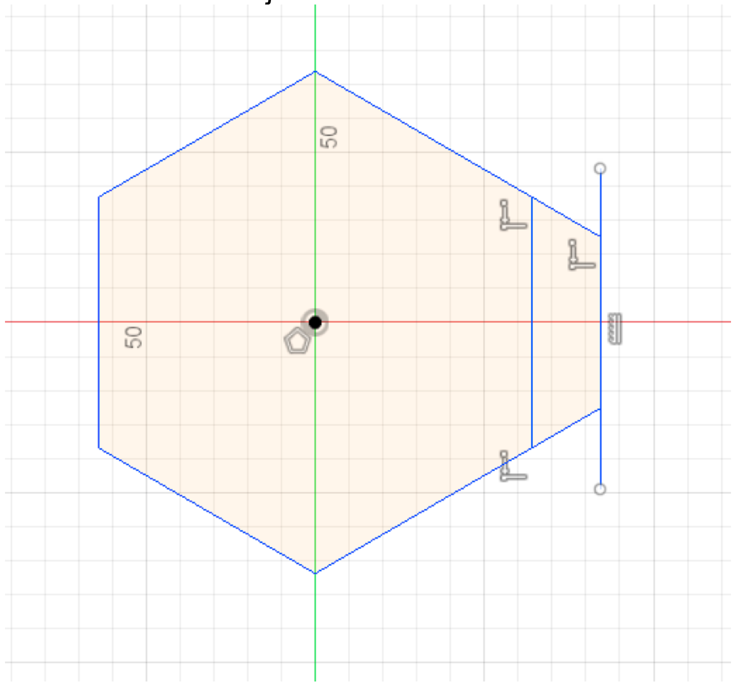


Which is how Fusion distinguishes one lateral face from the others.

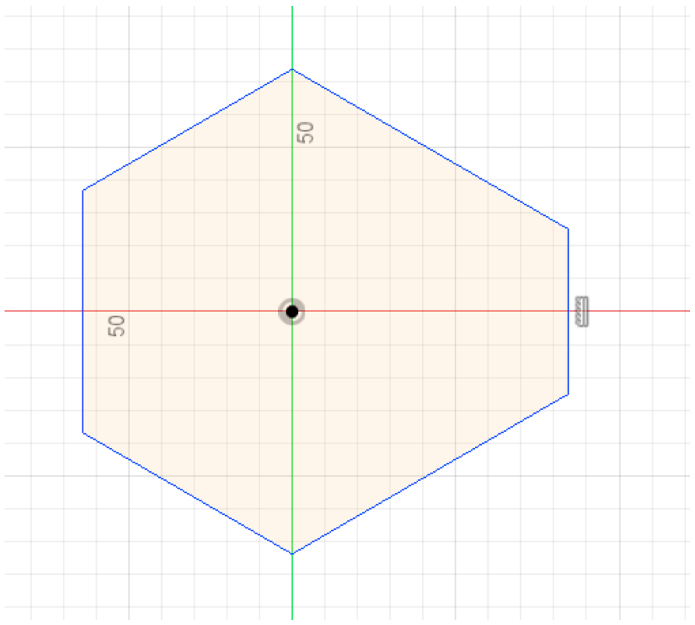
<u>So, what does this tell us?</u> The main lesson here is: You should be careful when editing sketches. Say, for example, that you wanted to change this model to one that is not a regular hexagon – you wanted to extend two faces to make one face narrower:



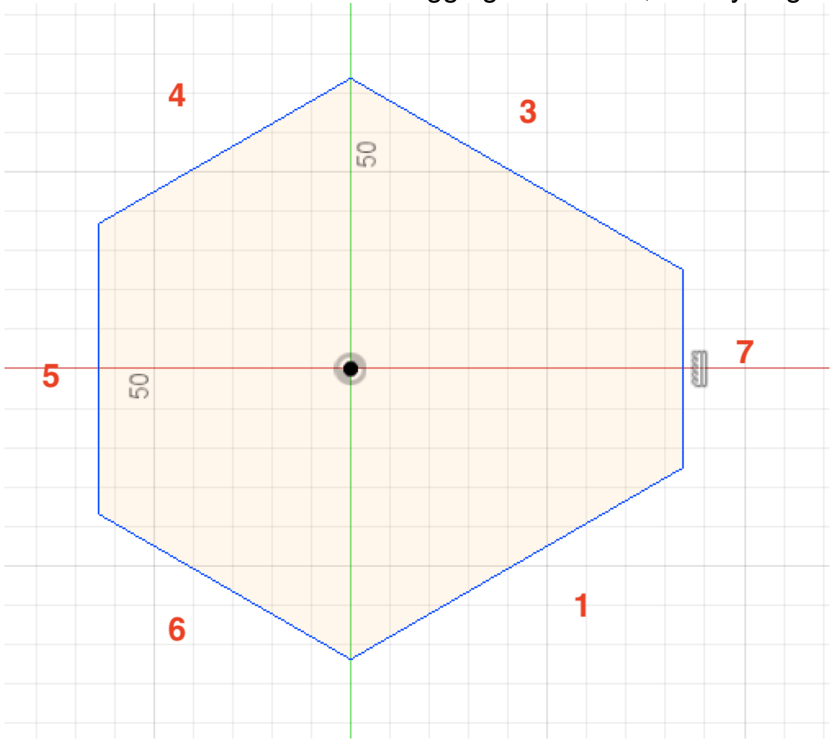
There are many ways to edit the sketch to achieve this result. You could add a new line, and use Extend to extend the adjacent lines:



Then, trim off the dangling pieces, and delete the original line:



But, if you think about it in terms of our tagging discussion, what you get is:



Because that new vertical line to the right is a <u>new</u> line. So, any reference to lateral face 2 will be lost.
<u>Instead</u>, if you break the polygon constraint (it will get broken anyway), and just drag the vertical line to the right, it will maintain its identity as curve 2, and any downstream reference to that face will get preserved.

Moral: When editing a sketch, try as much as possible to preserve the geometries in that sketch. Don't *delete* and *re-create* sketch geometry. Preserving the original sketch object preserves the feature that uses it.

So, this is how Fusion tracks faces, what about edges/vertices?

- An edge is just the intersection of two faces
- A vertex is just the intersection of two or more edges
- So, everything comes down to face tracking, which comes down to sketch identifiers.

4 Error Debugging/Fixing Strategies

I know we spent a lot of time on the theory of Errors and Warnings in Fusion, but you are all thinking: "but I came here to learn to fix the errors, not just understand them". So, now we'll move on to the fixing part.

4.1 General Strategies

These general strategies apply to all design errors

4.1.1 Fix Errors/Warnings When They Happen

Yeah, I know we're repeating ourselves, but this is important. The sooner you fix these, the better off you will be. We'll add one more recommendation here: understand the error, then fix it. That is, it will help you long term if you can convince yourself that you understand why the error occurred. This will help you both fix the error, and also prevent it from happening again. Yeah, you can always Undo to get rid of the error, and sometimes that itself is a good strategy. But, it's better if you can understand "OK, that error happened because the edge that this feature is tracking was eliminated by this edit to another feature I just made".

4.1.2 Fix the First Error in the Timeline First

You made a change, and it caused 35 Errors/Warnings in your timeline, and you are flipping out. Don't worry. As we discussed above, errors will cascade, depending on the type of error that is happening. Often, the sequence of errors flows from one or two direct errors at the beginning of the chain. So, go to the first error that shows up in the design, understand and fix that one, then see. Often, most of the downstream errors will be fixed by just fixing the first problem.

4.2 Debugging Sketch Errors

Sketch failures come in 3 flavors:

1. Sketch Plane failures
2. Sketch solve failures
3. Projected geometry failures

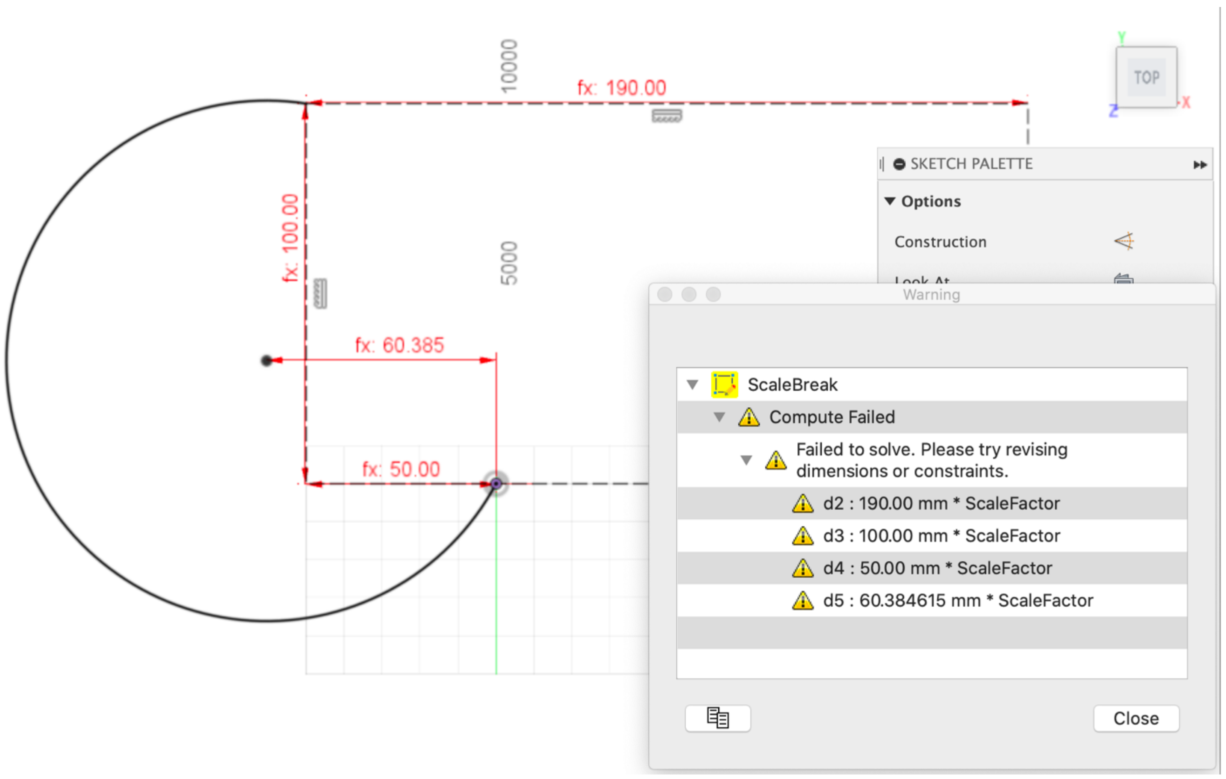
We'll look at each of these separately

4.2.1 Debugging Sketch Plane Failures

In this type of failure, the plane which was selected cannot be found. This is a Matching failure. Because this reference is a Geometry dependency, this should always be a Warning, not an Error. The only method for fixing a Sketch Plane Warning is Redefine Sketch Plane. This command allows you to choose a replacement plane for the sketch. **Note:** The Redefine Sketch Plane command can potentially alter the sketch coordinate system in unexpected ways. The sketch CS can rotate, or even flip completely. Also, projected geometry in the sketch will be converted to fixed sketch geometry (and links to the original source of the projection will be lost). If you select a new body face for the sketch, and if you have autoproject on, this command will create new projected edge geometry from that face. Be aware of these limitations when you use this command. We have a project lined up to improve the behavior of Redefine, but it has not yet been started.

4.2.2 Debugging Sketch Solve Errors

This one is a bit more tricky. Usually, there is a conflict between some constraint or sketch dimension that is at the root of the failure.



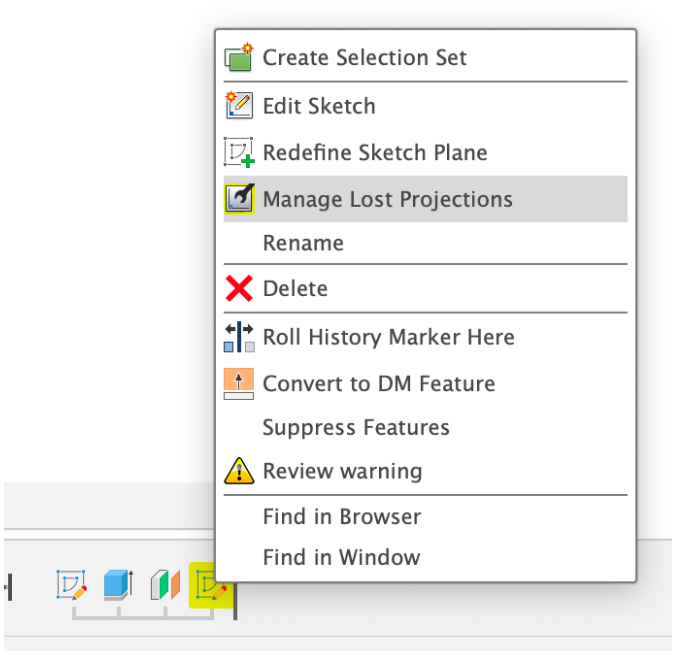
ScaleBreak
Compute Failed
Failed to solve. Please try revising dimensions or constraints.
d2 : 190.00 mm * ScaleFactor
d3 : 100.00 mm * ScaleFactor
d4 : 50.00 mm * ScaleFactor
d5 : 60.384615 mm * ScaleFactor

The error dialog in this case identifies all the conflicting dimensions or constraints. Usually, removing one of them will allow the rest to succeed.

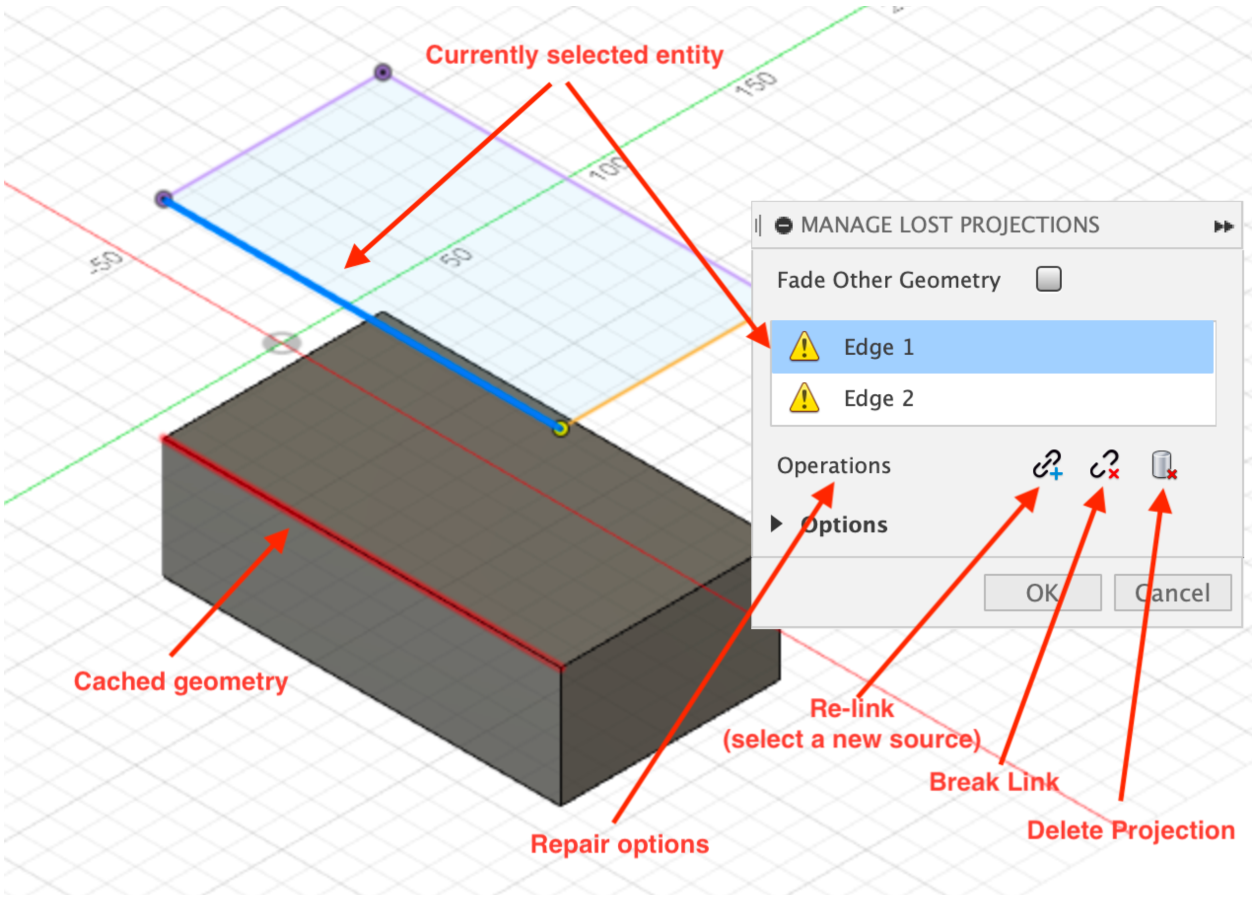
4.2.3 Debugging Projected Geometry Failures

This is probably the most common failure in Sketch. Some model edit has caused the object being projected to be lost (a Matching failure). Because projections are all Geometry Dependencies, these are all Warnings.

