SD473709

More Tips, Tricks, and the Future of the Forge Model Derivative Service

Kevin Vandecar – Developer Advocate
Autodesk

Denis Grigor – Developer Advocate
Autodesk

Learning Objectives

- Model Derivative サービスから来るデータセットを比較する方法をご紹介します。
- 部屋とスペースの情報を取得するための Revit generateMasterViews フラグについての詳細をご覧ください。
- Model Derivative サービスの新機能のプレビューを参照してください。
- 新しい SVF2 形式をいつ、どのように使用するかをご紹介します。

説明

昨年は "Forge の Model Derivative サービスのヒントとコツと未来 "をお届けしました。今年のプレゼンテーションには、さらに多くのグッズをお届けします。Revit フラグを使って部屋やスペースを生成して表示する方法を中心に、さらに詳しく取り上げていきます。Model Derivative データを使用してモデルのバージョン比較を行う方法を取り上げます。最後に、改良された SVF2 Forge Viewer 形式を含む Model Derivative サービスの新機能のいくつかをプレビューします。

スピーカー

Kevin Vandecar is a Forge developer advocate and also the manager for the Media & Entertainment and Manufacturing Autodesk Developer Network Workgroups. His specialty is 3ds Max software customization and programming areas, including the new Forge Design Automation for 3ds Max Service.

Denis Grigor likes to know how everything works under the hood, and is not afraid of low-level stuff like bits, buffers, pointers, stack, heap, threads, shaders and of course Math. He is interested in 3D for Web, from raw WebGL to libraries and frameworks with different levels of abstractions, as well as how virtual entities from 3D world can be linked to things from real world. In the end, "For an artificial mind, all reality is virtual" [The Animatrix]. To achieve my goals, he prefers to speak C/C++ mostly with modern dialect, Go, Python, and I have to speak Javascript/Node.js.

内容

Model Derivative サービス再入門

このクラスのタイトルが示すように、昨年の Autodesk University コース「Tips, Tricks, and the Future for Forge Model Derivative Services」の続編です。当時のコースと関連コンテンツはこちらからご覧いただけます: https://www.autodesk.com/autodesk-university/class/Tips-Tricks-and-Future-Forge-Model-Derivative-Services-2019。コースでは、Model Derivative サービスの基本と、Autodesk Forge のブログ記事やドキュメントなど、いくつかのソースからまとめられた入門的なヒントやコツをいくつか取り上げています。

ヒント: https://forge.autodesk.com ポータルもこの 1 年でいくつかの改善が行われました。まず、ブログ記事の特定の検索ができるようになりました。キーワードでの検索はもちろん、他のタグ付けされたコンテンツでも検索できるようになりました。例えば、Model Derivative サービスの最新ニュースやテクニックを見つけるには、「APIs and Services」のドロップダウンメニューから選択してください。また、Forge ポータル上のすべての関連コンテンツを検索するサイトワイド検索（右上隅にあります）もありますので、お試しください。generateMasterViews と入力すると、ドキュメントや stackoverflow の質問、ブログ記事など、関連するすべてのコンテンツを検索できるようになります。

Model Derivative サービスは、基本的には変換サービスであり、クラウドベースのシナリオで異なる形式のデザインを表現し、共有することを目的としています。SVF 形式は、Forge Viewer と共に使用することで、クラウドやモバイル環境での 3D のインタラクティブな表示を可能にする機能を提供しています。この機能によって、ソフトウェア開発者は、顧客やユーザーのためにリッチ データを活用した特定のワークフローを構築することができます。主なサービスは REST ベースの API を介して提供されます。また、Forge Viewer は JavaScript ライブラリを使用しているため、多くの環境で簡単に利用することができます。

Model Derivative API は、サポートされている形式を含め、下記にドキュメント化されています:
https://forge.autodesk.com/en/docs/model-derivative/v2/developers_guide/overview/

ヒント:この 1 年での改善点の 1 つは、変更履歴に固有の変更点を含めた点です。これらのメモは次の URL で参照することができます:
https://forge.autodesk.com/en/docs/model-derivative/v2/change_history/changelog/

Model Derivative からのデータセットの比較

同じモデルのバージョン間の違いを知ったり、異なるモデルの共通コンポーネントを識別したりすることは、非常に一般的なニーズであり、Forge ではこのニーズに対応するためのいくつかの方法が用意されています。

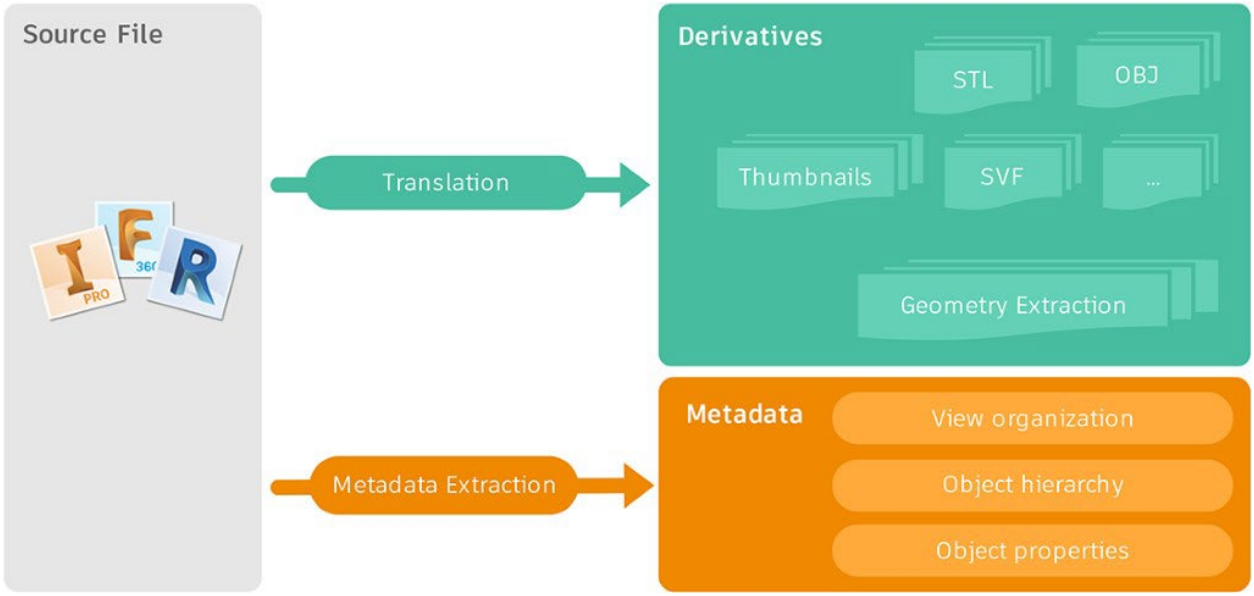
Forge Viewer のコンテキストには Autodesk.DiffTool Viewer エクステンションがあり、追加、削除、変更されたコンポーネントをハイライト表示することで変更を視覚化することができます。このエクステンションは、同じモデルやアセンブリのバージョン間の変更を説明するために Forge ベースの Team 製品（BIM 360 Team）で使用されていますが、Forge Viewer の独自アプリに簡単に統合することができます。

次のブログ記事で説明されています：
https://forge.autodesk.com/blog/difference-3d-models-autodeskdifftool-extension

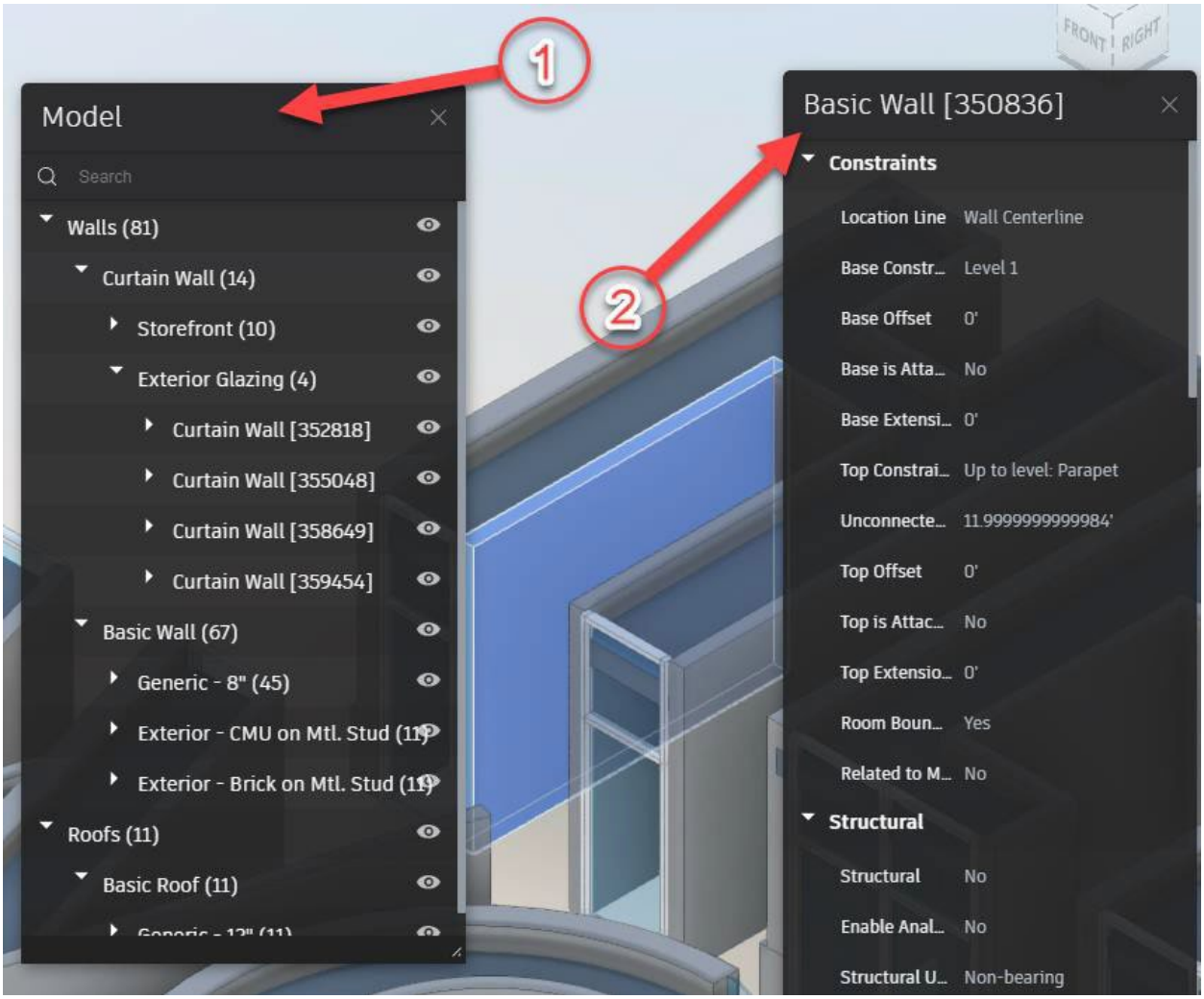
このアプローチの長所は、変更の視覚的な側面が自動的に表示される点です。主な短所は、ブラウザ上で実行されるインタラクティブな UI ベースのプロセスであることです。このアプローチでは、バッチプロセスの自動化が非常に困難です。