The good news here is that fairly recently, Fusion added the Manage Lost Projections command, making this very common area of failure one of the most convenient to repair:



This command helps you to repair sketch projections with a Matching failure.



This command shows all the failed projected sketch geometry in the dialog. When you select one row, the corresponding sketch item will be selected in the graphics area, and the cached geometry from the Geometry Dependency (see above) is also highlighted, to give some feedback as to what the original geometry was, to make it easier to select a new candidate.

In the "Operations" area of the dialog there are 3 repair options:

- Re-Link – this allows you to select a new source for the Projection, and maintains all downstream relationships (dimensions and constraints in the sketch, and even further downstream operations based on sketch geometry ID – we'll discuss that later)
- Break Link – this allows you to convert the failed projected sketch geometry into a normal sketch item, just like the Break Link command in sketch mode. This will also clear the failure for this projected item
- Delete Projection – this just deletes the projected sketch item, if you decide that it is no longer needed

4.3 Debugging Modeling Failures

This section covers some tips to help you recover from failures in features from the Model and Surface tabs of the toolbar. There are two main types of failures in these features:

1. *Geometry* or kernel modeling errors. These failures are caused when the basic solid modeling operation fails, usually because the geometry is too complex for the operation to succeed.
2. *Dependency* failures. All modeling features have geometry Dependencies, and as described above, any of them can fail due to upstream changes.

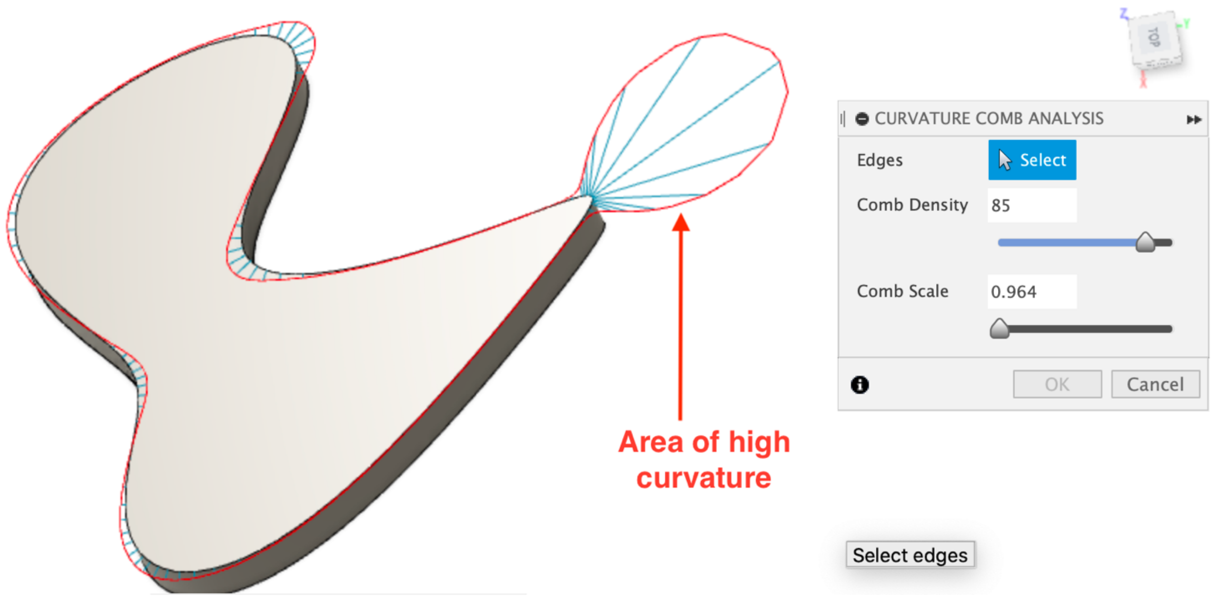
4.3.1 Geometry Modeling Errors

Sadly, there are no general rules that apply here. Each feature is different, has different failure modes, and you need to learn over time what failures are caused by what bad inputs. To be brutally honest, Fusion itself does not offer much help here, either, because the error messages are often cryptic and unhelpful. We're working on that, but progress is slow… This topic itself could probably be expanded into a multi-day workshop, so we will have to limit ourselves to the high points here, unfortunately.

4.3.1.1 Fillet Geometry Errors

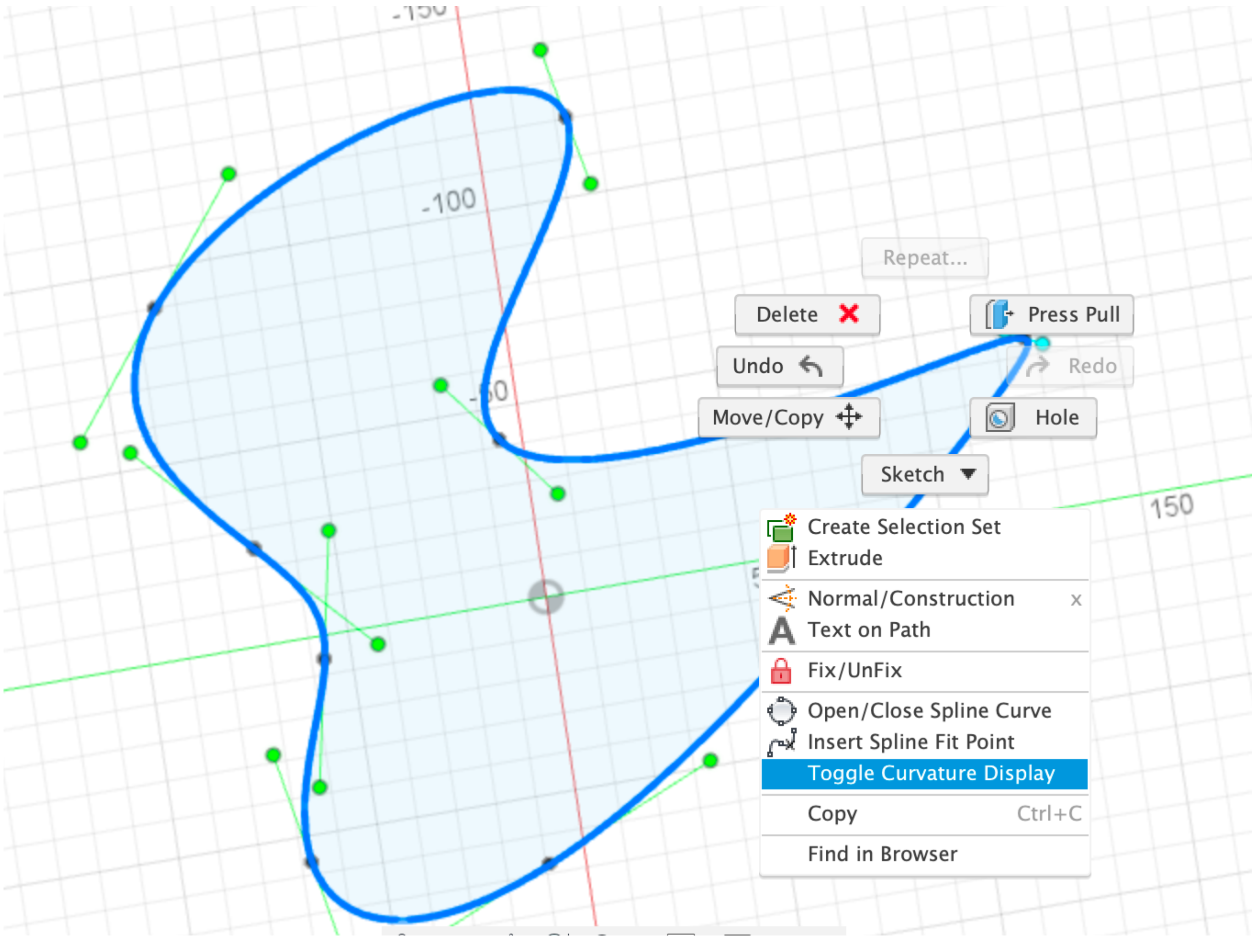
The two main causes of Fillet geometry failures are: areas of high curvature in the edge being filleted, and near-tangencies in the edges selected. Both of these will cause the Fillet kernel algorithms to fail.

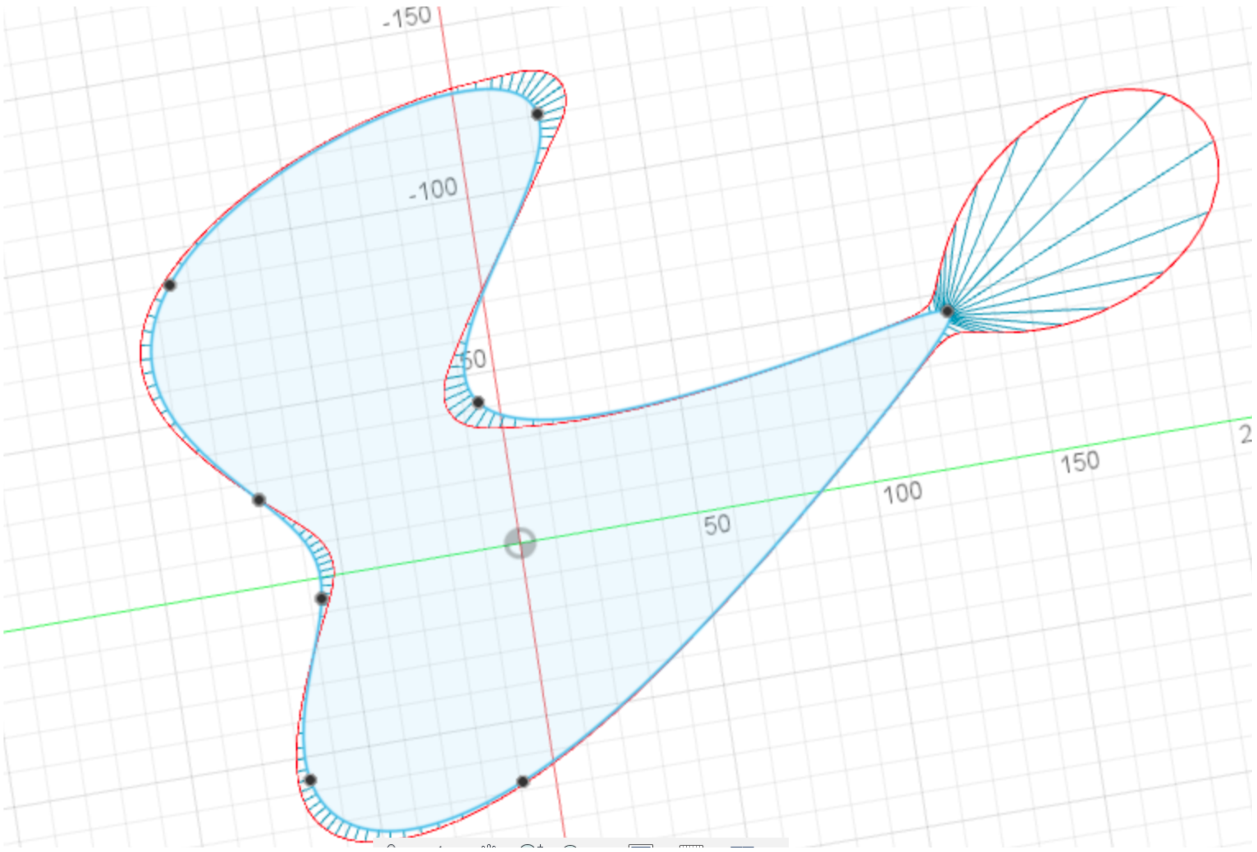
Areas of high curvature. If you have Spline edges which have a small radius of curvature (large amount of curvature), Fillet is likely to fail. You can see the curvature by creating a Curvature Comb analysis on the edge:



Edges: Select
Comb Density: 85
Comb Scale: 0.964
OK Cancel

The only way to fix this is to reduce the curvature. Go back to the original geometry which created this edge (usually a sketch curve). You can use a similar analysis within the sketch to look at the source curve (Toggle Curvature Display) to show curvature combs in a sketch:


-150
-100
-50
150
Repeat...
Delete
Press Pull
Undo
Redo
Move/Copy
Hole
Sketch
Create Selection Set
Extrude
Normal/Construction
x
Text on Path
Fix/UnFix
Open/Close Spline Curve
Insert Spline Fit Point
Toggle Curvature Display
Copy
Ctrl+C
Find in Browser



You can use normal sketch techniques such as constraints, dimensions, dragging or constraining spline handles, etc. Basically, you just need to smooth out the curve as much as you can.

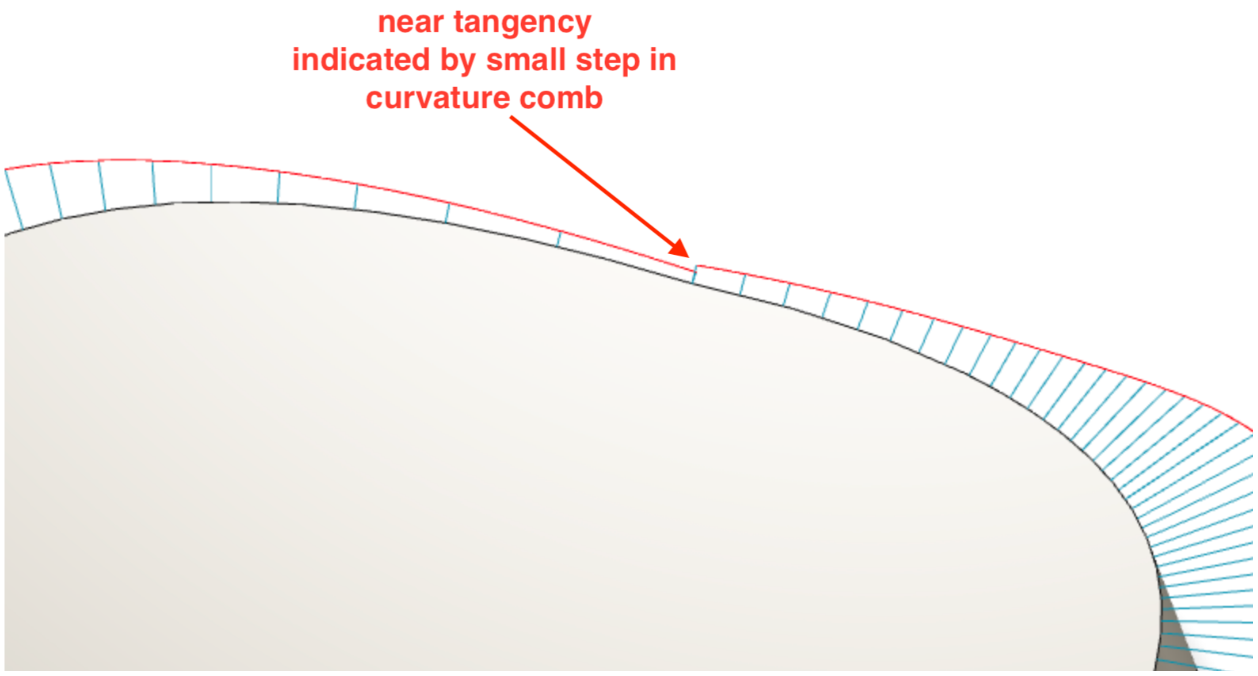
If features such as Loft or Sweep were used to create this edge, you may have to use these techniques on rail curves, or make other edits to the feature (e.g. takeoff weight in Loft) to reduce the edge curvature in areas where it is high.

Near Tangencies in the Edge Selection. This one is a bit harder to explain why it produces a failure, so we won't even try. We don't understand the math involved. The important message, here, though, is that it is a real cause of Fillet failure.

What do we mean by "near tangency"? This condition is pretty much what it implies. If two curves are very close to being tangent, but are not actually tangent to the resolution of the modeler, then those curves are considered "nearly tangent". This causes lots of issues with creating the "blend surface" for the Fillet feature.

How do you know that the Fillet fails because of a near tangency? You don't. Unfortunately, that is an example of the insufficient error messaging in Fusion – this cause will not appear in any error message. Fillet geometry errors will all appear the same: "this fillet/chamfer could not be created at the requested size…". So, just look for the kinds of conditions described in this paper to fix the problem.

How do you know whether you have a near tangency? The best way is to use the Curvature Comb analysis, as discussed for the high curvature case above. The length of the comb spine indicate curvature at that point in the curve (e.g. the curvature comb of a circle should be constant at all points, and the curvature comb of a line should be zero length). If you see a "step" in the curvature comb where two curves, that is a point of tangent discontinuity. A small "step", therefore, is a "near tangency":



To fix this type of failure, you will need to correct the near tangent condition. Usually, this takes the form of a tangent constraint in a sketch.

Unfortunately, this is often the cause of Fillet failures in extruded text. The reason is that text fonts produce outline curves that are not constrained to be accurately tangent. This is not needed for most uses of fonts (display, printing), but is a problem in a highly accurate application such as Fusion. Sadly, there is not a good answer here – it is usually not practical to explode the text, and go back in and add tangent constraints to text – that can distort the text and is, anyway, too laborious.

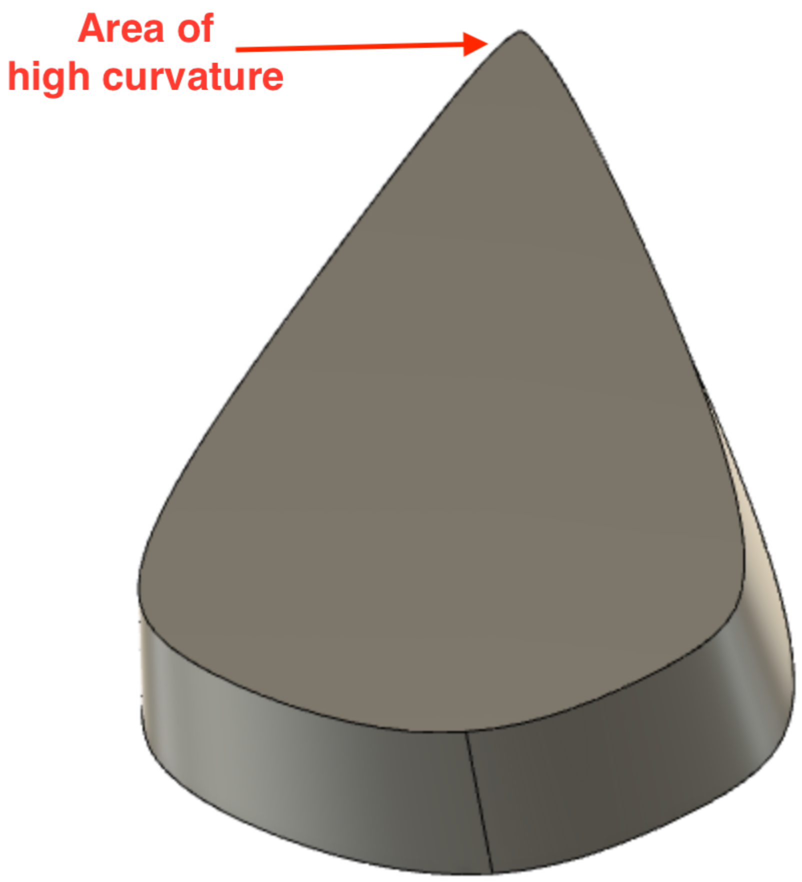
We are currently working on an improvement that will try to automatically fix these near tangencies, but it is a bit experimental, and no guarantees that it will happen…

4.3.1.2 Shell/Offset Geometry Errors

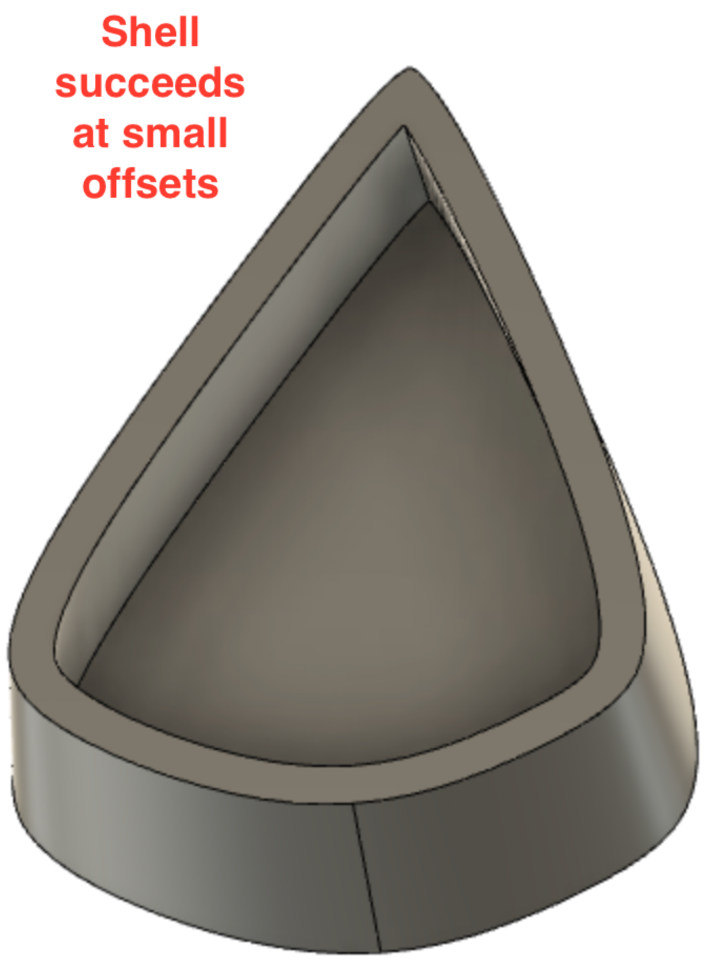
Shell is the next most common source of geometry errors. We've grouped Offset here (both Surface Offset and solid Offset Faces), because they use the same underlying geometry kernel functions – offsetting a set of faces, either inwards or outwards.

The main geometry error in surface offset is self-intersections. In general, areas of high curvature in the surface geometry is what causes these self-intersections. This should be fairly obvious, but it's worth a few minutes to understand.

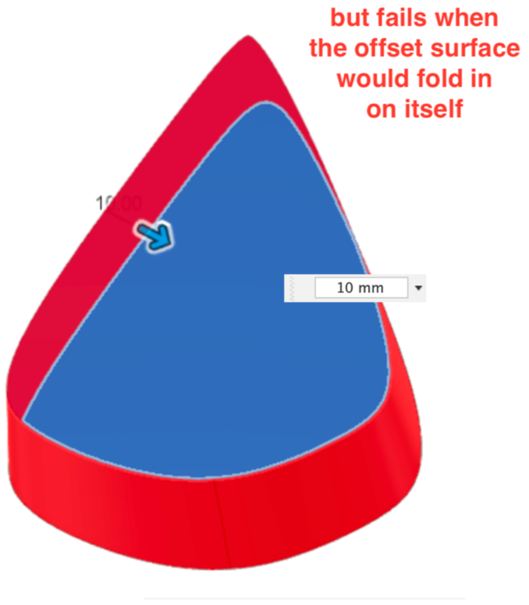
In this simple case, we have a body with an area of high curvature:

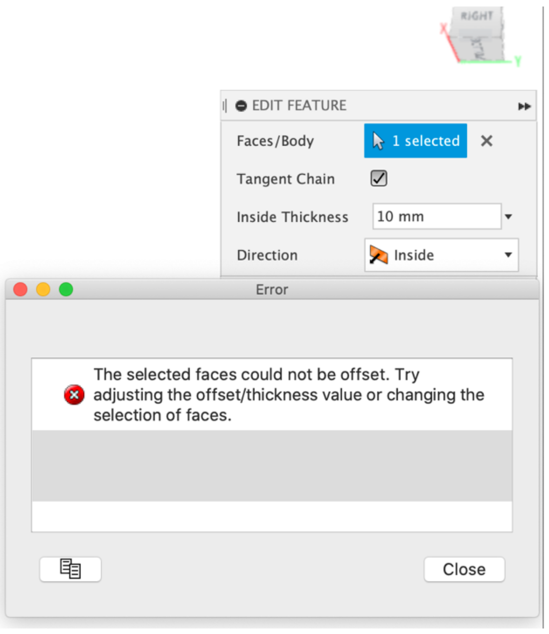


This body can be shelled using some values of offset:



But, at some values, the offset needed for the shell will start to self-intersect, and the shell will fail:




EDIT FEATURE

Faces/Body: 1 selected

Tangent Chain: ☑

Inside Thickness: 10 mm

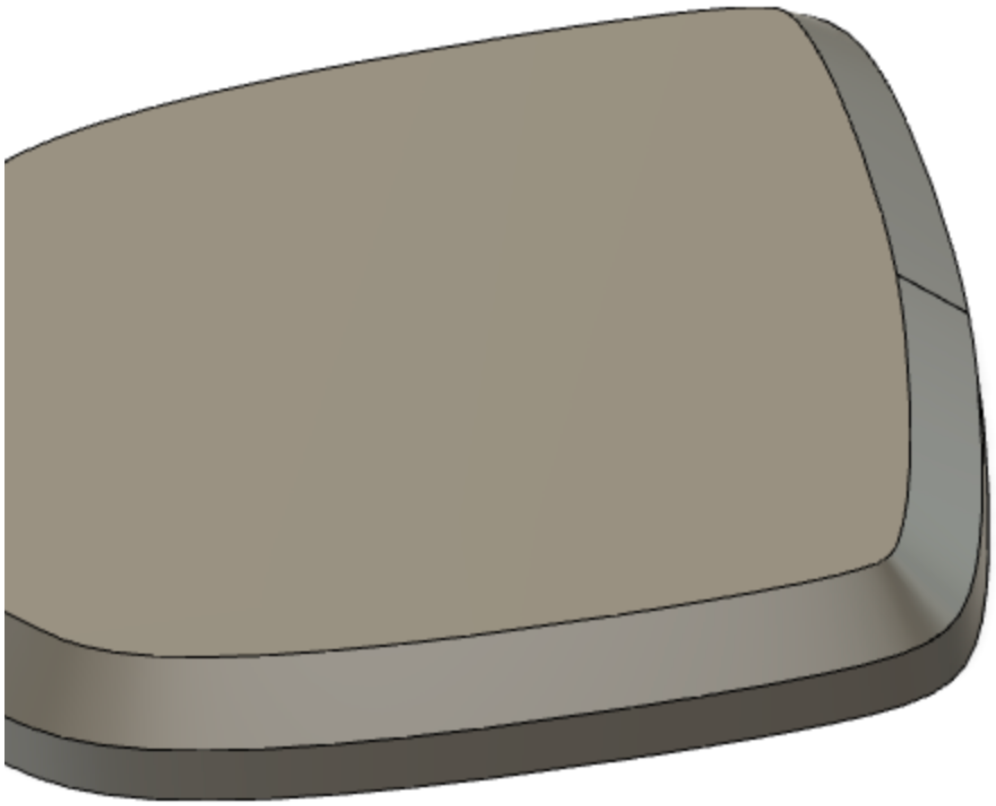
Direction: Inside

Error

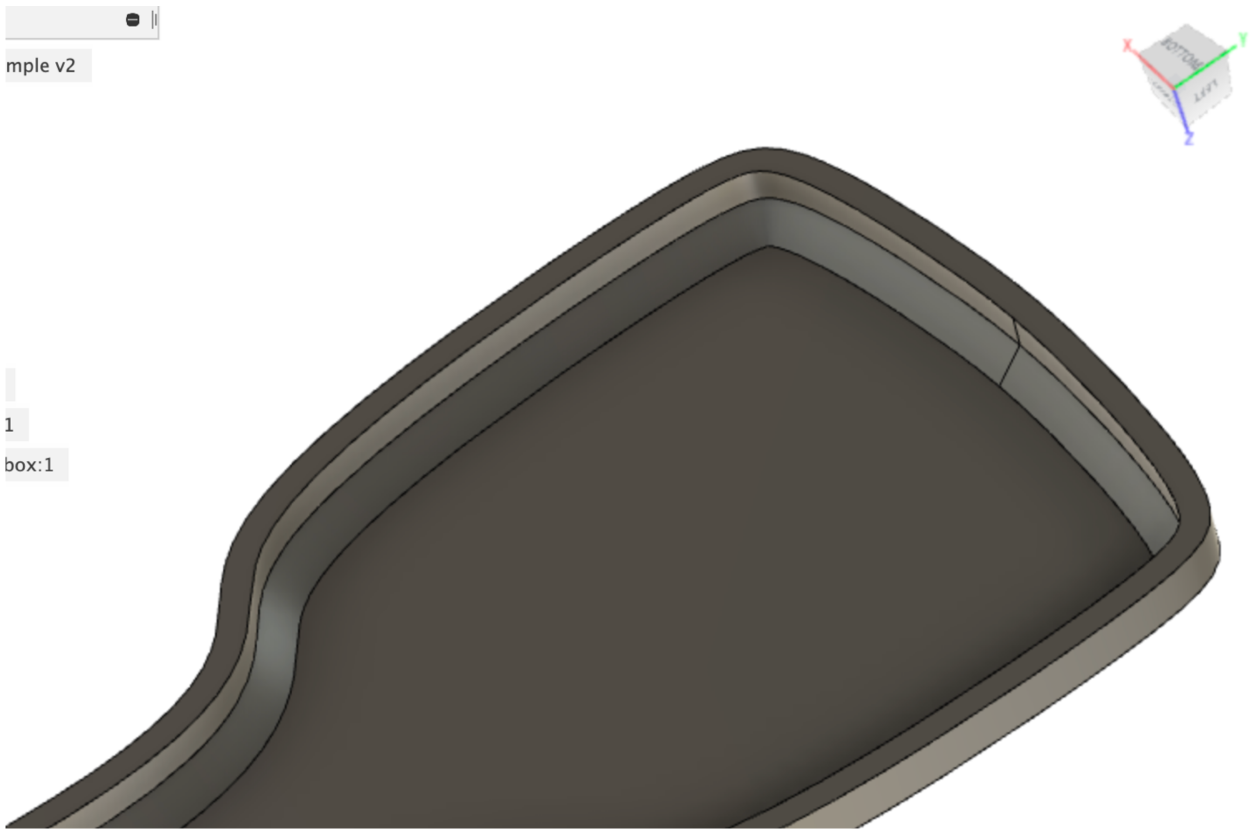
The selected faces could not be offset. Try adjusting the offset/thickness value or changing the selection of faces.