もう1つは、Design Automation エンジンを使用してモデルをロードし、同エンジン（Revit、Inventor、AutoCAD、または 3ds Max）のパワーを使用して入力シーンを処理し、変更点を特定する方法です。エンジンやワークフローによっては、ネイティブ ツール/プラグイン/スクリプトを使用して行うことができます（例：Comparing AutoCAD Drawings with Forge Design Automation）。より複雑な変更の場合は、プラグインの開発が必要になるかもしれません。
このアプローチの長所は、一度設定してしまえばパイプラインに統合しやすく、バッチ処理に優れていることです。プラグイン開発のためのリソースは、ユースケースの複雑さや処理されたファイル形式に応じて異なります。Revit と Inventor ファイルが混在している場合は、これら 2 つのエンジン用に別々のプラグインを開発する必要があり、作業できるファイル形式は、これらのエンジンが受け入れる形式と、インポート後に分析できる情報に制限されます。

ここで、Model Derivative サービスを使用したアプローチにたどり着きます。もちろん、このアプローチは変換の出力に基づくものとなるので、理想的ではないかもしれませんが、すでにいくつかの顧客のアプリケーションが使用、効果が証明されています。Model Derivative サービスは現在 70 以上の形式を受け入れており、SVF 形式を使用する際には共通の出力を提供しています。この形式にはメタデータ抽出が含まれており、このアプローチの鍵を握っています。



変換されたファイルを Forge Viewer で可視化すると、どのようなデータが抽出されているかを理解することが出来ます。



受け入れた形式には、2 つの重要な部分が生成されます。

1. モデルツリー(オブジェクト階層) - コンポーネントの階層的な関係を示す。
2. オブジェクトのプロパティ - 各コンポーネント、あるいはコンポーネント グループに対して生成されます。

Model Derivative サービスは、変換後、これらのメタデータを JSON ファイル（各モデルツリーとプロパティ）の形で利用できるようにします。

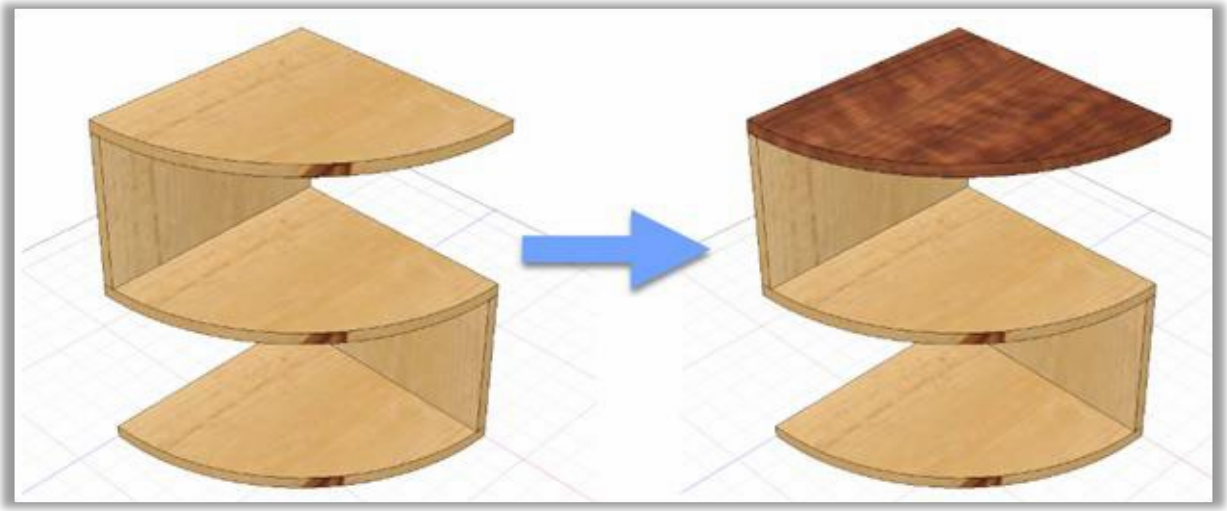
BIM360 ユーザーの特典は、通常、BIM 360 で利用可能なすべてのモデルがすでに変換されており、それを取得するために Model Derivative API を使用する必要があるという点です。

このメタデータは REST 呼び出しによって JSON 形式で利用可能になり、情報の比較が可能で違いを識別しやすい、という利便性が今回ご紹介する文脈です。

このアプローチの限界をよりよく理解するためにも、この方法でどのような変更や違いを捉えることができるのかを知ることが有用です。

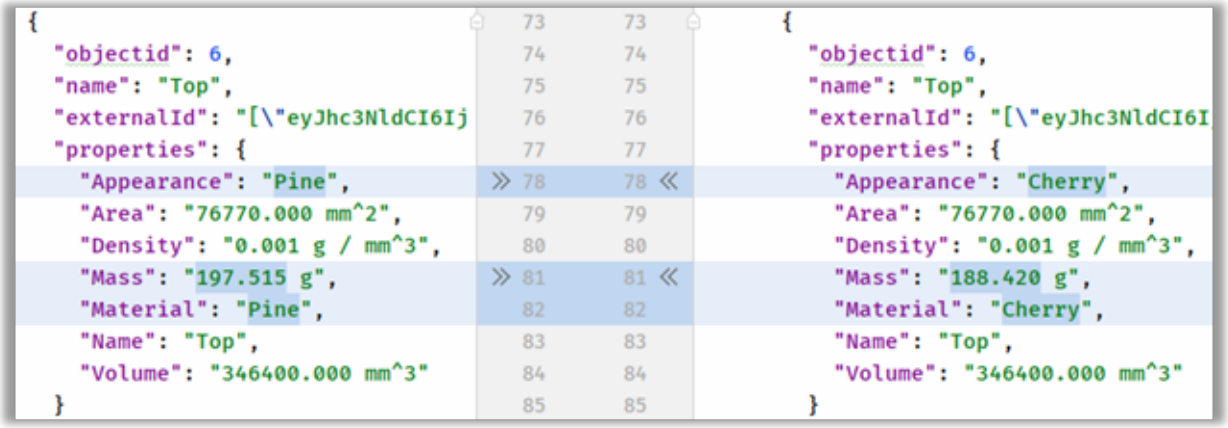
モデル内コンポーネントプロパティの変更

モデルのプロパティを変更した際に何が可能かを説明するために、上部の物理マテリアルを変更したコーナーシェルフユニットのモデルを見てみましょう。



これは Fusion モデルで、マテリアルの変化が密度などのプロパティに反映され、結果的にパーツ質量にも反映されます。

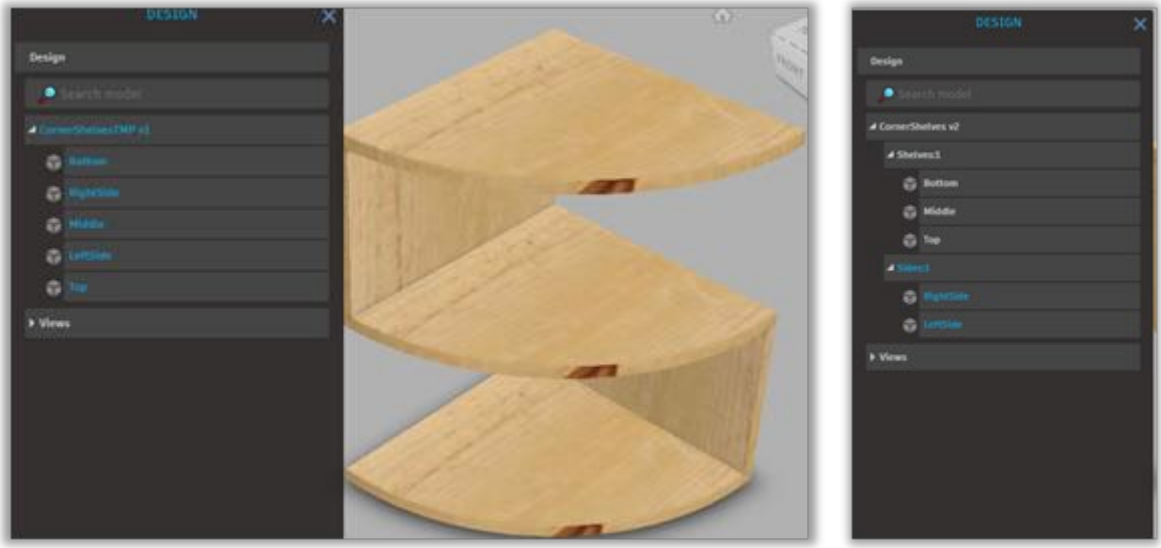
モデル階層ファイルを確認すると、明らかに何も変わっていませんが、オブジェクトのプロパティファイルを確認すると、簡単に変化がわかります。



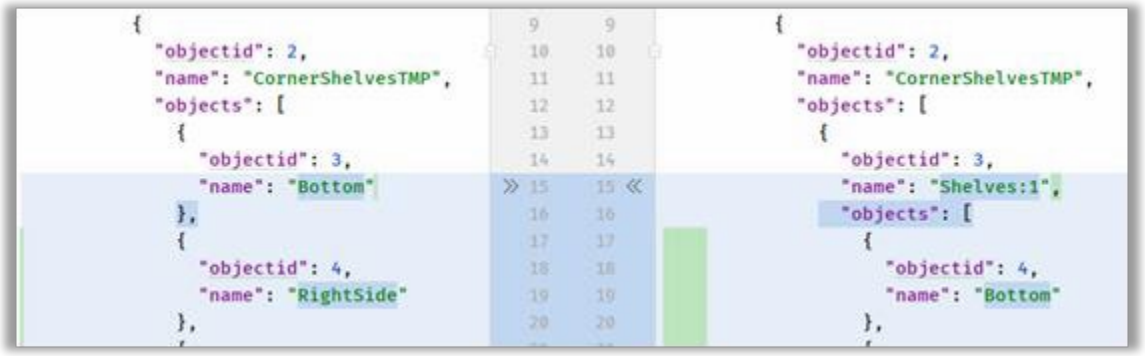
```
{                                          {
  "objectid": 6,                             "objectid": 6,
  "name": "Top",                             "name": "Top",
  "externalId": "[\"eyJhc3NldCI6Ij           "externalId": "[\"eyJhc3NldCI6I
  "properties": {                            "properties": {
    "Appearance": "Pine",                      "Appearance": "Cherry",
    "Area": "76770.000 mm^2",                  "Area": "76770.000 mm^2",
    "Density": "0.001 g / mm^3",               "Density": "0.001 g / mm^3",
    "Mass": "197.515 g",                       "Mass": "188.420 g",
    "Material": "Pine",                        "Material": "Cherry",
    "Name": "Top",                             "Name": "Top",
    "Volume": "346400.000 mm^3"                "Volume": "346400.000 mm^3"
  }                                          }
```

モデルの再構築

モデル/シーンをリファクタリングする場合、モデルはそのままですが、コンポーネント間の階層関係は変化します。

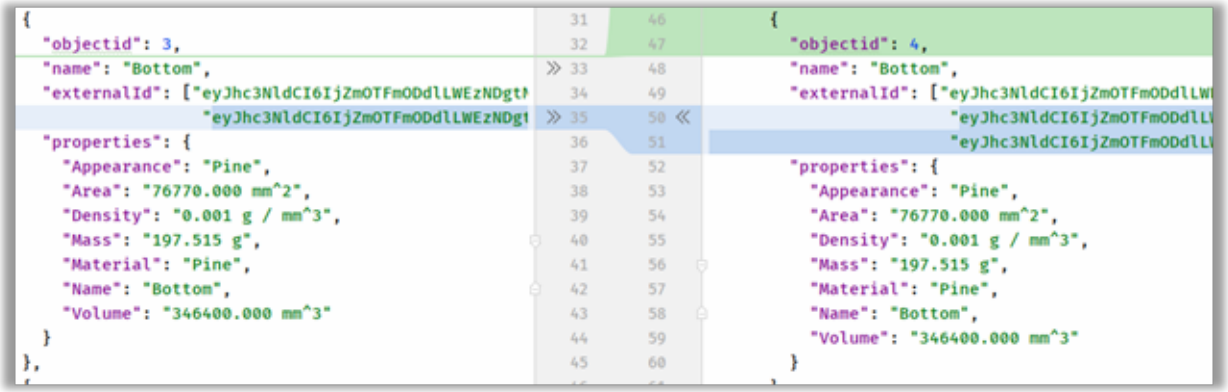


両ファイルのモデルツリーを取得して比較すると、再グループ化のためにいくつかのコンポーネントの ID が変更/シフトされていることに気付きます。"Bottom"の部分はかつて id = 3 を持っていましたが、2 つ目のモデルでは"Shelves"コンポーネントに移動されたので、間違いなく変更されています。



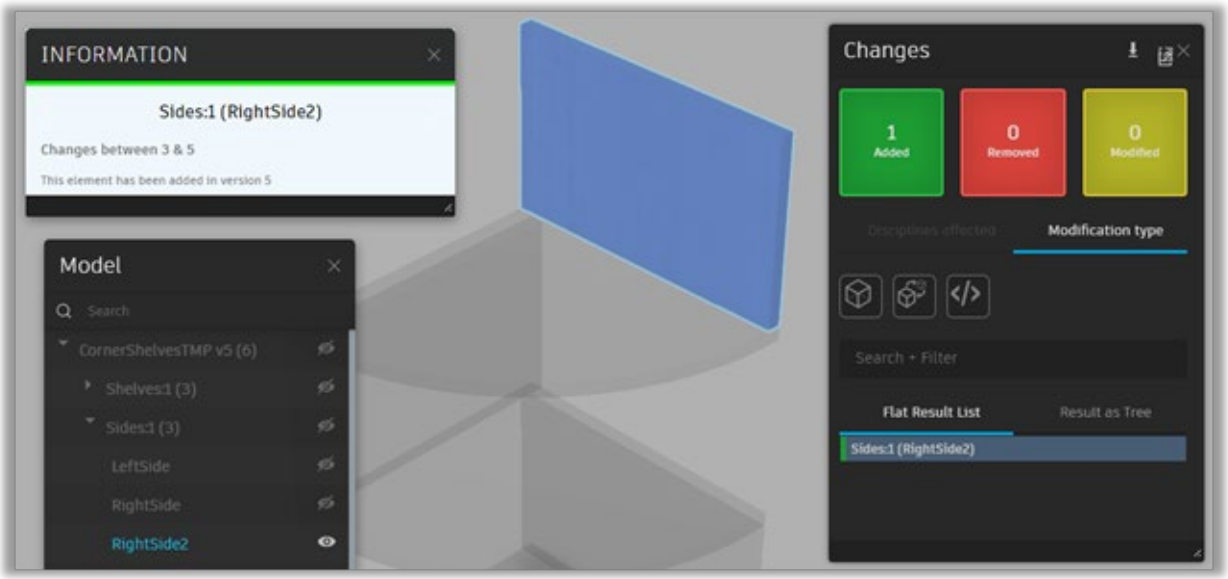
この概念のロジックは、Forge Viewer でもバックエンドのメタデータ処理でも、同じファイルの異なるバージョンでも id が同じ値であることに依存すべきではありません。この目的のため、external_id の参照が良いでしょう（サポートされていて利用可能であればの話ですが、すべての形式で提供されているわけではありません）。この例（後述）では、オブジェクトのプロパティファイルを調べてみると、プロパティはそのままに、objectid が変更されていることがわかります。しかし、externalid の配列を見ると、external id は変更を「生き残った」ことになります。

このアプローチは、このファイルを作成したアプリケーションが外部 ID メカニズムに依存していることと、パーツ/コンポーネントの外部 ID 履歴もエンコードしていることを想定しているため、少し厄介です。もしそれが利用可能で一致していれば、2 つの異なるモデルで同じパーツについて比較していることを確認することができます。



モデルへの新しいコンポーネントの追加

同じサンプルファイルを続けるために、別のパーツを追加してみましょう。視覚的には、これは次のように識別されます。

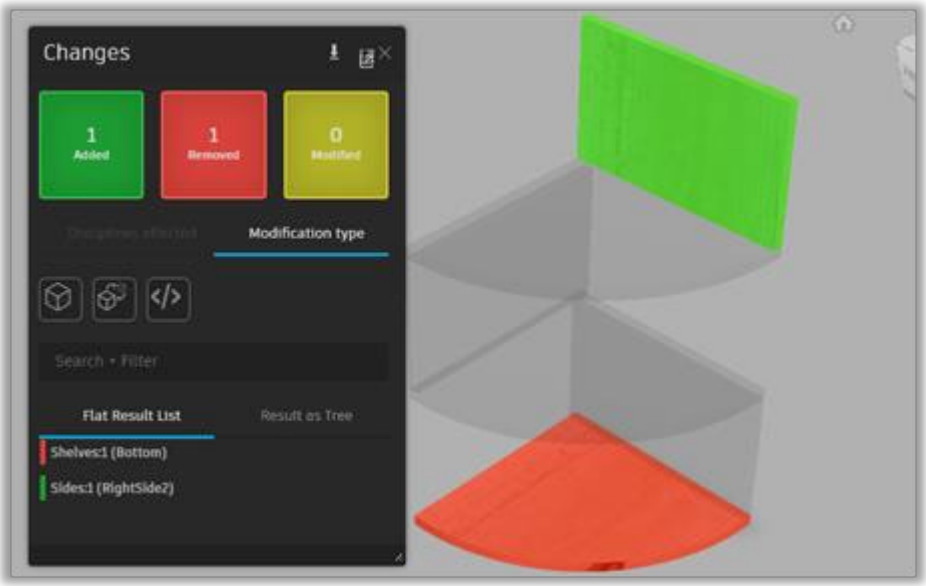


この部分が追加されたことを確認するために、改めてモデルツリーという観点から考えてみると、私たちにとってはモデルツリーを見ていれば十分なのです。

新しく追加された要素を見つけるのは簡単そうに見えますし、単純な場合には問題ありません。しかし、一度追加されたコンポーネントと削除されたコンポーネントが混在するようになると、内容は複雑になります。

モデルへのコンポーネントの削除と追加

次の例のように、パーツを削除するか、パーツの追加と削除を組み合わせて実行する場合：



モデルツリーが何らかの形でスクランブルされることが予想されますが、この状況でも、一般的なものをフィルタリングして変更に集中することで、それらを簡単に見つけることができます。

注：前述のように、objectid は、コンポーネントが追加/削除/並べ替えられた場合に「移動」する可能性があるため、信頼できる識別子ではありません。この理由から、別の一意な識別子を見つけることは必須です。名前だけで十分な場合もありますが、以下の例のように、"parentName "と "partName "を連結して作られた