Close

Here is another example from the Fusion forum, in this thread: splines shells and chamfers. This one is interesting because the root cause of the failure is an interaction between a Shell and a Chamfer that is at the root of the problem. Here is the model:

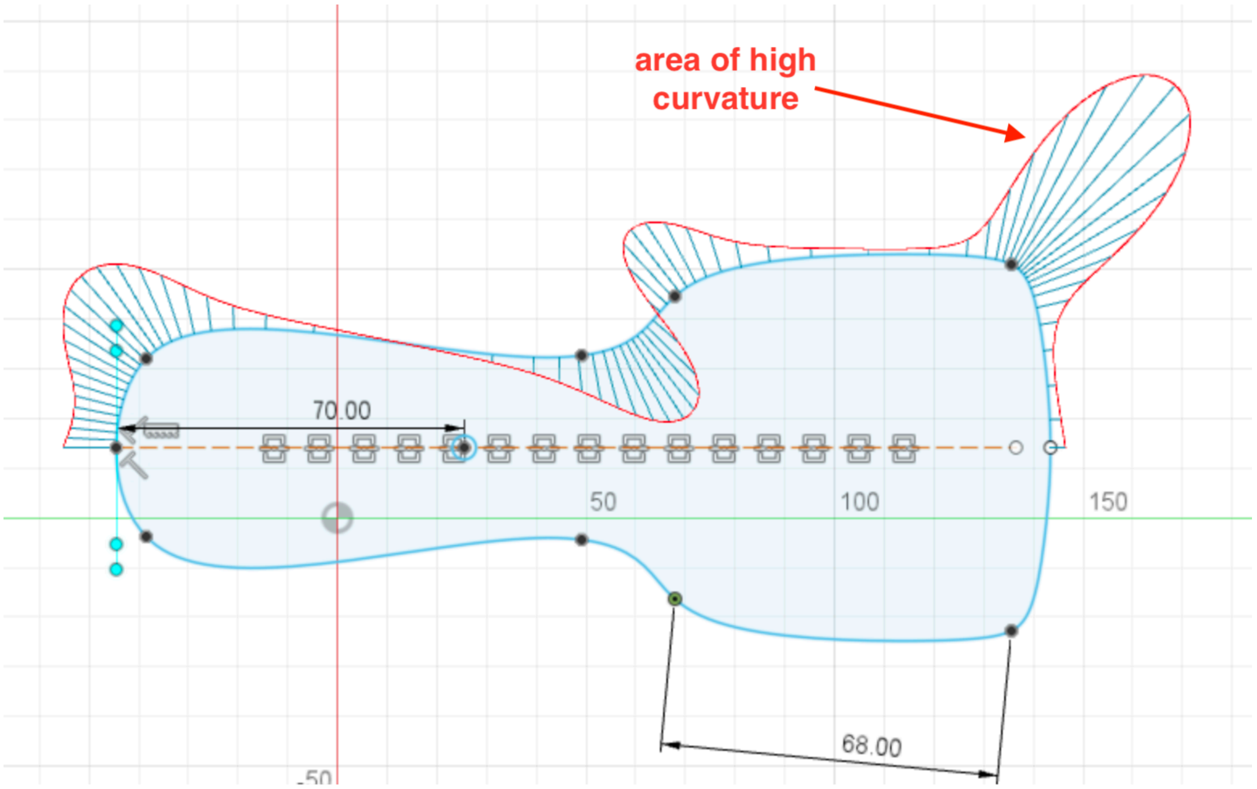


As you can see, the geometry is pretty simple – a spline profile, an extrude, and a chamfer. Then, Shell is used on the bottom side:

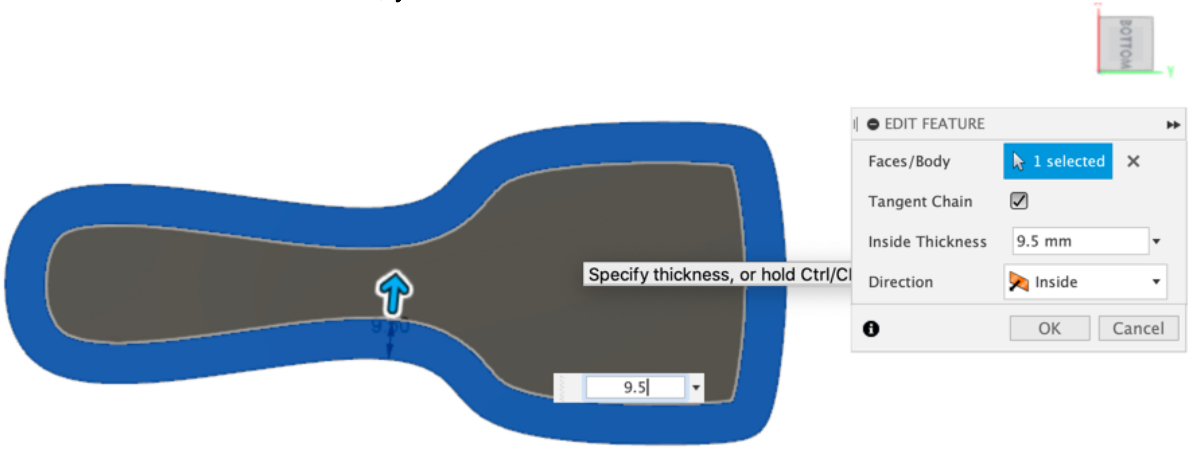


With a chamfer of 5 mm, this body shells OK at 3 mm thickness, but fails at 4 mm. However, with a chamfer of 6 mm, Shell will start to fail at 0.8 mm thickness. Why is that? The answer, again, is curvature, but this case is interesting because of the interaction of the curvature of the profile spline, the Chamfer, and the Shell. Let's dig in a bit deeper.

The curvature of the spline curve is the source of the issue here. If you turn on curvature comb analysis for the sketch, it looks like this:



Next, let's look at the Chamfer. Without the Chamfer, this body shells quite stably, at any reasonable thickness. Here, you can see that we can do a 9.5 mm thick shell:



EDIT FEATURE

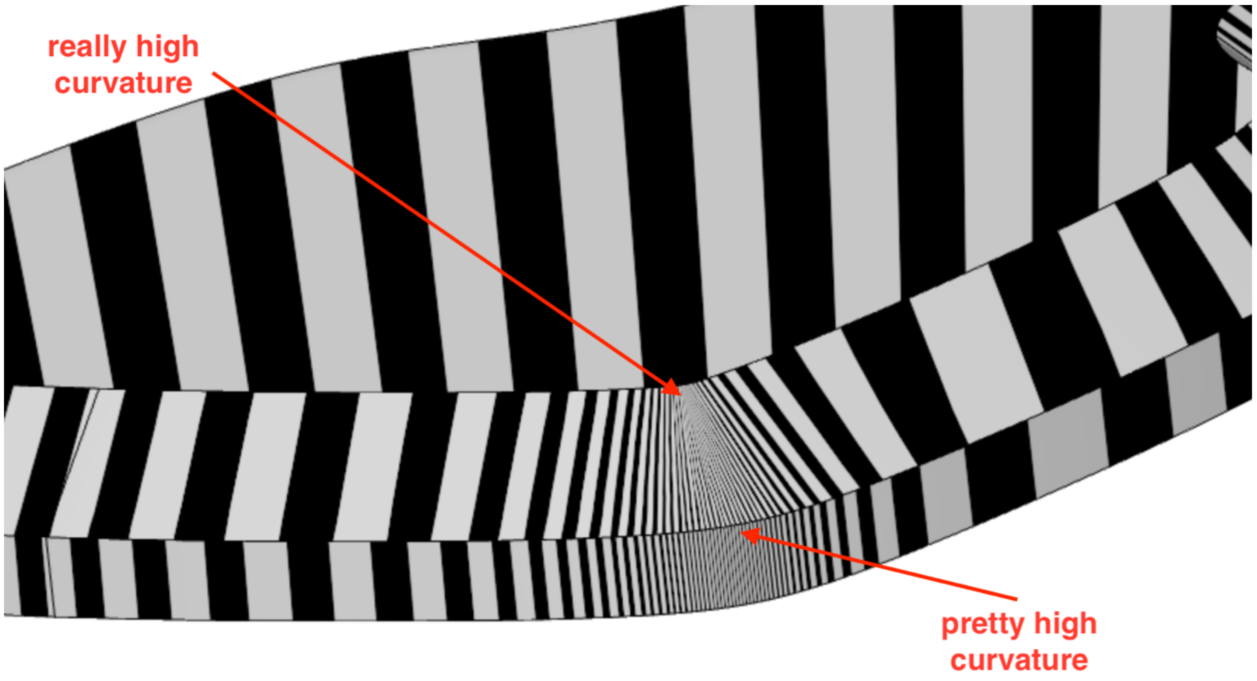
Faces/Body: 1 selected

Tangent Chain: ✓

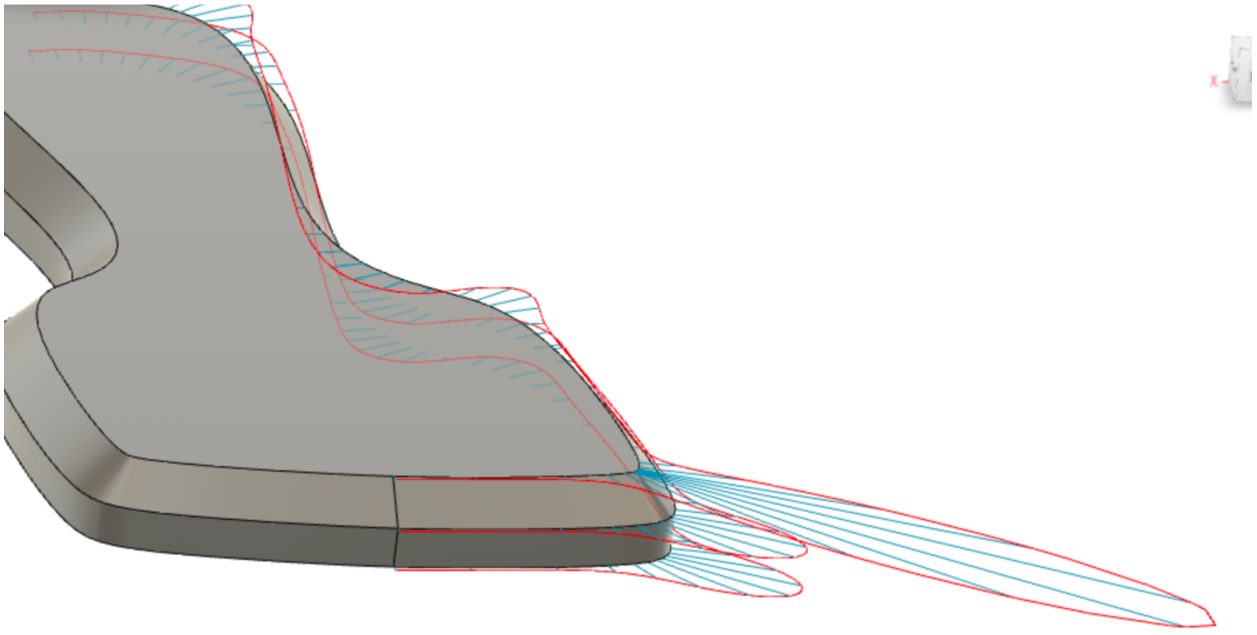
Inside Thickness: 9.5 mm

Direction: Inside

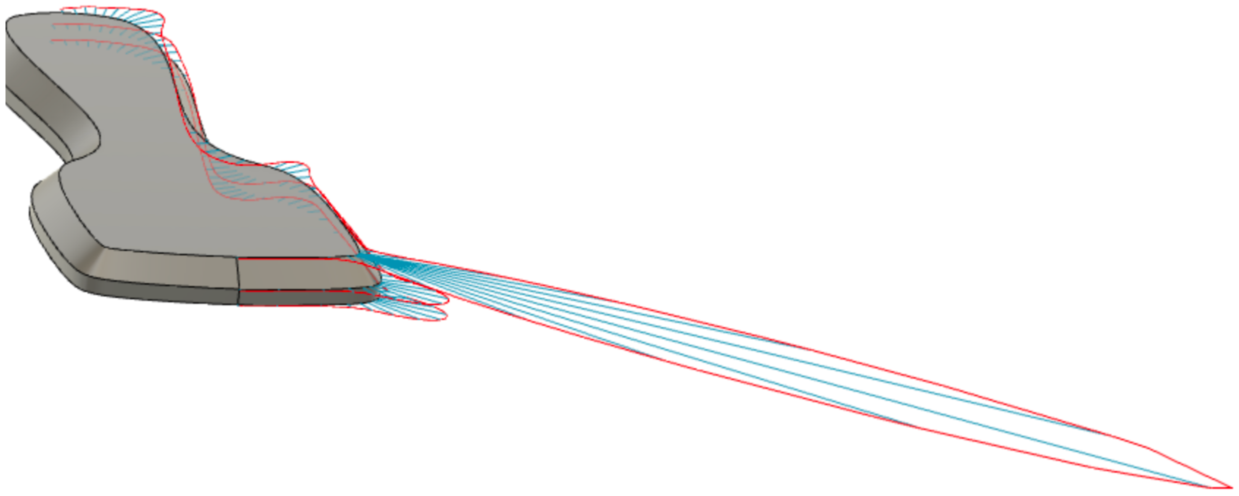
Eventually, this will fail in that same corner, but you can get a lot farther. However, when you add the chamfer around the outer edge of the body, it has the effect of concentrating the curvature of the surfaces that need to be offset, at the top of this body. You can see this if you turn on a Zebra Stripe analysis:



With a Chamfer of 5 mm, the curvature comb of edges looks like this:



But, if you bump that up to 6 mm, here is the curvature comb:



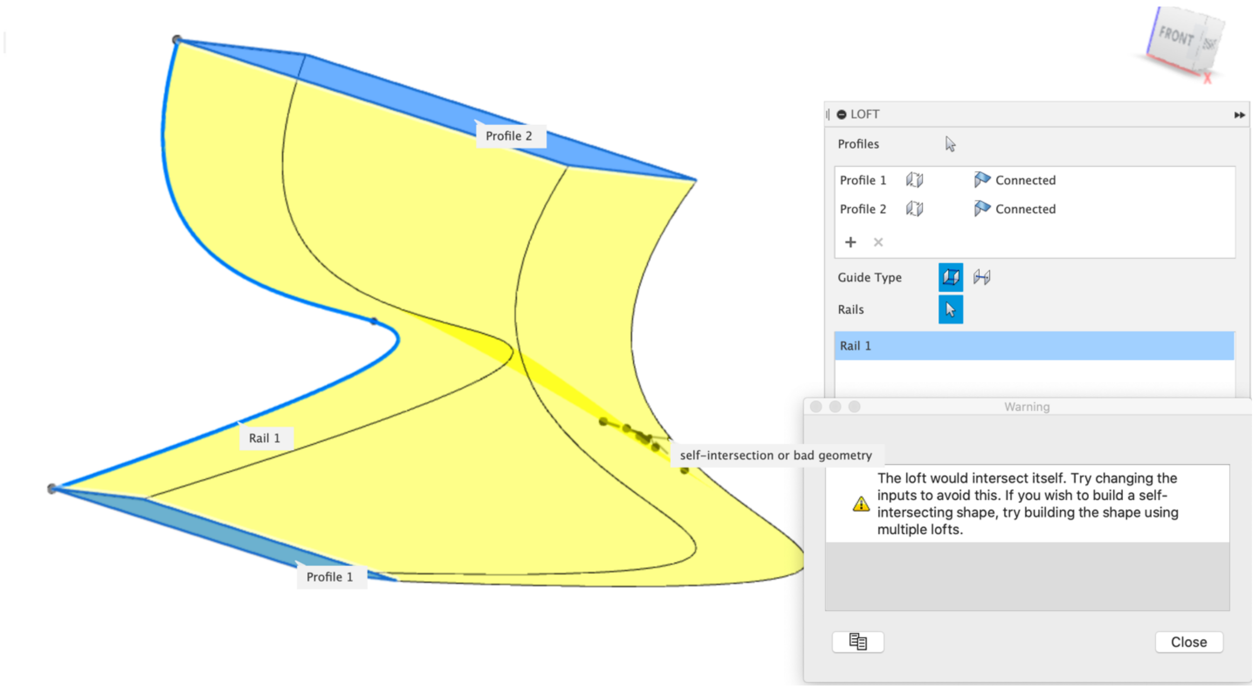
This is why the range of success for a 6 mm Chamfer is so narrow.

4.3.1.3 Loft Geometry Errors

Loft is another common source of geometry failures. There are really 3 most common sources of Loft geometry errors:

- Self-intersections (detecting a pattern here?)
- Rail issues
- Point Mapping issues

Loft Self Intersection Geometry Errors. Have you detected a pattern here yet? The Fusion modeling kernel does <u>not</u> like surfaces that intersect with themselves. This situation can be pretty easy to encounter in Loft, if you use complex profiles or rails. Here is a simple, obvious example:

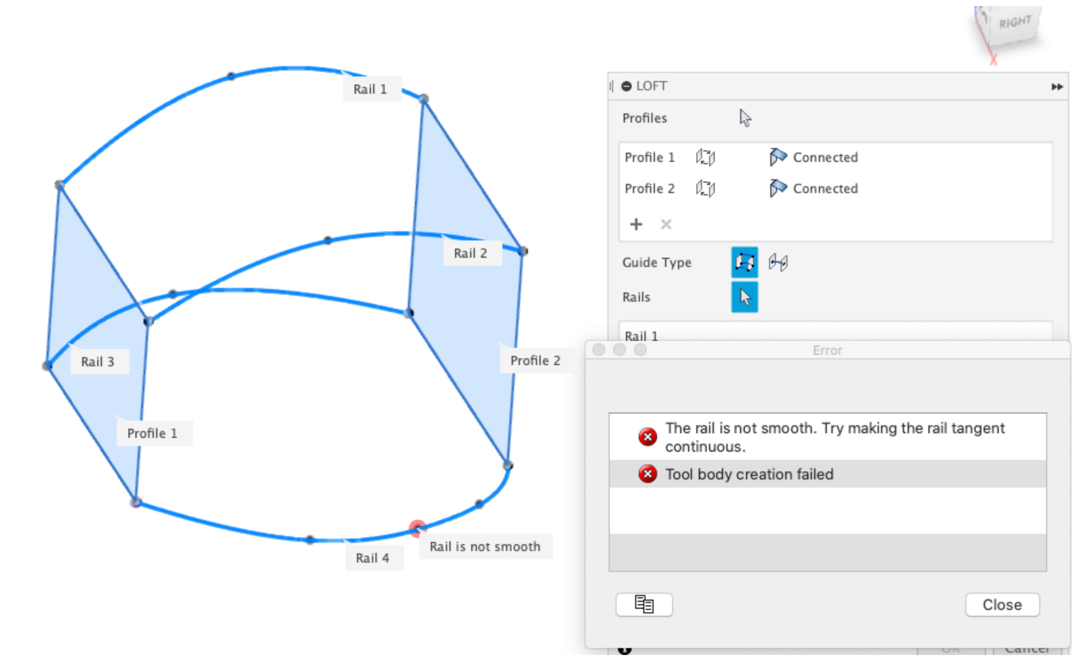


The loft would intersect itself. Try changing the inputs to avoid this. If you wish to build a self-intersecting shape, try building the shape using multiple lofts.

Basically, the rule is: The surface cannot intersect itself. This is true for both surface and solid lofts. (We mention this, because there are some cases for which the behavior and limitations are different…). The fix, as expected, is to modify the profiles and/or rails to prevent the self-intersection.

Loft Rail Errors. This is a fairly common set of failures for more advanced lofts, especially for "advanced-beginner" users who are just starting to explore lofts with rails. I wrote this post a few years ago as kind of a starter's guide to lofting with rails: introduction to loft using rail curves in Fusion. There are 3 main kinds of rail failures:

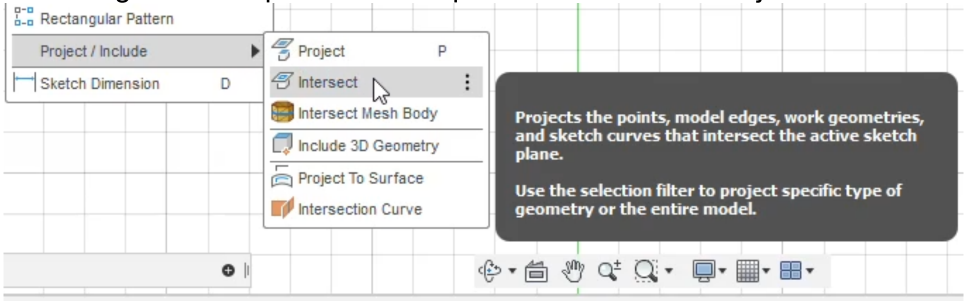
- Tangency failures in the rail itself. If the rail is composed of more than one curve, the entire chain must be tangent continuous. The error message for this is pretty self-explanatory, thankfully



The rail is not smooth. Try making the rail tangent continuous.

Tool body creation failed

- Profile/Rail intersection mismatches. This one is fairly common, and is the main topic of the forum post above. The modeling kernel requires that the rail curve must intersect every profile in the Loft, precisely. It is easy to create data that violates this condition if you are not careful. The main user frustration with these kinds of geometry errors is that the feedback is (currently) not very helpful. If you have 4 profiles and 4 rails, that means there are 16 possible intersections, any of which could be wrong. We are working on providing better feedback for this error, to help you know which intersection needs to be fixed. The best advice here is to avoid this condition in the first place. The details are in that forum post, but the single most important tool to prevent this error is Project -> Intersect.

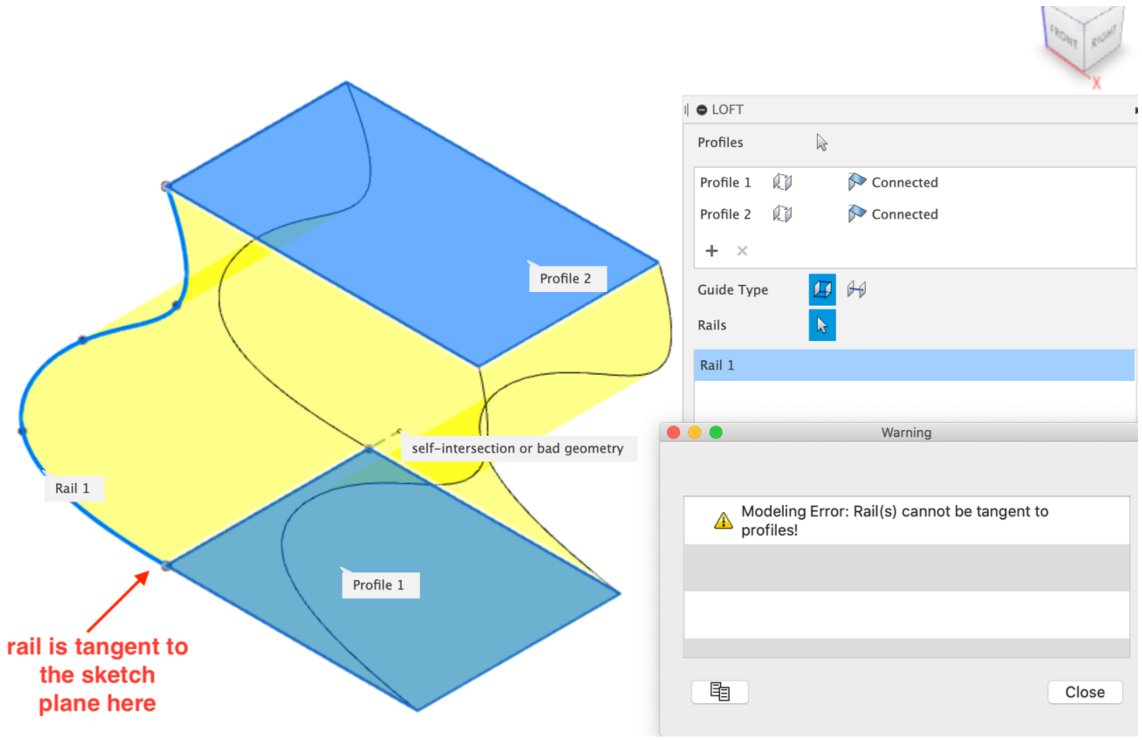


Projects the points, model edges, work geometries, and sketch curves that intersect the active sketch plane.

Use the selection filter to project specific type of geometry or the entire model.

Whether you choose to create the profiles first, and then the rails, or the other way around, this tool will guarantee that a precise intersection point exists in your sketch.

- Profile/Rail tangency failures. Another rule is: A rail (normal rail, not a Centerline rail) cannot be tangent to the plane of a profile at the point of intersection.

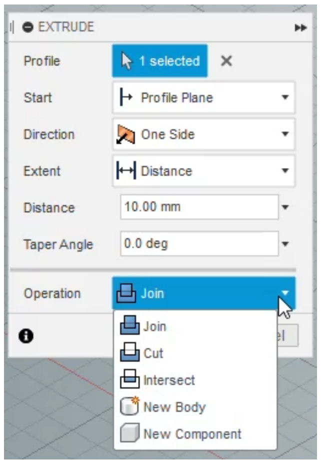


Modeling Error: Rail(s) cannot be tangent to profiles!

the fix is to make sure your rails intersect the profiles at some positive angle.
the fix is to make sure your rails intersect the profiles at some positive angle.

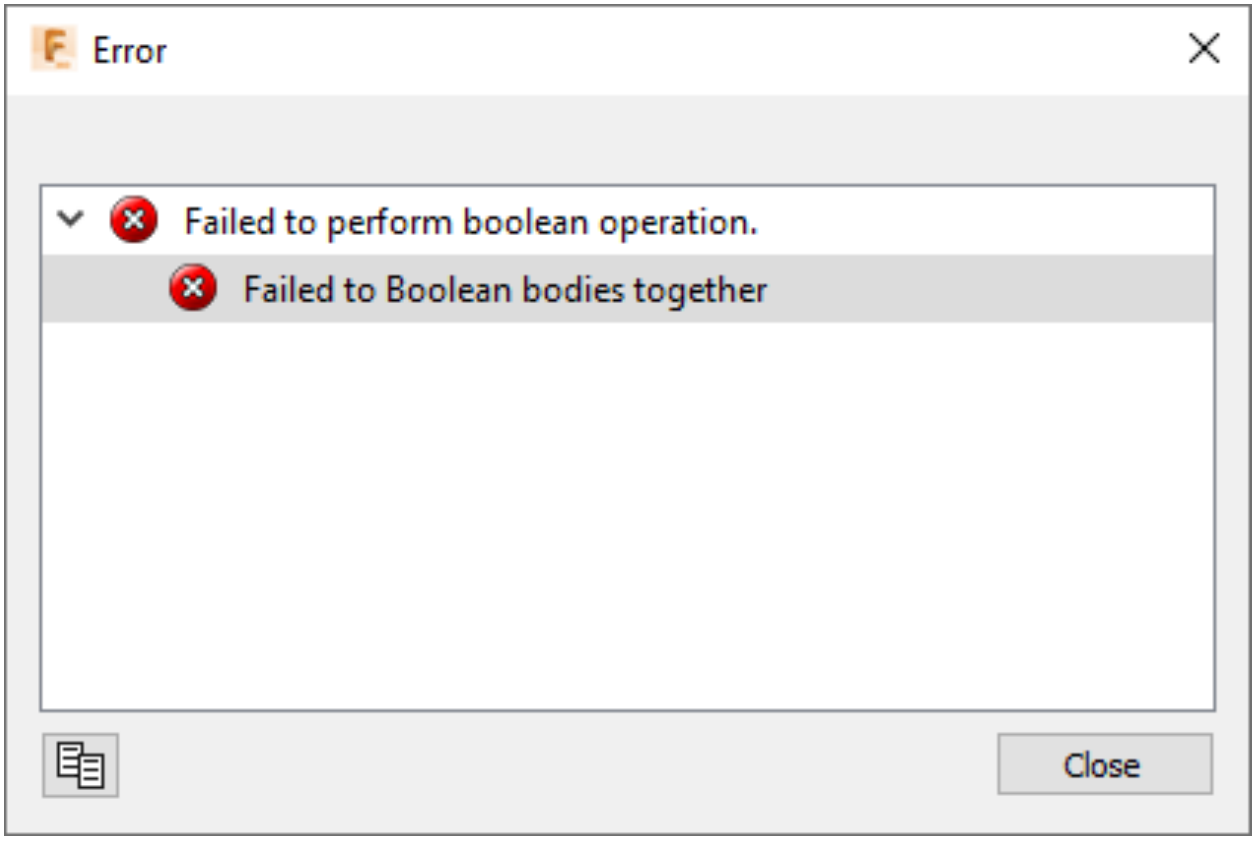
4.3.1.4 Combine/Boolean/Join/Cut/Intersect Errors

This class of operations all boil down to the same thing: A failure to join, subtract, or intersect two solid bodies. In feature creation command, such as Extrude, Revolve, Sweep, etc., the basic flow is: The commands produce a tool body from the selected geometry, then perform a Combine (internally) with the intersected existing solid body. So, all these features plus Combine end up executing the exact same kernel code for that operation.



The most common cause of Boolean failures (which can result in a number of different error messages, depending on where the error was caught in the kernel code) is because of "near coincident geometry". That is, geometry that is very close, but not precisely the same, in both the target and tool bodies. These near-equivalent surfaces can cause errors in the code trying to do the Boolean operation. If you think about this, it should be obvious – if you are trying to subtract a body from another body, and their geometry is, say, 1.0E-05 different from each other, it will be very hard to do that subtraction.

In general, the fix is to avoid these situations as much as you can. Make sure there is plenty of overlap, especially if the geometry is complex (e.g. a spline surface). If you are trying to cut out a hole, over-build the cutout. Note that "to object" terminations in Extrude should be fine. The surfaces there will be exactly the same, not "nearly coincident", so please feel free to use that termination type.


Error

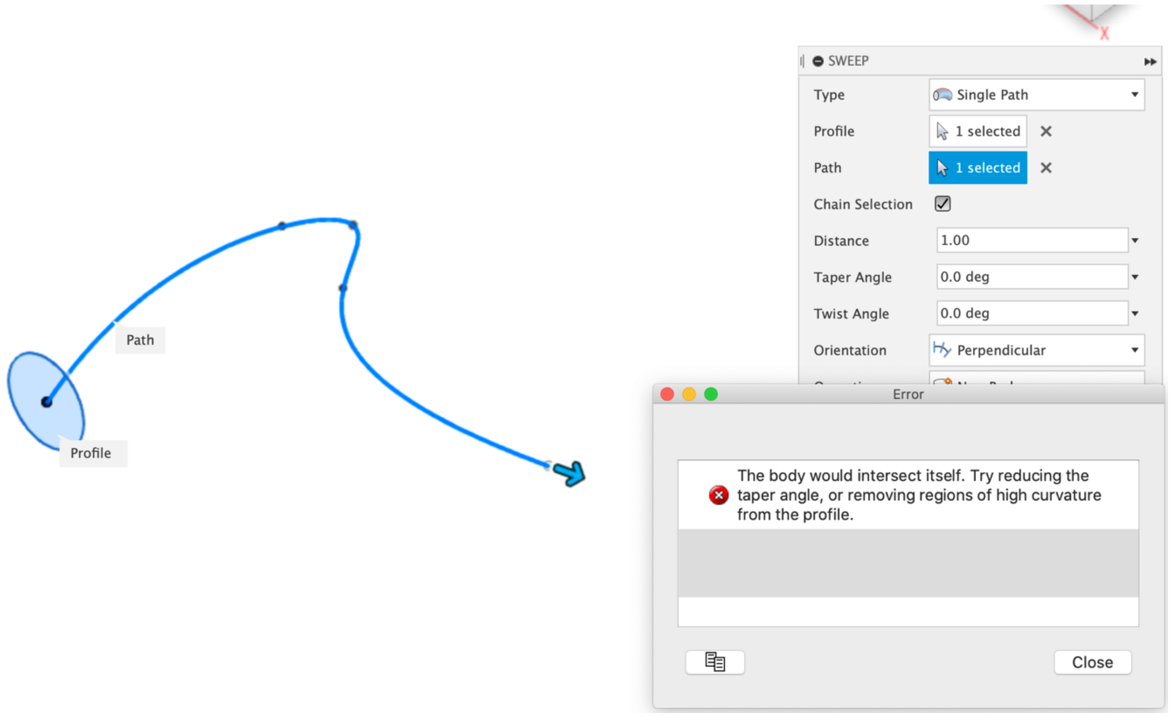
Failed to perform boolean operation.