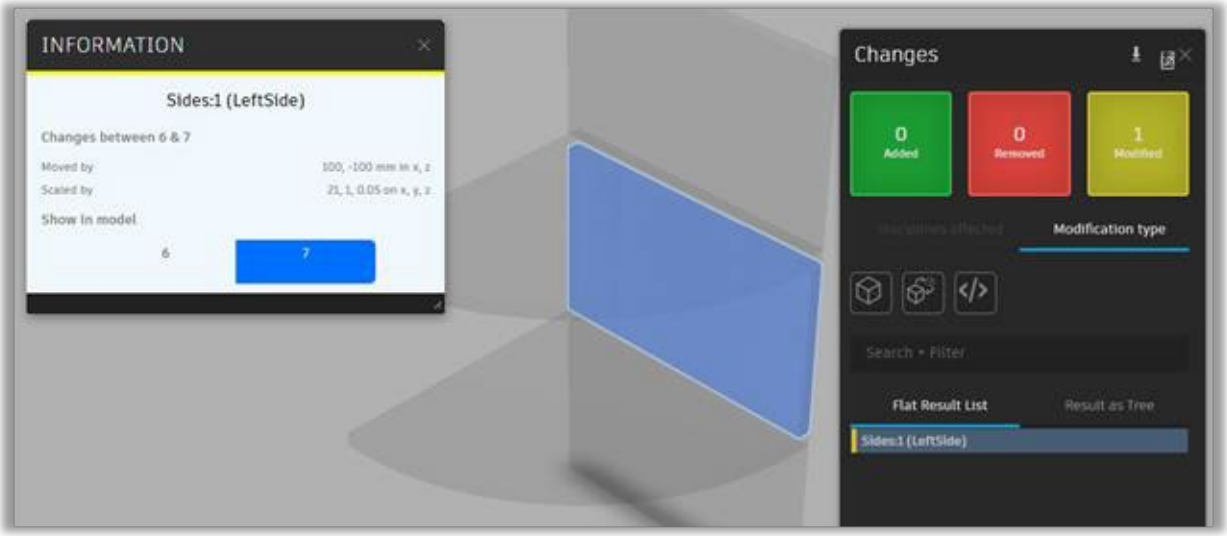
"Sides:1+LeftSide "のようなペアの名前のようなものを持つことで、名前を強化した方が良いでしょう。

```
"objectid": 7,
"name": "Sides:1",
"objects": [
  {
    "objectid": 8,
    "name": "LeftSide"
```

この場合、「LeftSide」の ID が変更されても、削除されていないか、階層が再配置されていることが簡単にわかります。
それでも、前述のように、安全な識別子は外部 ID（external Id）になりますが、アプリケーションがそれをサポートしていない場合、この回避策は問題の解決に役立ちます。

コンポーネントの変換（平行移動、拡大縮小、回転

コンポーネント階層を変更せず、また、プロパティを変更せずに、パーツを移動したり、回転したりするかにかかわらず、サイドパネルを回転させて他のサイドパネルと位置合わせするように移動するわずかな変換を行うと、次のようにモデルツリーJSON は以前のモデルとの違いを示さないと予想が出来ます。

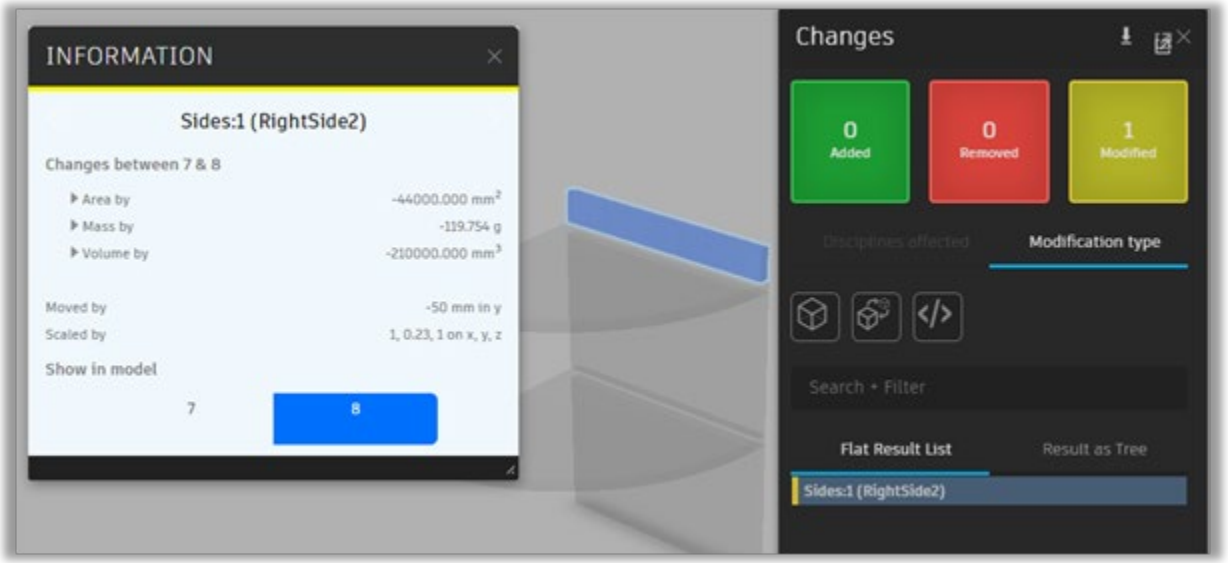


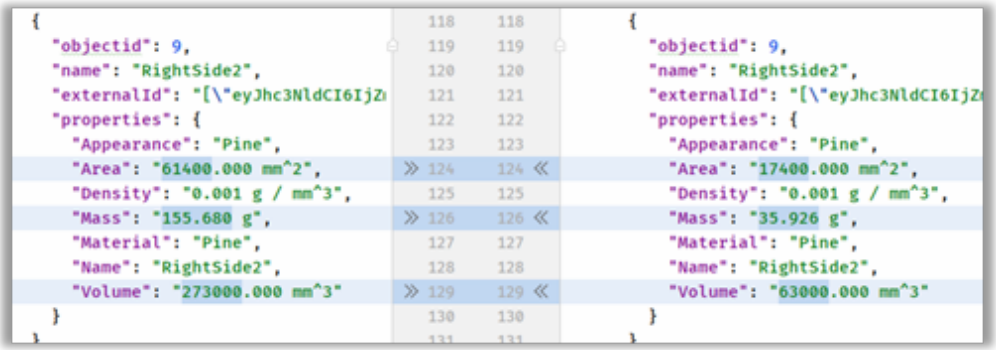
プロパティファイルを検査すると、ほぼ同じ画像が表示されます。変更されたのは、ルートノードの外部 ID だけです。

これはあまり役に立たないように見えるかもしれませんが、次の点を理解することが出来ます：

- 新しいコンポーネントが追加/削除/再配置されていません。そうでない場合、モデルツリーファイルが変更されます。
- プロパティは変更されていません。変更されていない場合、オブジェクトプロパティファイルが変更されます。.

ちなみに、これが「パラメトリックタイプのアプリケーション」（ここでは Fusion 360）の場合、パーツのサイズが変更されていないこともわかっています。そうでない場合は、プロパティの変更に反映されます。 たとえば、この場合、サイドパネルのサイズを変更すると、質量、面積、体積のプロパティに自動的に影響します。





残念ながら、これら 2 つのファイルだけでは、コンポーネントが移動または回転されたかどうか、またはファイルが別のバージョンで保存されたかどうかを確認できません（この外部 ID はファイルの変更時に再生成されるため）。

それでも、Model Derivative サービスを使用すると、パーツの形状が変更されたかどうかを調査して特定することが出来ます。面倒でやり過ぎに見えるかもしれませんが、ジオメトリの変更を把握する場合は、ジオメトリ自体を分析する必要があります。

ここでの場合、回転して平行移動した部分に戻ると、次のようになります。

モデルツリーからは変更されていないことがわかり、オブジェクトのプロパティファイルを比較することで、id の観点からは何も変更されていないことが確認できます。

Model Derivative API は、必要なパーツの id を提供することで、任意のパーツやグループの OBJ を抽出することができます。
POST *https://developer.api.autodesk.com/modelderivative/v2/designdata/job*

```
{
  "input": {
    "urn": "{{ENCODED_URN}}"
  },
  "output": {
    "formats": [{
        "type": "obj",
        "advanced": {
                    "modelGuid":"{{GUID}}",
                    "objectIds" : [7]
                }
    }]
  }
}
```

必要な部分の id を渡すことで、別の変換ジョブを起動して、その部分の OBJ を要求することが出来ます。
OBJ 形式の良いところは、それが「読める」ことであり、「読める」ということは比較出来るということです。

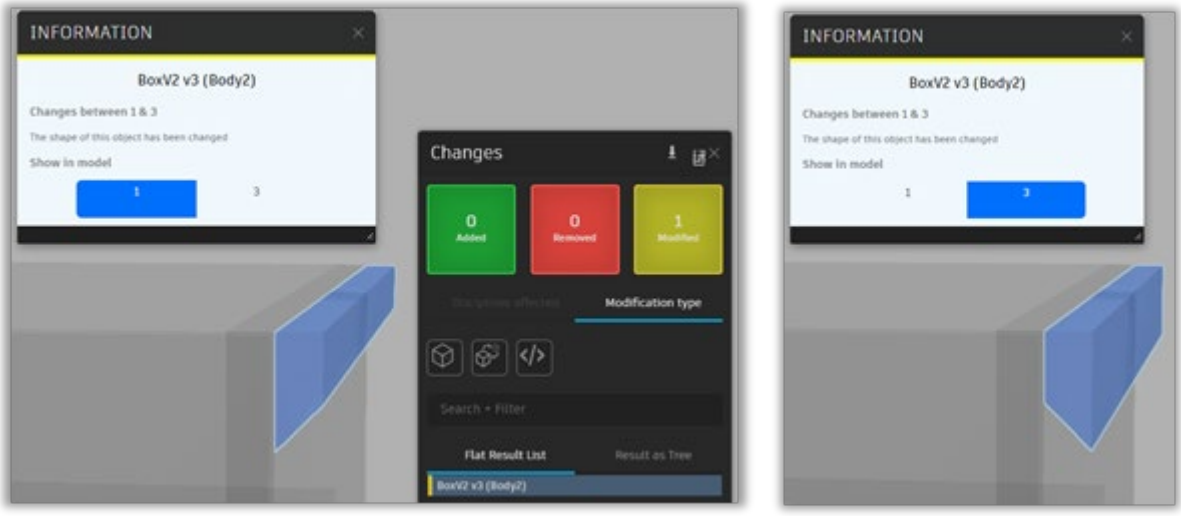
```
# WaveFront *.obj file (generated by Autodesk ATF)

g Obj.7

v 21.000000 28.000000 -0.000000
v -0.000000 28.000000 -0.000000
v 21.000000 28.000000 1.000000
v -0.000000 28.000000 1.000000
v 21.000000 15.000000 -0.000000
v 21.000000 15.000000 1.000000
```

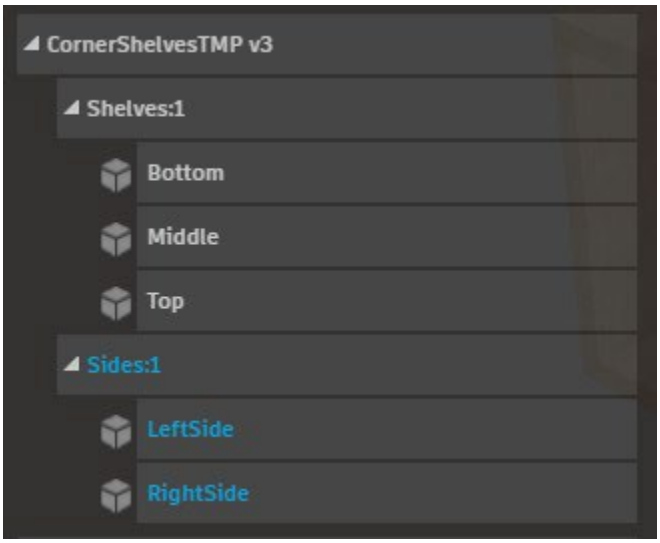
私たちの場合は、異なるモデルからの OBJ ファイルの ascii の内容を比較することで、ジオメトリが同じかどうかを知ることができれば十分です – 一致していれば同じであり、一致していなければおそらく変換されたものです。このように、最初の不一致を見つけるまでファイルを分析するだけで十分です。

```
# WaveFront *.obj file (generated by Autod   1   1   # WaveFront *.obj file (generated by
                                             2   2
g Obj.7                                      3   3   g Obj.7
                                             4   4
v 1.000000 28.000000 21.000000               5   5   v 21.000000 28.000000 -0.000000
v 1.000000 28.000000 0.000000                6   6   v -0.000000 28.000000 -0.000000
v -0.000000 28.000000 21.000000              7   7   v 21.000000 28.000000 1.000000
v -0.000000 28.000000 0.000000               8   8   v -0.000000 28.000000 1.000000
v 1.000000 15.000000 21.000000               9   9   v 21.000000 15.000000 -0.000000
v -0.000000 15.000000 21.000000              10  10  v 21.000000 15.000000 1.000000
v 1.000000 15.000000 0.000000                11  11  v -0.000000 15.000000 -0.000000
```

しかし、さらに進んで、変換のタイプを特定する必要がある場合は、これは良い出発点となります - それは変換されたか、回転されたか、またはその両方か。

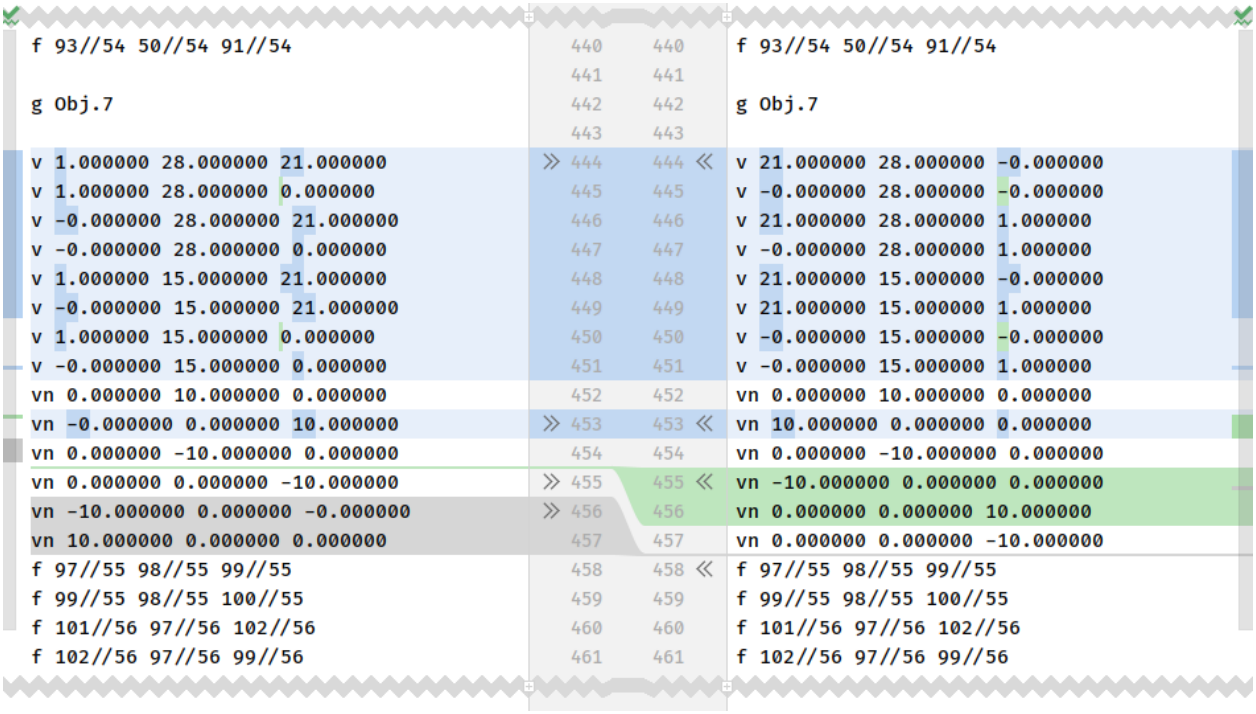
注意: このアプローチは、移動と回転だけを識別することに限定されていません。
頂点レベルでジオメトリを解析すると、テーパー角度、ツイスト、曲げ、鏡像反転などが施された部分を簡単に識別することができます。形状が変わってしまえば、それは顕在化します。





ヒント:最適化のために、各 ID を個別にチェックするのではなく、divide and conquer アプローチを採用することができます。このツリー関係では、Sides:1 ノードはジオメトリを持っていませんが(リーフ以外のノードはジオメトリを持っていません)、LeftSide と RightSide ノードの親ノードとしてジオメトリを持っています。ここで重要なのは、OBJ を求めて親ノードを提供すると、その子ノードのすべてのジオメトリが結果の OBJ に提供される(マージされる)ということです。したがって、ブランチレベルでジオメトリを比較することで、リーフのジオメトリが同じかどうかを識別するのに十分です。真偽の分析だけで、ジオメトリが変更されたコンポーネントの ID を識別することは簡単にできるはずです。同時に、これはルートレベルでも適用できます。ルートの OBJ を照会することで、頂点だけでなく、頂点のリストが属する部品の ID についても情報が得られます。多少の解析を伴いますが、この情報は取得するために存在します。

私たちのケースでは、id=1 (通常はルート) の OBJ の変更だけを分離して、"g Obj.7" を見ることで、それが id 7 の部分に属していることがわかります。



f 93//54 50//54 91//54	440	440	f 93//54 50//54 91//54
	441	441	
g Obj.7	442	442	g Obj.7
	443	443	
v 1.000000 28.000000 21.000000	444	444	v 21.000000 28.000000 -0.000000
v 1.000000 28.000000 0.000000	445	445	v -0.000000 28.000000 -0.000000
v -0.000000 28.000000 21.000000	446	446	v 21.000000 28.000000 1.000000
v -0.000000 28.000000 0.000000	447	447	v -0.000000 28.000000 1.000000
v 1.000000 15.000000 21.000000	448	448	v 21.000000 15.000000 -0.000000
v -0.000000 15.000000 21.000000	449	449	v 21.000000 15.000000 1.000000
v 1.000000 15.000000 0.000000	450	450	v -0.000000 15.000000 -0.000000
v -0.000000 15.000000 0.000000	451	451	v -0.000000 15.000000 1.000000
vn 0.000000 10.000000 0.000000	452	452	vn 0.000000 10.000000 0.000000
vn -0.000000 0.000000 10.000000	453	453	vn 10.000000 0.000000 0.000000
vn 0.000000 -10.000000 0.000000	454	454	vn 0.000000 -10.000000 0.000000
vn 0.000000 0.000000 -10.000000	455	455	vn -10.000000 0.000000 0.000000
vn -10.000000 0.000000 -0.000000	456	456	vn 0.000000 0.000000 10.000000
vn 10.000000 0.000000 0.000000	457	457	vn 0.000000 0.000000 -10.000000
f 97//55 98//55 99//55	458	458	f 97//55 98//55 99//55
f 99//55 98//55 100//55	459	459	f 99//55 98//55 100//55
f 101//56 97//56 102//56	460	460	f 101//56 97//56 102//56
f 102//56 97//56 99//56	461	461	f 102//56 97//56 99//56

まとめると、上記で議論した使用例に基づいて、Model Derivative サービスは、最も一般的な変更を判断するために必要な情報を提供していると結論づけられます。メタデータの変更を識別することに優れており、ジオメトリの変更を発見するために必要なデータをある程度提供しています。しかし、コンポーネントを再配置、追加、削除したモデルを比較する際には、多少の作業が必要です。
一般的なパーツと追加されたパーツを適切に識別するためには、視覚的なレビューがより有用であることがわかります（例えば、Forge Viewer エクステンションを使用するなど）。

Model Derivative アプローチは、Autodesk.DiffTool エクステンションを利用した Forge Viewer アプローチと比較して、ほぼ同じレベルの情報を提供しますが、この場合は「ヘッドレス」（UI なし）で実行されるため、大量のファイルの自動化やバッチ処理に理想的です。

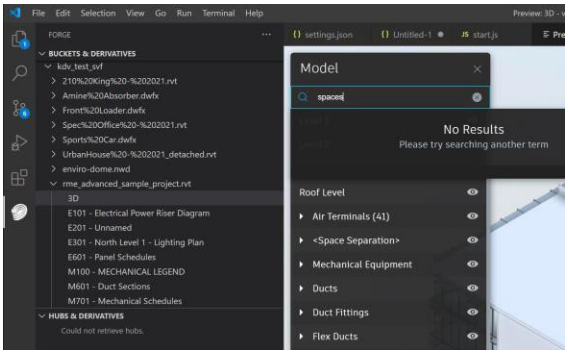
Design Automation アプローチと比較して、Model Derivative アプローチは、ネイティブ・ファイルと API 機能（特定の Design Automation・エンジンのコンテキストでは、はるかに豊かな）で作業していないので、あまり強力ではありません。

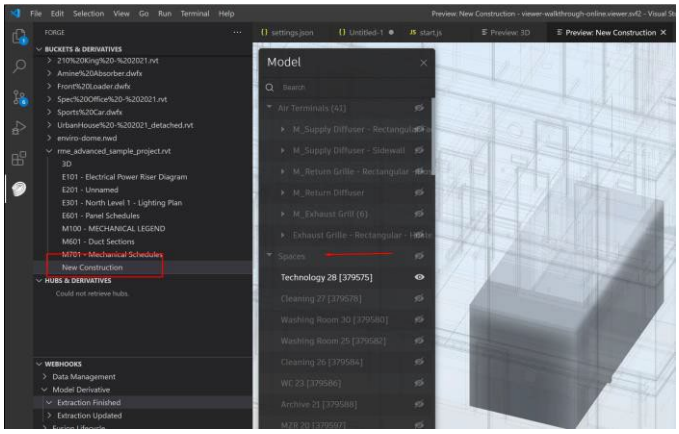
選択肢がより明確になると同時に、Model Derivative は多くの形式をサポートし、メタデータを扱うのに必要なリソース（時間と労力）が少なくて済むことがわかると思います。このアイデアにより、Model Derivative はモデル比較を自動化するための有効なオプションとなります。