Failed to Boolean bodies together

4.3.1.5 Sweep Geometry Errors

Sweep is next. Again, the main geometry error in Sweep is… Self-intersecting surfaces. This one can be even more annoying, because, Fusion allows self-intersecting *surface* geometry, but not so in solid.

Solid example:



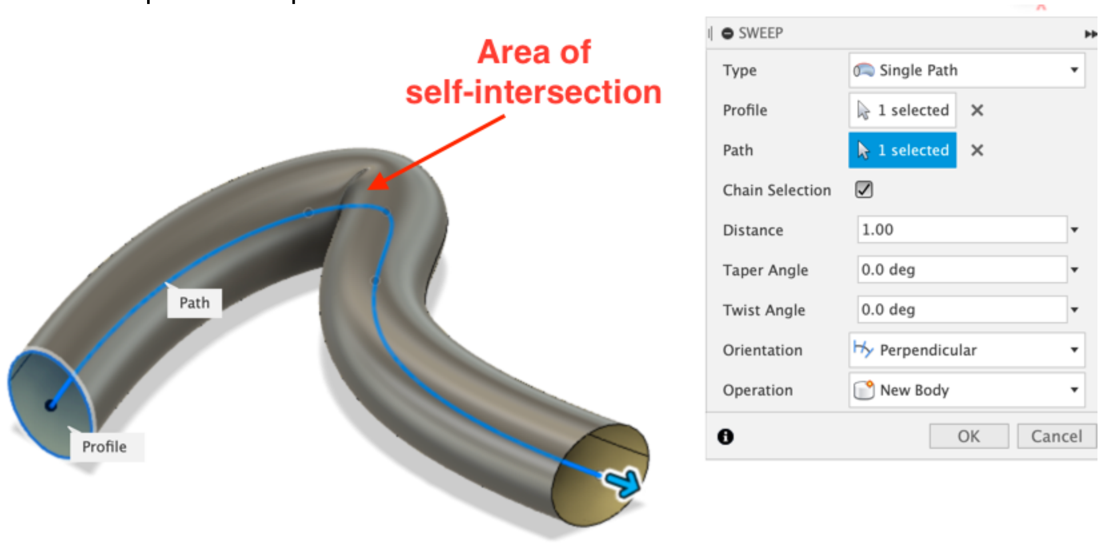
SWEEP

Type	Single Path
Profile	1 selected
Path	1 selected
Chain Selection	✓
Distance	1.00
Taper Angle	0.0 deg
Twist Angle	0.0 deg
Orientation	Perpendicular

Error

The body would intersect itself. Try reducing the taper angle, or removing regions of high curvature from the profile.

Same exact profile and path in surface:



SWEEP

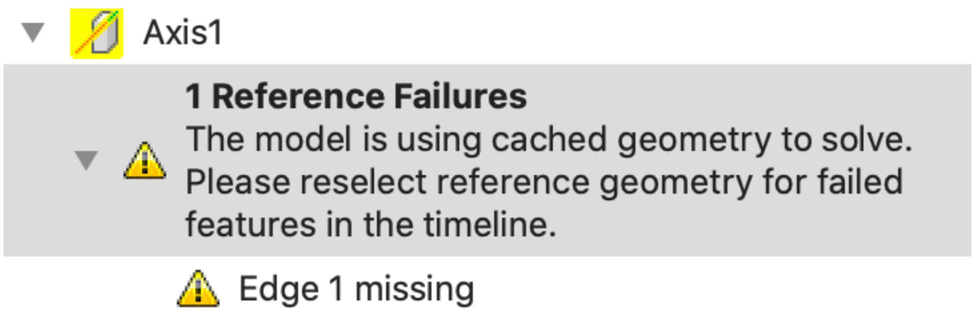
Type	Single Path
Profile	1 selected
Path	1 selected
Chain Selection	✓
Distance	1.00
Taper Angle	0.0 deg
Twist Angle	0.0 deg
Orientation	Perpendicular
Operation	New Body

Two possible fixes here:

1. Reduce curvature in the path, or size of the profile
2. Cheat – create it as a surface, patch the ends, and Combine it with your target body

4.3.2 Dependency Modeling Errors

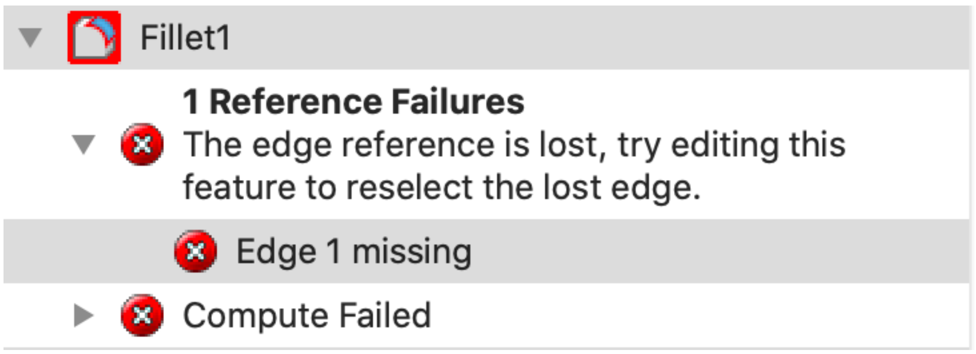
Unlike Geometry modeling errors, Dependency errors are pretty universal. They will usually show up as some kind of a “not found” error. Here are some examples. Each feature adds its own wording, so the Errors may be slightly different. This type of Error/Warning should be pretty obvious, however.



Axis1

1 Reference Failures
The model is using cached geometry to solve. Please reselect reference geometry for failed features in the timeline.

Edge 1 missing

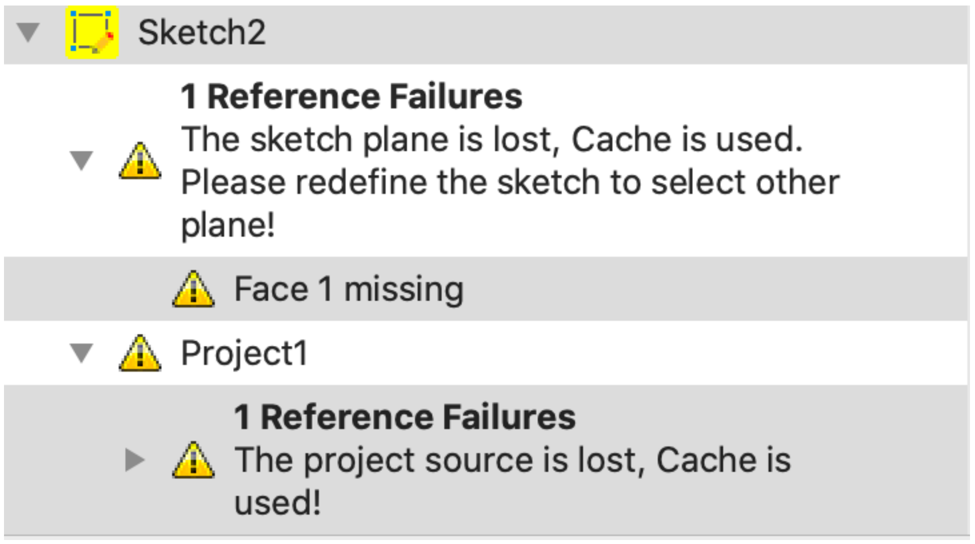


Fillet1

1 Reference Failures
The edge reference is lost, try editing this feature to reselect the lost edge.

Edge 1 missing

Compute Failed



Sketch2

1 Reference Failures
The sketch plane is lost, Cache is used. Please redefine the sketch to select other plane!

Face 1 missing

Project1

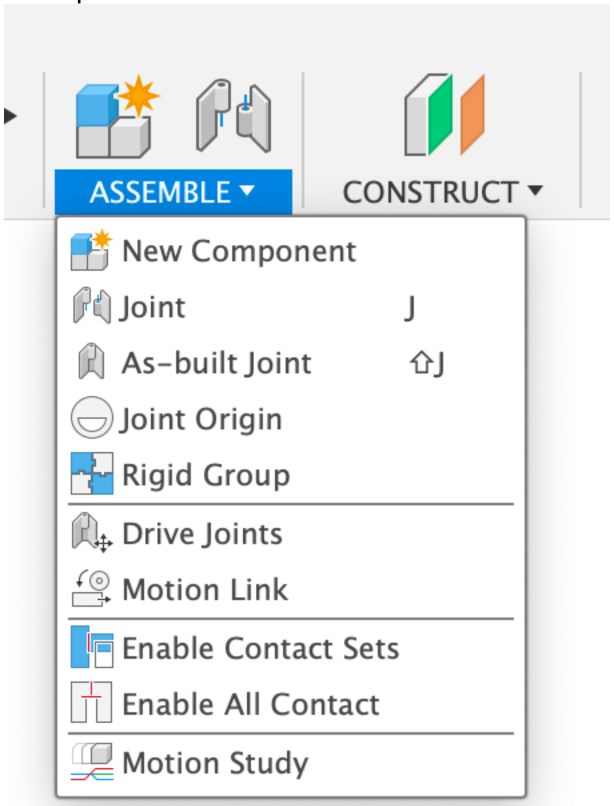
1 Reference Failures
The project source is lost, Cache is used!

There are only two real methods for repair of this type of error:

- Fix (or undo) the design edit that caused the error. This is not always possible, but if you can use this method, it will be preferable, and possibly result in downstream, cascading Errors to be fixed as well.
- Use Edit Feature to re-select the failed reference geometry/topology
    - One special case of this is a sketch plane reference failure. As described above, the Redefine Sketch Plane command can be used to choose a replacement plane.

4.4 Debugging Assembly Errors

In this section we address some common failures in the area of Assembly Modeling. Though Fusion does not support a distinct Assembly design type, there are certain workflows that can be classified as "assembly" tasks. Most of these involve commands that exist in the Assemble dropdown in the Design workspace:



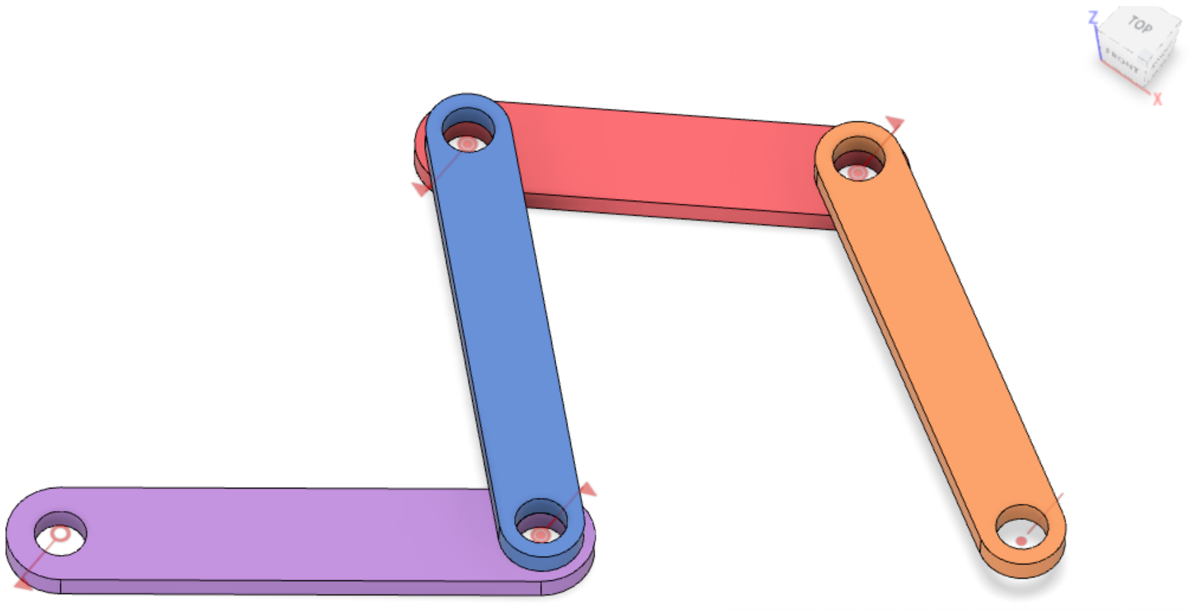
4.4.1 Joint Failures

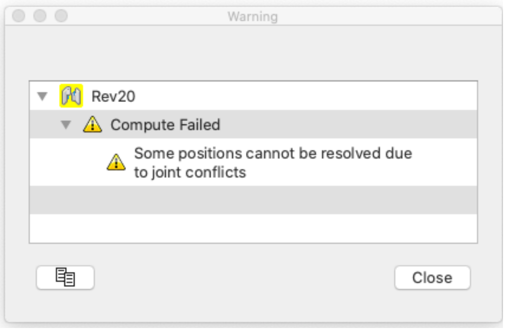
To be completely honest, this is one of the most challenging areas of Fusion to try to debug. There are unfortunately not a lot of good tools here to help you. But, there are some techniques that can help.

There are a range of failures that can happen when designing a mechanism, or even positioning rigid components within an assembly. Some of these are actual Errors (i.e. produce an error dialog, show red in the timeline), but a lot of them are just bad mechanism behavior, or incorrect positioning of components, or the command to create the Joint fails.

Most actual Errors in the Joint system (either failure to create or actual Error states in Joint features) are predominantly caused by inaccuracies in the geometry being joined. Unfortunately, the Joint solver in Fusion does not have any allowances for “slop” in components (as you would get in the real world). So, you must make sure that all the distances and angles between connection points in your design are exactly accurate. A 2 degree difference in angle, or a .01 mm discrepancy in length will cause the joint to fail.