AEC 入力タイプからの SVF 出力の改善

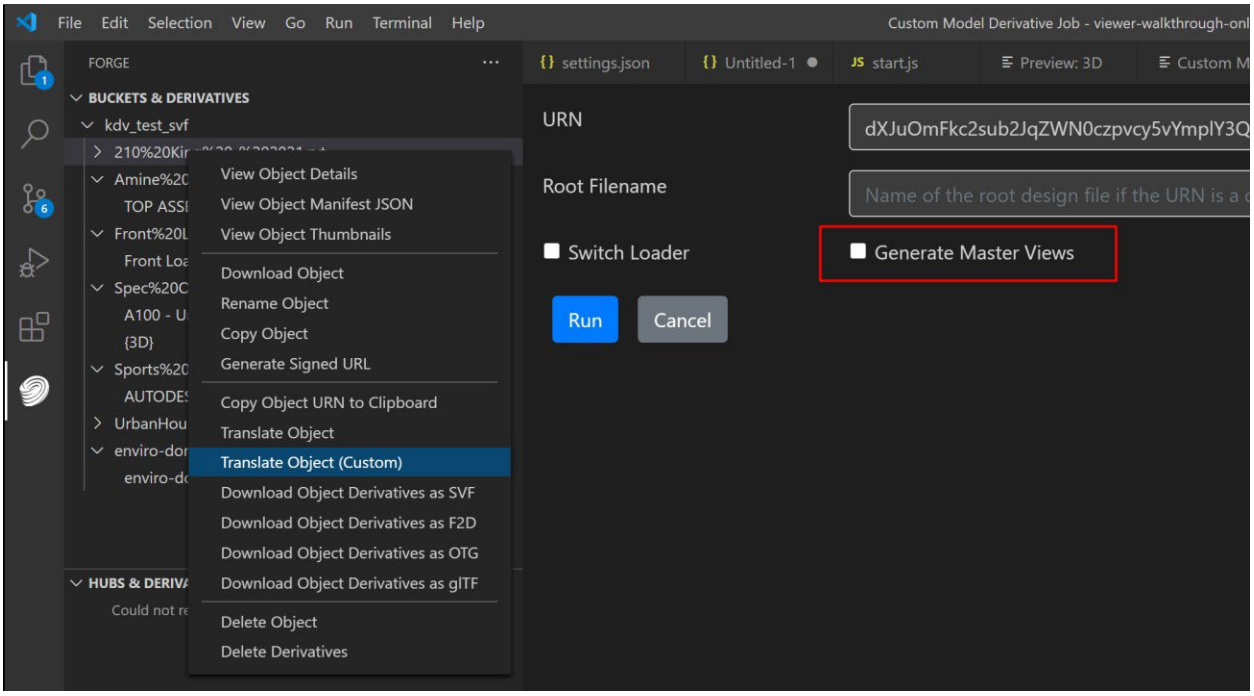
generateMasterViews 属性の使用

昨年、GenerateMasterViews と呼ばれる部屋/空間やゾーン情報も Revit モデルで変換できるようにする新しい属性を発表しました。マスタービューとは、Revit モデルの各フェーズごとに生成される viewable のことです。これには、そのフェーズのソースモデルに存在する部屋要素を含むすべての要素が含まれています。変換が完了すると、フェーズ情報から名前が付けられた追加の viewable が作成されます。これらの viewable の中には、追加の部屋/スペースとゾーンの詳細が含まれています。下記は、rme_advanced_sample_project.rvt ファイルのスクリーンショットで、generateMasterViews 属性の違いを示しています。左側が既定の動作（generateMasterViews なし）で、右側が generateMasterViews 属性を使用しています。"New Construction という新しい viewable があり、オブジェクトの "space"カテゴリが含まれていることに注目してください。





この 1 年の間に、Visual Studio Code Extension の generateMasterViews のサポートを追加し（紹介はこちら：https://forge.autodesk.com/blog/forge-visual-studio-code）、機能面での問題点もいくつか修正しました。私が作ったオリジナルブログはこちらです。(https://forge.autodesk.com/blog/new-rvt-svf-model-derivative-parameter-generates-additional-content-including-rooms-and-spaces)です。下記は VS Code 拡張での使用例のスクリーンショットです。



generateMasterViews はこちらでドキュメント化されています:
https://forge.autodesk.com/en/docs/model-derivative/v2/reference/http/job-POST/ (「advanced」セクションの「Case 2: Input file type is Revit」を参照してください。

また、このワークフローに特化したチュートリアルも用意されています:
https://forge.autodesk.com/en/docs/model-derivative/v2/tutorials/prep-roominfo4viewer/

この属性を使用した際の注意点は、少数のケースですべてのスペースが存在するわけではないという点です。この問題は、BIM 360 Docs の自動変換でも発生しました。原因は、エキスポータが奇妙な形や変形した形のスペースを理解できない場合、それ以上のスペースの変換を停止してしまうというものでした。この問題は、その後修正されています。
ただし、「誤った」スペースはスキップされてしまい、まだ行方不明になる可能性に注意してください。Viewer グラフィックスとして可視化が必要な場合は、スペースオブジェクトをより規則的な形状に再形成することをお勧めします。しかし、その場合、形状自体が正確なサイズと体積ではないことを覚えておいてください。将来的には、これをより良く処理するためのオプションを検討しています。検討されているオプションの 1 つは、スキップされたアイテムのオブジェクト ID をマニフェストにリストアップして、どのアイテムが欠けているかを判断し、解決方法を決定できるようにすることです。ここでの改善点については Forge ブログをご覧ください。

9 月の AEC アップデート

次のセクションでは、2020 年 9 月 7 日に実施されたアップデートについて説明します。これらは IFC、Navisworks、Revit ファイルでの変換に影響します。

非推奨の IFC switchLoader

昨年、IFC ファイル用の switchLoader 属性も導入しました。これは基本的に IFC ファイルを変換する際に Revit エンジンと Navisworks エンジンのどちらを使用するかを切り替えます。すべての変更点の詳細については、次の IFC->SVF についてのセクションを参照してください。

新しい IFC -> SVF 変換オプション

IFC から SVF 形式への出力を制御するために、"advanced "セクションにいくつかの新しい属性が追加されました。これについては、https://forge.autodesk.com/en/docs/model-derivative/v2/reference/http/job-POST/ (ケース 1: 入力ファイルの種類が IFC の場合) を参照してください。

これらのオプションを見てみましょう。

conversionMethod は switchLoader を置き換え、switchLoader よりも明確な名前になっています。両方が存在する場合、conversionMethod は上書きされます。後で switchLoader を失うことを避けるため、出来るだけ早く conversionMethod にアップデートすることをお勧めします。

- legacy は Navisworks IFC 抽出を使用しており、速度が遅くなります。
- modern は Revit IFC 抽出を使用し、より多くのオプションを提供するようになりました。

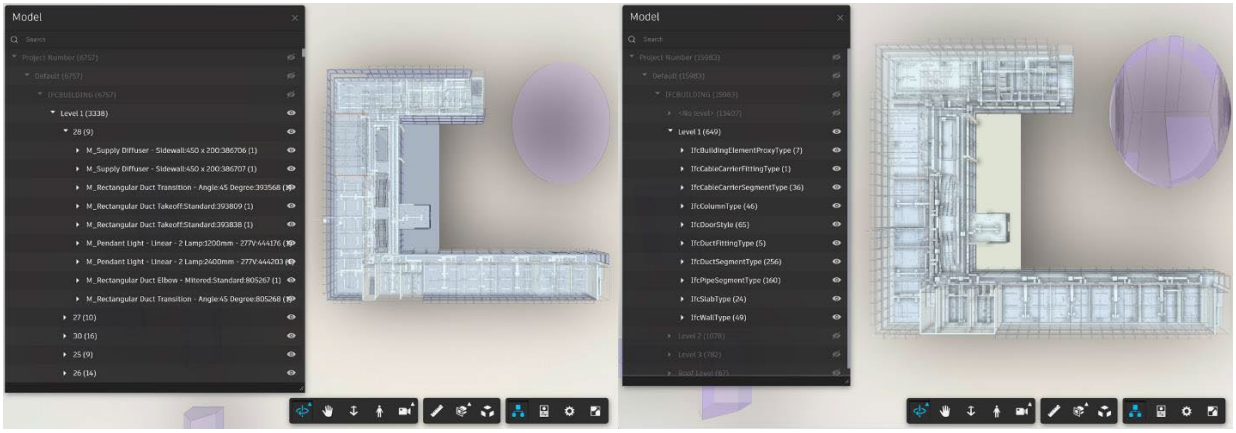
また、conversionMethod が modern に設定されている場合、ニーズに合わせて IFC 変換を最適化するために利用可能なオプションがいくつかあることにも注意してください。

- buildingStoreys は、階数の非表示、表示、スキップを可能にします（スペルは IFC の仕様と一致していることに注意してください）。
- spaces では、スペースの非表示、表示、スキップしたりすることができます。
- openingElements は、オープンを非表示にしたり、表示したり、スキップしたりすることができます。

まず第一に、何もしない（属性がない）場合は、legacy の Navisworks 抽出を使用することになります。Navisworks 抽出の方が高速で、Revit トランスレータと同様の結果が得られます。ただし、Revit トランスレータには特定の利点があります。まず、IFC 標準をより厳密かつ正確に再現します。また、より最新のものであり、定期的に改善されています。また、Revit 抽出は、Navisworks よりも「Swept サーフェス」を処理します。最後に、Revit 抽出を使用することで、buildingStoreys、spaces、openElements の側面を制御するという追加の利点を得ることができます。

これが変換にどのような影響を与えるのでしょうか？Revit サンプルの rme_advanced_sample_project.rvt を IFC データセットとして考えてみましょう（Revit から IFC としてエクスポートされます）。生の IFC はテキスト形式なので、同じデータセットでもかなり大きくなります（この場合は 100MB 以上）。