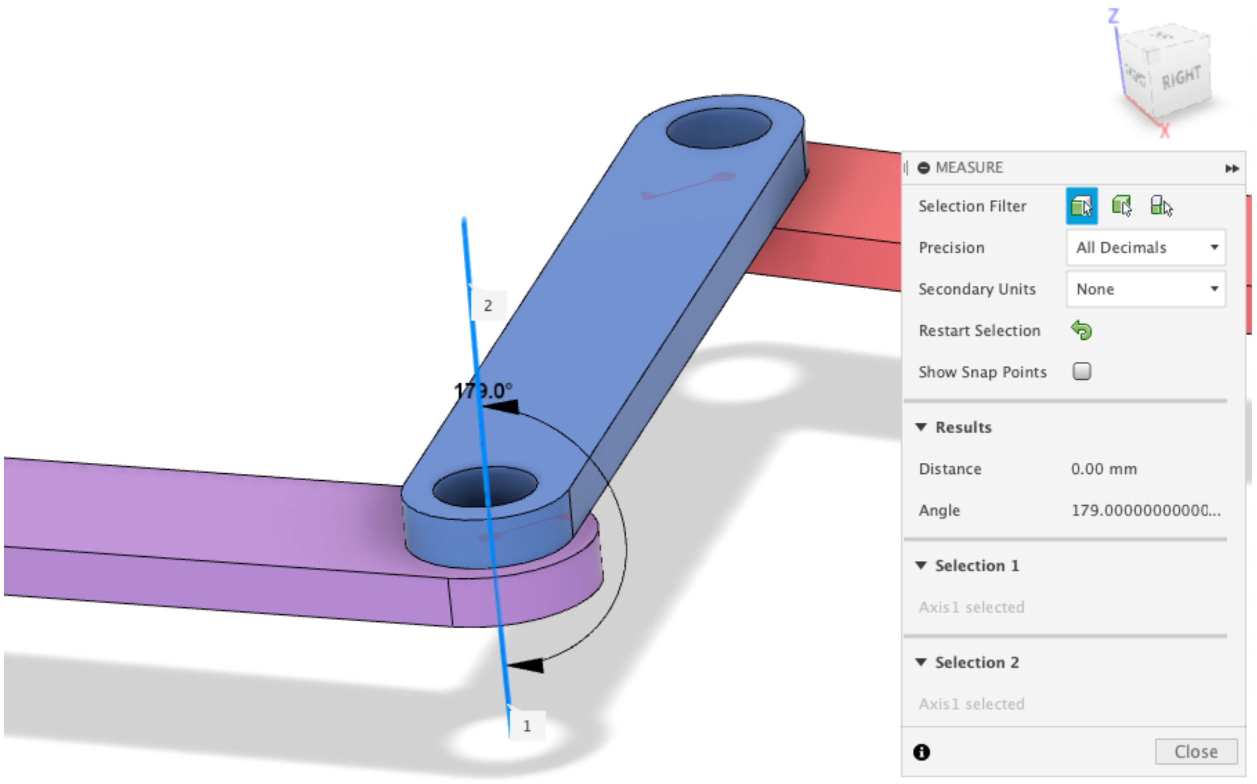
A technique that can help here is the degrees of freedom in some joints. For instance, if you have a Joint system which fails, you can sometimes replace one Joint type with another that has extra motion, and this will help to compensate for the lack of accuracy in the geometry. For example, If you have a series of Revolute Joints which fail, try replacing one or more of the Revolutes with Cylindrical Joints. The Cylindrical Joint allows the components to move linearly along the axis of revolution of the Joint, and this can sometimes free up enough movement to fix the assembly.





Rev20
Compute Failed
Some positions cannot be resolved due to joint conflicts

In this case, the underlying problem is a small error in an angle in the geometry in the failed joint:



MEASURE

Selection Filter	
Precision	All Decimals
Secondary Units	None
Restart Selection	
Show Snap Points	

Results

Distance	0.00 mm
Angle	179.00000000000...

Selection 1

Axis1 selected

Selection 2

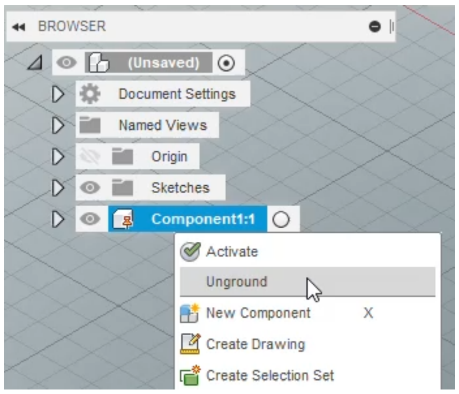
Axis1 selected

Close

Correcting this angle will fix the joint system.

4.4.2 Understanding Ground in Fusion

Ground in Fusion is somewhat unique in CAD applications. Understanding Ground can be critical to getting your system to work as you expect, if you use bottom/up (distributed designs, external referenced designs).



There are three principles to understand for Ground.

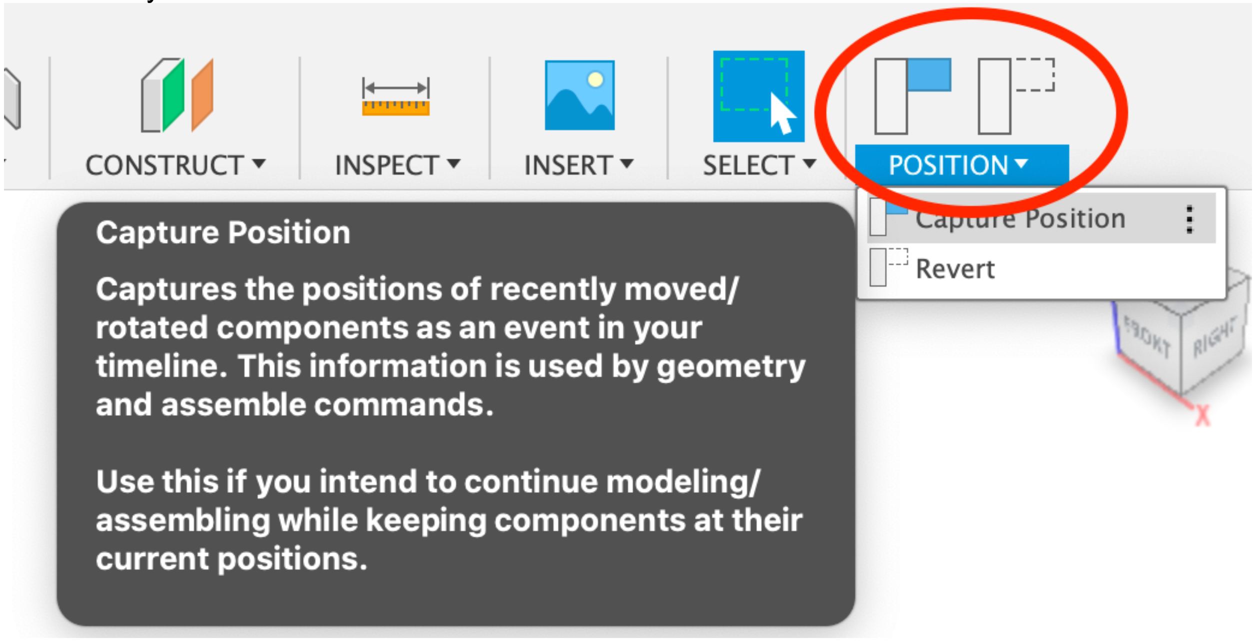
1. Ground is a Timeline feature. It appears in the Timeline, but more importantly, Unground is also a Timeline feature. Ground is not an "attribute" of a component but is a state which is applied at a given time in the timeline. Unground can also be applied at a certain point in time. There are some powerful tricks you can use knowing this, but for 99% of users, you will never use those tricks. Just be aware of this nature of Ground, and avoid overuse or redundant instances of Ground/Unground features in your timeline
2. Ground in Fusion is defined to only be effective in the design in which the Ground feature is placed. The Ground state of a component is not transferred if that design is Inserted into another design. This can be confusing at first, but makes total sense if you think about it. Ground will freeze the component to which it is applied. But, if you insert that design into another design, and Ground were transferred, that would mean that this component would be fixed in all usages of this design, and this is almost certainly not what you want. For example, if you have a model of a car suspension in a design, you will want to Ground some part of the suspension in order to test the function of the suspension. However, if you insert that sub-assembly into the car assembly, you want that suspension to be free to move (especially for the front suspension, which will need to turn with the steering mechanism)
3. Grounding the root of a sub-assembly does not ground the children. Most of the time, it will appear as if it does nothing, which can be confusing. Grounding a sub-assembly (local or external) just grounds that component. It does not ground any children of that component, which are still free to move. This will be explained in the next section on component positioning. But, you can see this effect by turning on the coordinate system of the sub-assembly. You will see that it is, indeed, grounded, but that the child components of the assembly are still free to move. There are tricks you can use to work with grounded sub-assemblies, if you want to do that. That is discussed in the "flexible sub-assembly" section below. Most traditional CAD applications treat ground in a sub-assembly as defining that sub-assembly to be rigid.

4.4.3 Understanding Component Positioning Model in Fusion

Another important concept to understand in the area of Joints and assemblies in Fusion is how components are positioned in Fusion. There are several important points to understand:

1. Moving components and Capture Position
2. “flexible sub-assemblies”

Moving Components. In Fusion, there are several ways to move a component. These include Component drag, Drive Joint, Animate Joint, Animate Assembly, and Align. Unlike Body Move, moving a component does NOT create a Timeline feature. Why? Mainly this is because users who are designing a mechanism will want to exercise this mechanism to see how it behaves. If every one of these movements created a Timeline entry, your Timeline would possibly contain hundreds of Component Move features. Besides polluting your Timeline, these features would be a performance concern. Instead, a Component move puts the Fusion session into a “Capture Pending” state, indicated by this UI in Fusion:

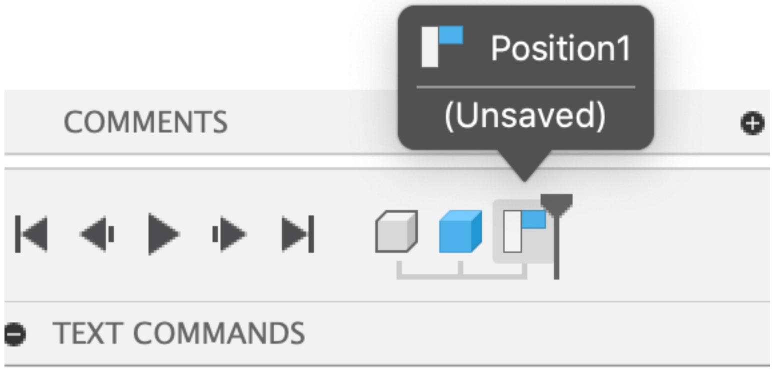


Capture Position

Captures the positions of recently moved/ rotated components as an event in your timeline. This information is used by geometry and assemble commands.

Use this if you intend to continue modeling/ assembling while keeping components at their current positions.

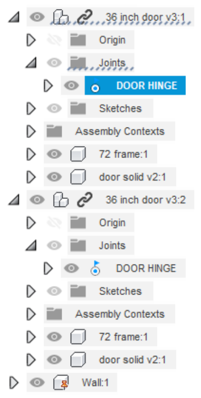
When these icons appear in the Toolbar, your design is in a Capture Pending state. It indicates that some components in the design have been moved. If you click on “Capture Position”, a feature will be added to your Timeline which, well, captures the position of all components that are in this moved state:

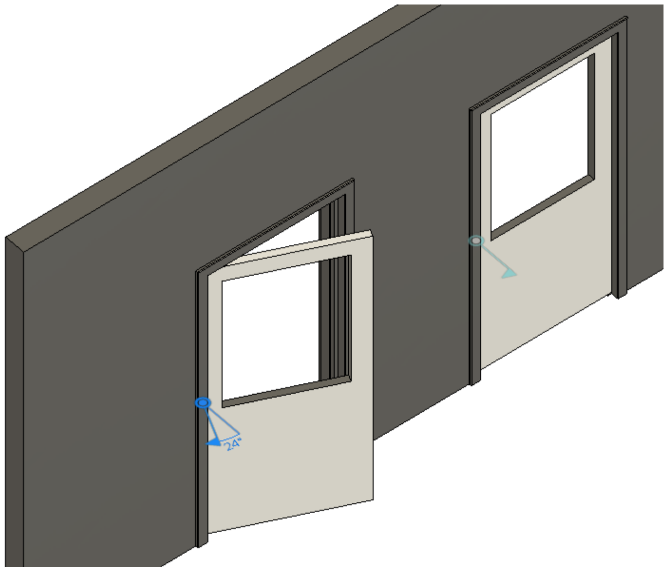


This feature stores the position of each moved component, and any Joint values that have been used to position those components, if applicable. When this feature is evaluated in the course of a Timeline compute, the components' positions which are captured in this feature are set to the captured valued, and an assembly Solve is done. (side note: This is also why having lots of Capture Position features in your Timeline can affect performance. If your assembly is large, each assembly solve can be expensive. If there are lots of them, compute time will be significantly impacted).

Flexible sub-assemblies. This is another very important concept to understand in order to debug your assembly. If you have more than one level of component hierarchy in your design, then you will have "sub-assemblies". In Fusion terms, that is a Component which is a child of the root Component, which also owns child Components of its own. Whether this is an external design or totally contained in a single design doesn't matter here. Next, if you have more than one instance of a sub-assembly in your design, then you can see the difference that flexible sub-assemblies can cause. Fusion's sub-assemblies are considered "flexible" because child components owned by different instance of the same sub-assembly can be in different positions. The classic example that the development team uses to discuss this is the "door + frame" example. Imagine you have a sub-assembly which is composed of a door frame component and a door component. (probably plus hinges, but we'll ignore those for now). There is probably a Revolute Joint in between the door and its frame, that is owned by the sub-assembly. That Joint has a position, or a setting of the revolve angle. If there are multiple instances of the door + frame assembly, flexible sub-assemblies mean that each instance can have this Revolute Joint with a different angle. Or, to state it another way, you can have one door in your design be in an open state, and another in a closed state.

How is this managed? The short version is: when an assembly is instanced into a top-level assembly, under the covers, each Joint owned by that assembly is "promoted" to be driven by the top-level design. So, there are “instances” of each Joint, as well as instances of the design itself, and each of those can take on different settings.





OK, so why is this important? The main thing this does for Fusion designs is that it allows child components of a sub-assembly to have different positions in different instances of that sub-assembly. This means that those positions are <u>of the child component, not of the sub-assembly itself.</u> This has a number of implications. One of which we’ve seen above: Grounding the sub-assembly itself can often appear to “do nothing”. If you turn on the Origin of the sub-assembly you can quickly see why this is the case. The sub-assembly is, correctly, grounded. That origin will not move. However, because of flexible sub-assemblies, the child components of this sub-assembly are still very free to move.

Similarly, creating a Joint to a sub-assembly will also appear to “do nothing”. The explanation is the same as for Ground. Though you may have defined a Joint to that sub-assembly component, that Joint <u>does not</u> constrain the child components at all.

What if I want my sub-assembly to be rigid? Today, there is no built in command to make a sub-assembly rigid, but there are a couple of ways to do that, and also to mimic traditional CAD application behavior around Ground and Inserted designs.

Making a sub-assembly rigid is pretty simple. The most straightforward way to do this is to fully constrain all the child components of that sub-assembly using Rigid Joints. Or, you can do this if your design is built in-place, using As-Built Rigid Joints (or Rigid Group, which is the exact same thing – make sure you understand why this is true…)

Another way to achieve sub-assembly rigidity is by creating joints between child components and the origin of the parent sub-assembly. This does essentially the same thing as a rigid group but has one main difference: This method also locks down those child components with respect to the sub-assembly's origin. Again, turning on the origin in an example design will help to illustrate this.

This also has one additional added effect that relates back to our Ground discussion. Think about this for a minute: If you create a Rigid Joint between a component and the origin of the root component (i.e. the sub-assembly, once this is Inserted into another design), then, within this document (while it is the active edit target), that component is effectively Grounded. The behavior is identical. However, the big difference her is: When you Insert this design into another design, this Rigid Joint (unlike a Ground state) does come along. This is actually the behavior that a lot of existing CAD systems show with Ground + Insert. In the context of the sub-assembly design, the component is grounded, but when inserted into another design, it is effectively rigid. This is pretty powerful!

5 Preventing Errors in Your Fusion Design

The third major topic we want to cover today is one I'll call "preventative medicine". Up until now, we've focused on understanding the errors in your design and some techniques to fix them. That's really the "ER" approach. But, if we can keep you out of the ER with some preventative measures, we think your experience with Fusion will be much better.

These techniques can be used to help you keep your design clean and free from errors under most edit scenarios.