デフォルトの Navisworks 抽出では 172.33 秒かかりました。Revit IFC 抽出では 340.05 秒かかりました。他の数人と議論した結果、現在の最新の（Revit の）抽出は、一般的に平均して約 2〜4 倍の時間がかかることに同意しました。しかし、結果、特にモデル構造は、"現代の Revit 変換の方がはるかに優れています。



conversionMethod = legacy (Navisworks) (左)、conversionMethod = modern (Revit) (右)

Navisworks 変換エンジンの更新

まず、パフォーマンスが向上したのはもちろんのこと、大きなファイルを変換する際にタイムアウトのエラーメッセージが出た時に多くの問題が解消されています。

また、レンダリングの設定も現在検討されています。以前はこれが完全に無視されていて、Navisworks の設定とは大きく異なるように見える結果が Forge Viewer に与えられていました。

既定ではコンバージェンス時間を改善する様々なものを処理するための追加設定が追加されましたが、今では自由にコントロールして使用することが出来るようになりました。これらの設定は、materialMode、hiddenObjects、basicMaterialProperties、autodeskMaterialProperties、timelinerProperties です。

詳細は https://forge.autodesk.com/en/docs/model-derivative/v2/reference/http/job-POST/ を参照し、「ケース 3: 入力タイプが Navisworks の場合」までスクロールしてください。

Model Derivative 固有の Webhooks

Webhook システムは Model Derivative よりも後から導入されたもので、前回のプレゼンテーションでは議論していませんでしたので、ここで指摘したいと思います。現在のところ、チュートリアルやサンプルのドキュメントではどれもこれを紹介しておらず、"Webhooks"セクションのみに記載されていますので、見落としているかもしれません。これは、Model Derivative ジョブを操作するのに非常に便利な方法です。

例えば、"Translating a Source file "のチュートリアルでは、マニフェストを使って進捗状況を確認する方法を紹介しています。https://forge.autodesk.com/en/docs/model- derivative/v2/tutorials/translate-to-obj/task3-translate-source-file/ と、最後のセクションでは繰り返しマニフェストを要求して "status "と "progress "を確認しています。イオンファクトでは、ほとんどのサンプルがこれを行っています。

2 つの Model Derviative Webhook があり、このような繰り返しのマニフェストへのステータス取得要求を回避するのに役立ちます。例えば、ジョブが終了したことだけを本当に気にしている場合は、extraction.finished を使用できます。また、通常のステータスを知りたい場合は、extraction.updated を使用できます。

Webhooks のドキュメント開始点:
https://forge.autodesk.com/en/docs/webhooks/v1/developers_guide/overview/
具体的なイベントの詳細:
https://forge.autodesk.com/en/docs/webhooks/v1/reference/events/model_derivative_events/
チュートリアル:
https://forge.autodesk.com/en/docs/webhooks/v1/tutorials/create-a-hook-model-derivative/

ここでは、Webhook を設定する簡単な例のコードを紹介します（これは、Express と Axios という、REST API を直接明確に呼び出すことができるライブラリを使用したオンライン Viewer アプリのチュートリアルに追加されています）。チュートリアルはこちらです: https://forge.autodesk.com/developer/learn/viewer-app/overview.

ステップ 1. webhook とコールバックを設定します。これは extraction.finished を使用していることに注意してください。イベントを設定し、ワークフロースコープを 'workflow-extraction-complete' に設定します。

```
Axios({
    method: 'POST',
    url: 'https://developer.api.autodesk.com/webhooks/v1/'+
            'systems/derivative/events/extraction.finished/hooks',
    headers: {
        'content-type': 'application/json', Authorization: 'Bearer ' +
        access_token
    },
    data: JSON.stringify({
        // This would be a real location on your sever when deployed.
        // For local server ngrok redirects to your localhost. 'callbackUrl':
        'http://8a1524b9b9cd.ngrok.io/callback/jobfinished',
        // Scope is important to identify the specific job callback you want.
        // See the job API call. 'scope' : {
            'workflow': 'workflow-extraction-complete'
        }
    })
})
```

ステップ 2. コールバック関数を作成します。先ほどのコードでは /callback/jobfinished です。次の例では

```
app.post('/callback/jobfinished', jsonParser, async (req, res, next) => {
    // Best practice is to tell immediately that you got the call
    // so return the HTTP call and proceed with the business logic
    // From Augusto Goncalves
    res.status(202).end();

    // Get the hookId... You can get from the response when creating, too.  var
    mdFinishedHookId = req.body.hook.hookId;
    // Was the job successful at end of the job?
    var mdFinishedStatus = req.body.payload.Payload.status;

    console.log('hit...' + urn);
    console.log('status...' + mdFinishedHookId + ' : ' + mdFinishedStatus);

});
```

ステップ 3. ジョブ API を使用する際に、ボディの misc 属性でこのジョブのワークフロースコープを指定することができます。:

```
Axios({
    method: 'POST',
    url: 'https://developer.api.autodesk.com/modelderivative/v2/designdata/job', headers: {
        'content-type': 'application/json', Authorization: 'Bearer ' +
        access_token
    },
    data: JSON.stringify({ 'input': {
            'urn': urn
        },
        'output': {
            'formats': [
                {
                    'type': 'svf', 'views': ['2d',
                    '3d']
                }
            ]
        },
        "misc": { // webhook callback
            "workflow": "workflow-extraction-complete"
        }
    })
```

ステップ 4. 最後に、webhook/callback 機能の使用が終わったら、それを削除します。:

```
// delete the webhook... Axios({
    method: 'DELETE',
    url: 'https://developer.api.autodesk.com/webhooks/v1/' +
    'systems/derivative/events/extraction.finished/hooks/'+ mdFinishedHookId,  headers: {
        'content-type': 'application/json', Authorization: 'Bearer ' +
        access_token
    }
})
```

3ds Max 物理 マテリアルの Model Derivative SVF 形式サポート

3ds Max は、オートデスクの主要なデスクトップ ビジュアライゼーション ソフトウェアの 1 つであり、オンライン オートデスク ビューイングの忠実度体験をリードしています。最近では、3ds Max の Model Derivative が改良され、ラージ モデル ビューイング テクノロジが強化されたことで、Viewer がレンダリングできる形式に変換された物理 マテリアル(PBR)のサポートが可能になりました。

この機能を使用するには、3ds Max シーンで物理マテリアルを設定し、Model Derivative ジョブに送信して SVF に変換する必要があります。これらの機能については、最近の 2 つの Changelog エントリを参照してください。

- https://forge.autodesk.com/en/docs/model-derivative/v2/change_history/changelog/#release-date-2020-08-27
- https://forge.autodesk.com/en/docs/model-derivative/v2/change_history/changelog/#release-date-2020-06-02

モデルの 3ds Max ビューポートレンダリングを示す図 1 のスクリーンショットを参照してください。図 2 の画像は、3ds Max *.MAX 形式を使用して変換され、その後 Forge Viewer でレンダリングされた同じモデルです。図 3 のスクリーン ショットは、以前に FBX 形式を使用していたことを示しています（トランスレータが物理マテリアルをサポートしていません）。最後に図 4 は、Model Derivative サービスで変換された PBR モデルを Forge Viewer で表示している別の例を示しています。

図1

図2


Forge Viewer showing 3ds Max scene exported
to FBX with materials embedded

図3

図 4

これらのモデルのクレジットは、https://www.turbosquid.com/ で公開されているアーティストによるものです。

一般的なデジタルカメラ一眼レフ- by 3d_molier International
https://www.turbosquid.com/3d-models/digital-camera-slr-generic-3d-model/934661

男性用 SF スーツ - by Dyasharuku
https://www.turbosquid.com/FullPreview/Index.cfm/ID/1121012

前回のクラスでも触れましたが、3ds Max のシーンファイルをマテリアルや xref などで処理する最も簡単な方法は、アーカイブ形式、つまり ZIP 出力を利用することです。例えば、上記のスクリーンショットの例のカメラシーンを使用して、"Archive..." オプションを選択します。図 5 では 3ds Max 2021 のメニューにこのように表示されています。

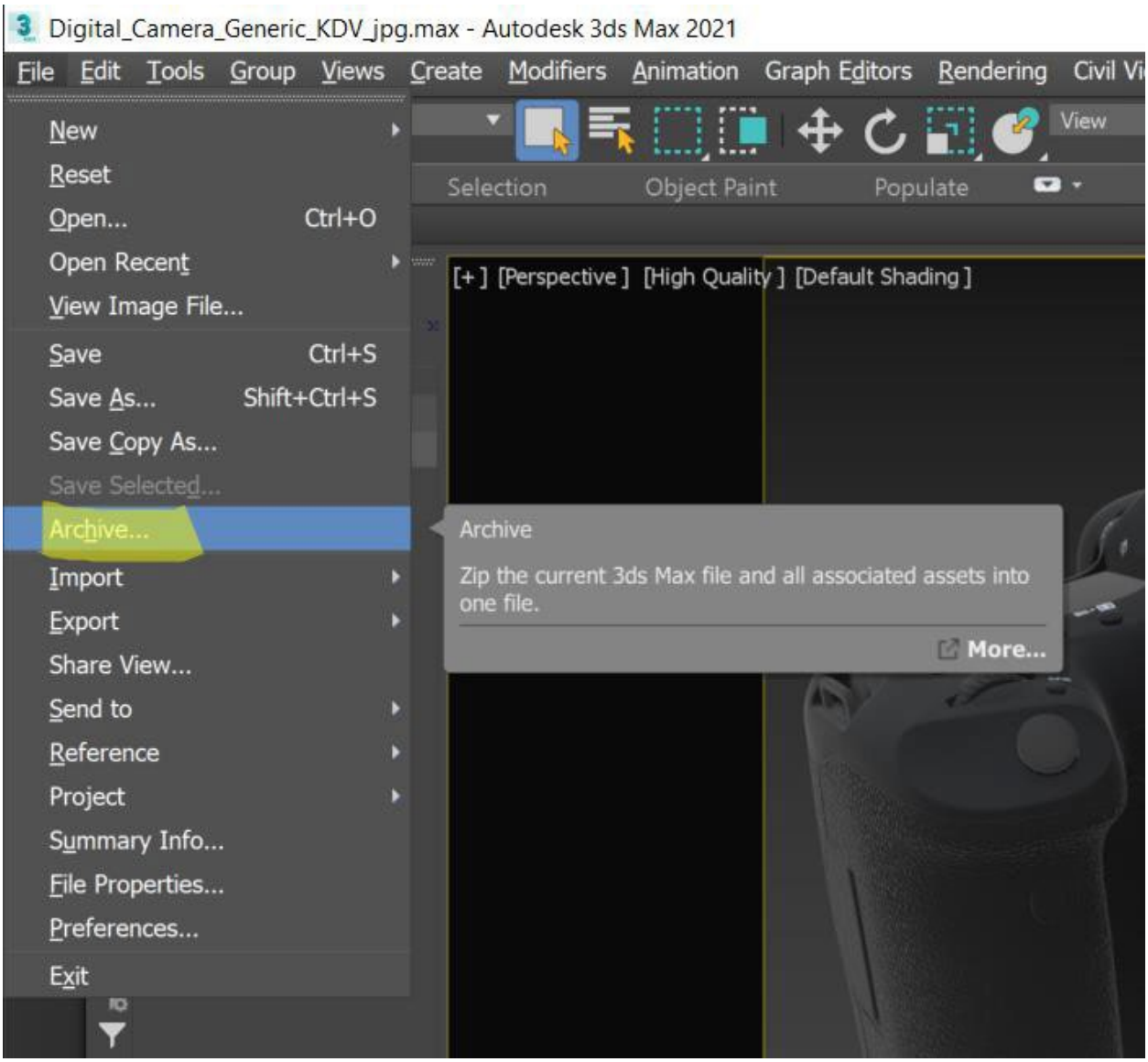


図 5

そして、そのジョブを Model Derivative POST https://developer.api.autodesk.com/modelderivative/v2/designdata/job エンドポイントに送信するために、リクエスト・ボディの JSON ペイロードは図 6 のようになります。compressedUrn 属性が true に設定されていることと、メイン・シーン名を示す rootFilename 属性に注意してください。シーンが物理マテリアル用に設定されている場合、それらは Forge Viewer。最低でも Forge Viewer バージョン 7.13.1 を使用することを忘れずに、Viewer のすべての現在の機能を使用するには、最新バージョンを使用することをお勧めします。

```
{
   "input": {
     "urn": "{{Base64URN}}",
     "compressedUrn": true,
     "rootFilename": "camera.max"
   },
   "output": {
     "formats": [
       {
         "type": "svf",
         "views": [
           "2d",
           "3d"
         ]
       }
     ]
   }
 }
```

新しい SVF2 形式

昨年の Autodesk University 2019 では、Forge Viewer のパフォーマンスを向上させるために研究されていた OTG というコードネームの新しい形式を発表しました。いくつかのデモが行われましたが、この取り組みについての短い言及は、こちらの Forge Keynote のディスカッションで見つけることができます (https://youtu.be/c8-AxaoHDIk?t=1147)。このビデオセグメントでは、Susanna Holt 氏が戦略的パートナーがより大きなモデルを扱いたいと考えていたことを説明しており、そこで OTG 形式が生まれました。その後、オートデスクは OTG 形式を SVF2 形式にして、AU 2020 までにパブリックベータとして利用できるよう予定しています。詳細は https://forge.autodesk.com/blog をチェックしてみてください。

SVF2 とは?

SVF 自体はすでに WebGL に最適化された形式です。Three.js ライブラリを使用した Forge Viewer は、ブラウザやモバイルアプリケーションで非常に大きなデザインを表示することができます。オートデスクのビューイング技術のユーザー側では、LMV（Large Model Viewer）と呼ばれることが多く、BIM 360 Docs や Fusion 360 のビューイング用ハブなど、他の多くの製品で利用されています。Model Derivative サービスや Forge Viewer も同じ技術を使用して、カスタム アプリやワークフローに同じ表示体験を提供しています。

SVF2 は、反復形状を含むモデルの読み込み時間を短縮するために SVF 形式を最適化しています。AEC 業界（建築、エンジニアリング、建設）で一般的に作成される大規模モデルは、このカテゴリに該当します。現在、Model Derivative サービスでは、最初に SVF を生成し、同じ Viewer ブルでメッシュを共有し、可能な場合は複数の Viewer ブルにまたがってメッシュを共有することでさらに最適化することで SVF2 派生を生成しています。この最適化により、SVF2 形式では、viewable のストレージサイズを大幅に削減することができます。また、インクリメンタルロードや複数のバージョン間での高速な切り替え、異なるバージョンをより細かい粒度で比較するなどの操作も可能になります。余分な処理があるため、SVF2 への変換には時間がかかります。さらに、SVF で可能な変換後の操作はすべて SVF2 でも可能です。例えば、メタデータの抽出やジオメトリの抽出などです。さらに、メタデータの抽出とジオメトリの抽出のプロセスは、どちらの形式でも同じです。

今のところ、要求があれば SVF 形式から直接 SVF2 形式が生成されます。SVF2 の生成は基本的に後処理/最適化です。

SVF と比較した SVF2 のパフォーマンス向上の例は以下を参照してください。ご覧のように、ほとんどの場合、SVF2 形式は SVF 形式を凌駕しており、わずかな追加時間しかかかりません。しかし、追加の時間がかかることに注意してください。理想的には、モデルがかなり大規模な場合には、そのメリットとコストを比較して使用すべきです。

3 つの API が SVF2 をサポートするように更新されます。:

GET https://developer.api.autodesk.com/modelderivative/v2/designdata/formats
POST https://developer.api.autodesk.com/modelderivative/v2/designdata/job
GET https://developer.api.autodesk.com/modelderivative/v2/designdata/:urn/manifest
- or -
GET https://developer.api.autodesk.com/modelderivative/v2/regions/eu/designdata/:urn/manifest

例えば、POST job エンドポイントの使用では、形式だけが異なることがわかります。

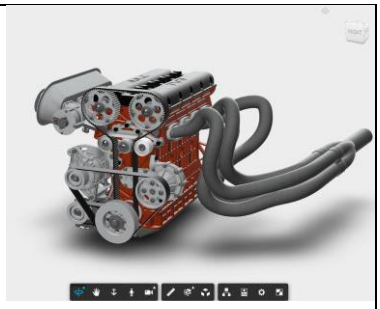
```
Axios({
    method: 'POST',
    url: 'https://developer.api.autodesk.com/modelderivative/v2/designdata/job', headers: {
        'content-type': 'application/json', Authorization: 'Bearer ' +
        access_token
    },
    data: JSON.stringify({ 'input': {
            'urn': urn
        },
        'output': {
            'formats': [
                {
                    'type': 'svf2',
                    'views': ['2d', '3d']
                }
            ]
        }

    })
})
```

また、viewer.model.isSVF2() を呼び出して、読み込まれた viewable が SVF2 viewable であるかどうかを確認することもできます。これはすでにここでドキュメント化されています:
https://forge.autodesk.com/en/docs/viewer/v7/reference/Viewing/Model/#issvf2

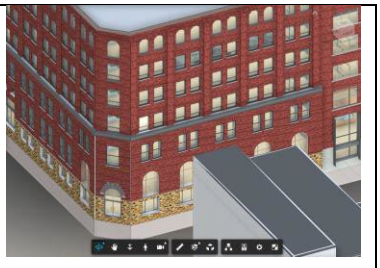
ここでは、SVF と SVF2 の統計的な違いをいくつか例示します。特に、Revit と Navisworks の例では、メッシュ数に大きな違いがあることがお分かりいただけると思います。

Model	Front Loader.dwfx	
Model File Size	10.3 MB	
Translation Type	SVF2	SVF
Translation time	66.29 seconds	20.78 seconds
Loading Viewable	0.57 second	1.03 seconds
Total geometry size:	27.179 MB	30.630 MB
Number of meshes:	1701	1919
Num Meshes on GPU:	1701	1919
Net GPU geom memory used:	27928096	31473282

Model	engine_type_01_2_liter_asm.zip	
Model File Size	62.8 MB	
Translation Type	SVF2	SVF
Translation time	83.04 seconds	53.97 seconds
Loading Viewable	0.77 seconds	1.16 seconds
Total geometry size:	24.084 MB	34.559 MB
Number of meshes:	307	549
Num Meshes on GPU:	307	549
Net GPU geom memory used:	25150610	36053286



Model	210 King - 2021.rvt	
Model File Size	92.8 MB	
Translation Type	SVF2	SVF
Translation time	488.49 seconds	385.35 seconds
Loading Viewable	0.60 seconds	1.51 seconds
Total geometry size:	22.589 MB	166.515 MB
Number of meshes:	3362	29646
Num Meshes on GPU:	3362	10000
Net GPU geom memory used:	22556672	101428240



Model	UrbanHouse - 2021_detached.rvt		
Model File Size	28.4 MB		
Translation Type	SVF2	SVF	
Translation time	149.58 seconds	110.2 seconds	
Loading Viewable	0.19 seconds	0.86 seconds	
Total geometry size:	8.414 MB	17.413 MB	
Number of meshes:	776	3090	
Num Meshes on GPU:	776	3090	
Net GPU geom memory used:	8561832	17221052	

Model	ice stadium.nwd		
Model File Size	2.4 MB		
Translation Type	SVF2	SVF	
Translation time	60.62 seconds	40.49 seconds	
Loading Viewable	0.54 seconds	1.26 seconds	
Total geometry size:	4.481 MB	12.342 MB	
Number of meshes:	3854	13333	
Num Meshes on GPU:	3854	10050	
Net GPU geom memory used:	3403678	6572896	

Autodesk University で SVF2 のベータ版が利用できるようになると思われます。詳細は forge.autodesk.com/blogs で確認できます。Model Derivative タグを検索します。

Model Derivative プロパティ API の拡張

既存の objectid API (/modelderivative/v2/designdata/:urn/metadata/:modelGuid/properties?objectid=:id) によるプロパティ取得のパフォーマンス向上を計画しています。非常に大きなデータセットを扱う際に役立つように、また、そのデータを取得する際のパフォーマンスを向上させるために、ここでいくつかの重要な作業を行いました。また、モデルの階層やプロパティへのより詳細なアクセスを可能にするために、メタデータ API を改善する予定です。これらの変更はまもなくベータ版の形で提供される予定です。最新情報は forge ブログ（https://forge.autodesk.com/blog）をご覧ください。

参考

https://forge.autodesk.com/ - デベロッパー ポータル

https://forge.autodesk.com/en/support/get-help - Forge help

https://forge.autodesk.com/blog - Forge ブログ

https://forge.autodesk.com/en/docs/model-derivative/v2/developers_guide/overview/ - Model Derivative サービスのドキュメント