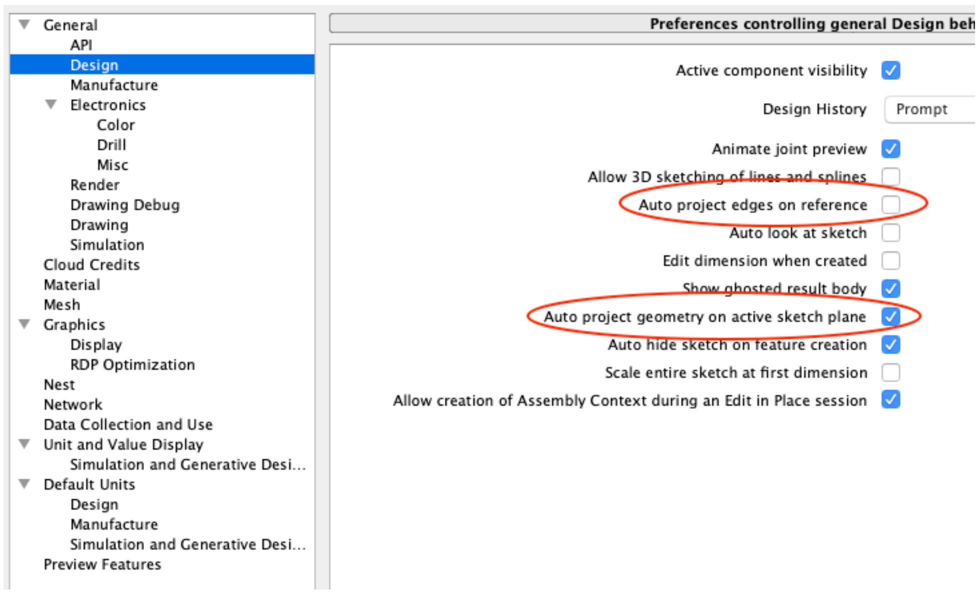
5.1 Dependency Management

As we discussed earlier in this class, the largest source of Timeline errors are failed references. If you are able to put some thought into geometry references as you create your design, you will produce a design that is as stable as possible, and thereby prevent a lot of the failures you may have seen in the past.

5.1.1 Minimize Unnecessary Dependencies

This recommendation is certainly the most important. If each reference is a possible future error, then it should be obvious that the fewer references there are, the less likely you will be to cause an error with a design change. So, with that in mind, here are a few recommendations to reduce the sheer number of them:

- Turn off the sketch preferences which "automatically" create references. Sadly, this is Fusion's default settings actually working against you. These options were put into place in an attempt to make Fusion easier to use. I suppose these options do, in fact, make Fusion somewhat easier to use in the short term, but they do so at great risk. These two are the problem options:


General
API
Design
Manufacture
Electronics
Color
Drill
Misc
Render
Drawing Debug
Drawing
Simulation
Cloud Credits
Material
Mesh
Graphics
Display
RDP Optimization
Nest
Network
Data Collection and Use
Unit and Value Display
Simulation and Generative Desi...
Default Units
Design
Manufacture
Simulation and Generative Desi...
Preview Features

Preferences controlling general Design beh

Active component visibility ✓
Design History Prompt
Animate joint preview ✓
Allow 3D sketching of lines and splines ☐
Auto project edges on reference ☐
Auto look at sketch ☐
Edit dimension when created ☐
Show ghosted result body ✓
Auto project geometry on active sketch plane ✓
Auto hide sketch on feature creation ✓
Scale entire sketch at first dimension ☐
Allow creation of Assembly Context during an Edit in Place session ✓

The first one (auto project edges on reference) creates references (by Projecting edges) during curve creation. This one is particularly bad, because it is not obvious that this is happening.

The second one ("auto project geometry on active sketch plane") automatically Projects each edge of a planar face selected for a new sketch. If that face is complex, this can create hundreds of geometry references without you even knowing it.

Instead, turn these off, and manually project only the edges/faces/vertices that you actually need to have in your sketch. Yes, it's one more step, but I think you're design will be better off if you do.

- Sketch dimensions and constraints are even worse, because there is no preference that actually disables auto-projection – it happens unconditionally. So, be careful when creating/placing sketch dimensions and constraints – don't accidentally select an edge to dimension to if it is not absolutely necessary to dimension to that edge. Sadly, you may need to turn off body visibility, or make the body un-selectable temporarily to prevent this. We really need to fix this…
- Be mindful when creating any feature of only picking geometry when that relationship is actually required to achieve your design intent. Often, that reference is not necessary, and your design intent can be more stably achieved using methods like referencing a user parameter.

5.1.2 Choose the Most Stable Dependency

When referencing geometry in your model, reference stability is everything. Well, that is, assuming you ever want to edit your model, and have it update correctly. In Fusion, all model references are **not** created equal in terms of stability.

- **Origin work geometry is the most stable of all.** These guys never change. If you can get away with using an origin plane from your component as a reference instead of a body face or edge, I can guarantee that this reference will never go bad.

- **Work geometry that only references origin geometry is also good**. If you can create an offset plane from the YZ plane, and use that as a reference, you also will have a very stable relationship. You can think of this as defining datum planes.
- **Not all work geometry references are inherently stable.** This is worth noting. The above two hints might seem to imply that somehow work geometry is more stable than BRep geometry. This is not true if that work geometry also contains a BRep reference. For instance, if you offset a workplane from a face, you might think that it is fine. However, all you are doing is adding one degree of separation from the BRep reference. If the BRep reference inside of the offset workplane fails, then your workplane is no good, either.

5.1.2.1 BRep dependencies

- **Body references are more stable than other BRep references.** Select an entire Body as a reference, rather than an individual face or edge, if you can.
- **Face references are the next level of stability.** With the caveats discussed above (faces depend on sketch geometry), faces are the next most stable BRep reference.
- **Edge references are less stable that face references.** The underlying reason for this is that in order to resolve an edge reference, Fusion needs to resolve two face references (an edge is the intersection of two faces). So, it is twice as likely to fail as a single face reference.
- **Vertex references are the least stable of all.** Even though this is tempting, because it leads to a visually simpler sketch to just project a vertex, you should avoid these if you can. For a vertex reference to succeed, Fusion needs to match 3 different faces.

5.1.2.2 Sketch dependencies

- **Sketch references can also be very stable**, as long as you are careful. Often times, you can refer to a sketch curve in your model. This is a very common practice for Joint references. Unless you go into the source sketch and manually delete a line, that line will be there forever. That is, if there is a failure, it will absolutely be your fault.

Recommendation: Use the most stable object you can when making a reference.

5.1.3 Be Careful When You Edit Sketches

As we saw in section 3.3 (CAD Naming and Matching), the entire entity matching apparatus in Fusion is driven by IDs of sketch curves. So, as you are editing a sketch which is consumed by modeling features, you need to keep this algorithm at least in the back of your mind. The main advice here is: Don't delete sketch curves in a consumed sketch, if you can at all avoid it. Instead, just edit the curves by:

- Dragging
- Adding or editing dimensions
- Adding or removing geometry constraints

Second, be aware of some sketch commands which can end up creating new sketch curves (which therefore get new sketch IDs, and which can cause downstream features to fail):

- Trim
- Break
- Fillet
- Offset – sometimes, on update of Offset (if the offset distance changes, or the input curves change) the curve ID's of the offset curves can change. This one is hard to work around, to be honest. The best advice is to try to avoid making too many direct references to offset sketch geometry.

5.1.4 Using Parameters Instead of Dependencies

One advantage of a parametric design CAD application is: It contains parameters… And, in many cases, you can use parameters to avoid having to reference geometry. For example, consider a simple bookcase design. This is an assignment that Phil uses in his Fusion classes.

When you build the components for this bookshelf (2 sides, a back, a top and a bottom, and however many shelves you want). There are many ways to do this. Let's look at 2 of them:

The first starts by creating a component using an extruded rectangle for the back panel. The sides, top and bottom are all made by sketching on the side faces of this component, and extruding using "To Object" to make sure the depth is the same. The shelves are created using sketches on the side faces, and, again, "To Object" to make the geometry line up. This design is quite functional, parametrically. Just change the length and width of the back panel and everything updates. However, there are many dependencies here. Each one of them is a potential design failure.

The second approach starts by defining a handful of User Parameters that drive the entire rest of the design. **<length, width, height, depth, thickness, etc>**, and all the components are then created using these parameters. Sketches are placed on work planes which are also dependent on those parameters. All references are to very stable entities.

6 Misc Topics

This section is kind of a grab bag of general recommendations and other techniques that may help keep your design as stable as possible.

6.1 Design Iterations

Always throw away the first version, and re-do it. Any experienced designer will tell you the same thing. Start the design, fully intending to throw it away. You won't always have to, but if you start with this mindset, you won't be as frustrated when you do.

You will always learn a lot about the inherent nature of the design you are building when you try to build it: What parameters are likely to change, where the important relationships are, and what areas are less important to get right. You'll make mistakes that will be easier to avoid the next time than they are to fix on the first iteration.

6.2 Names and Groups

I like to think of this as an analogy to putting comments into source code, for a programmer. These techniques allow you to easily figure out your own design intent, which is very useful when trying to debug a model failure.

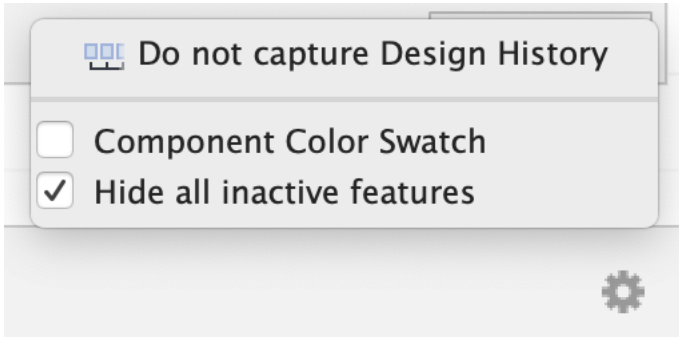
6.2.1 Name everything

Unfortunately, Fusion has no way to put comments on features in the Timeline. So, the only way to put information on these items is to put it into the feature name. I find this very helpful for myself, when I come back to a design that I worked on a long time ago, I can easily find which sketch matches to which feature, and what the feature is for. This can result in very long feature names, but to me this is worth it.

6.2.2 Group features from a component together

When making a multi-component design, I find it helps if you use Timeline Groups to groups sets of features which contribute to a single aspect of the design together using a Timeline Group. Say you have 5 features which together implement a fan grille for your design. Put them into a group.

Second, put all features that contribute to a local component near each other on the timeline. This will help you visually tell where one component starts and stops in the Timeline. This requires some discipline. When you go to add more features to a component after it has first been created, make sure to roll back the timeline marker to that section. **NOTE:** Component activation does **<u>not</u>** roll back the timeline, even if you have it set to "hide inactive features":



☐ Component Color Swatch
✓ Hide all inactive features

The setting only filters the Timeline to only show features from the active component, it does not change the insertion point for new features.

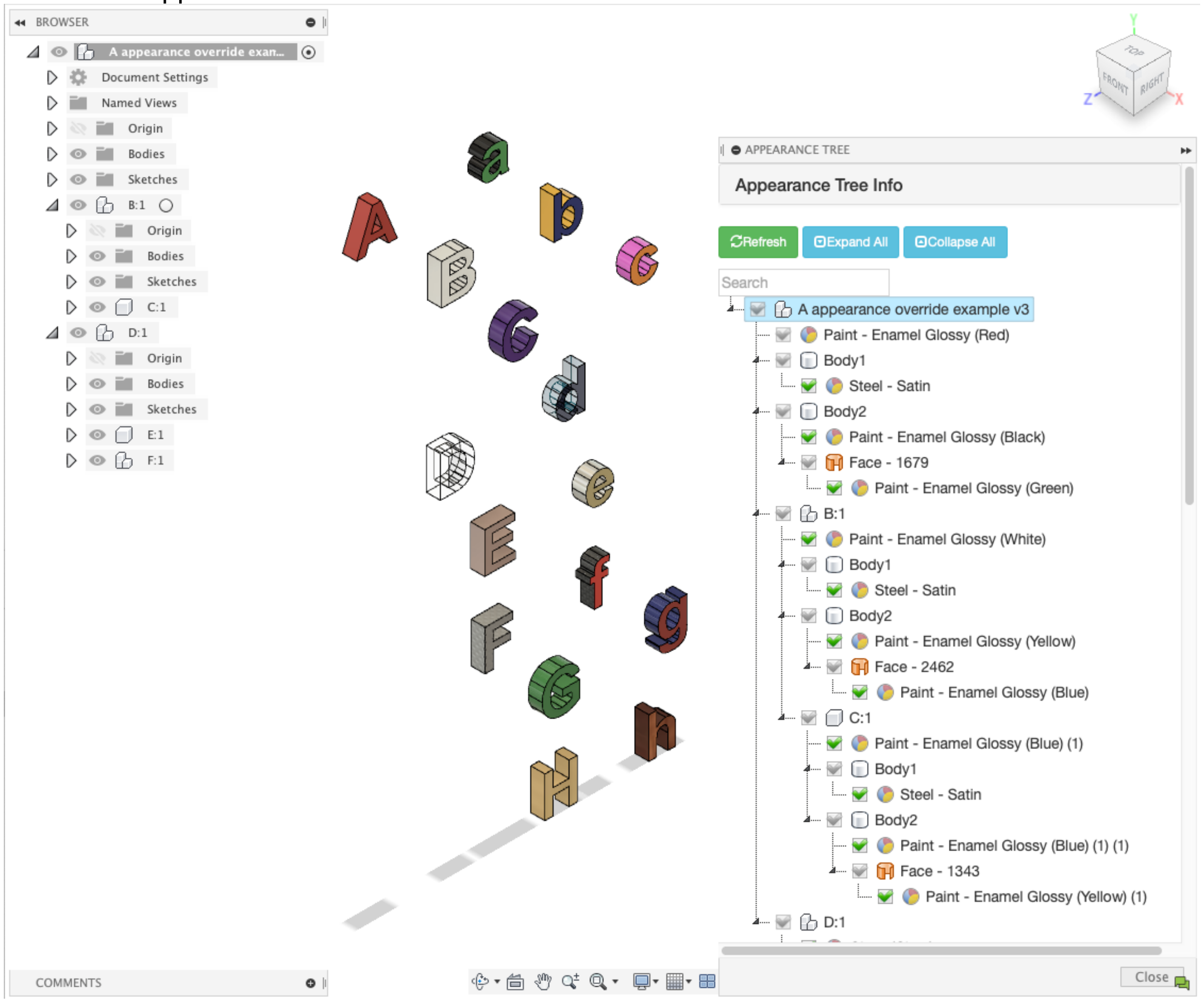
6.3 Debugging Appearances

Appearances in Fusion can be confusing and frustrating. Worse, there is no error system for appearances, because there is no "wrong" results, as far as Fusion itself is concerned. But, getting what you think is the correct visual results can be frustrating.

The important rule to understand with Appearances is the priority with which they are applied. This priority is basically bottom-up:

- Face appearance
- Body appearance
- Component appearance – this is applied recursively upward (or downward, depending on how you think about it). Part component appearance is higher priority than sub-assembly appearance, is higher priority than assembly appearance, etc.

The only way I've found to debug appearance issues is to use Patrick Rainsberry's Appearance Utility here: https://github.com/tapnair/AppearanceUtility. This produces a nice browser-like view of the appearance at ever level:


BROWSER
A appearance override exan...
Document Settings
Named Views
Origin
Bodies
Sketches
B:1
Origin
Bodies
Sketches
C:1
D:1
Origin
Bodies
Sketches
E:1
F:1
COMMENTS
TOP
FRONT
RIGHT
APPEARANCE TREE
Appearance Tree Info
Refresh
Expand All
Collapse All
Search
A appearance override example v3
Paint - Enamel Glossy (Red)
Body1
Steel - Satin
Body2
Paint - Enamel Glossy (Black)
Face - 1679
Paint - Enamel Glossy (Green)
B:1
Paint - Enamel Glossy (White)
Body1
Steel - Satin
Body2
Paint - Enamel Glossy (Yellow)
Face - 2462
Paint - Enamel Glossy (Blue)
C:1
Paint - Enamel Glossy (Blue) (1)
Body1
Steel - Satin
Body2
Paint - Enamel Glossy (Blue) (1) (1)
Face - 1343
Paint - Enamel Glossy (Yellow) (1)
D:1
Close

This UI allows you to toggle appearances on and off to see the effect they have on the model display

6.4 Cannot Delete a Design

Have you ever encountered a design which cannot be deleted? The error dialog says "cannot delete because this design is still referenced", but you've checked, and swear that it is not being referenced.

The root cause of this has to do with Fusion’s versioning model. Remember, Fusion keeps every version you have ever saved of a design. Second, you can Promote any of these older versions to be the current version. Well, the reference to this design still exists in older versions, so the design is still technically “in use”, just not in use by the current design.