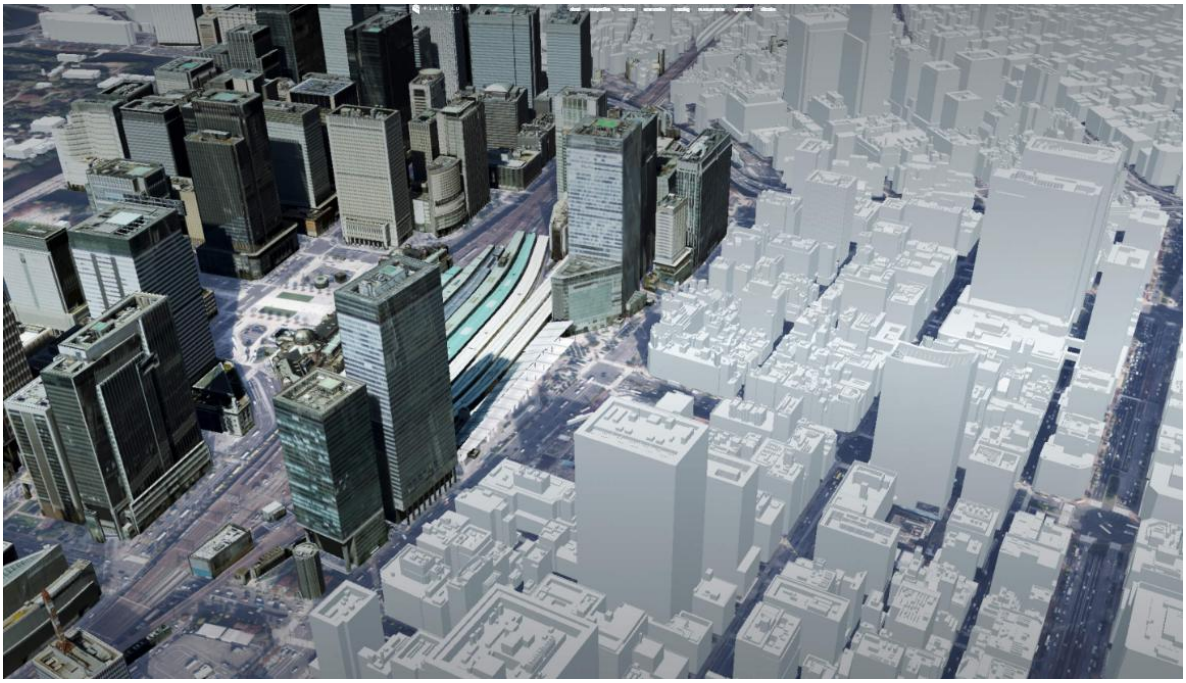
タイトル：

第 3 回 PLATEAU で 3 次元都市モデルに触れてみる



https://www.mlit.go.jp/plateau/learning/

近年、世界各国において 3 次元都市モデルの整備が進められています。現在、国土交通省が主導している「Project PLATEAU」も、そうした 3 次元都市モデル整備プロジェクトの 1 つです。 今回は、この「Project PLATEAU」についてご紹介をしていきます。

Project PLATEAU とは

PLATEAU View: https://plateauview.mlit.go.jp/　(渋谷駅から道元坂を覗いたところ)

Project PLATEAU (プラトー) (https://www.mlit.go.jp/plateau/)は 2020 年度の事業として開始された、国土交通省が主導する 3 次元都市モデルの整備・活用・オープンデータ化のプロジェクトです。(https://www.mlit.go.jp/report/press/toshi03_hh_000074.html)

2020 年度の事業としては、全国 56 都市の 3 次元都市モデルの整備、活用事例 44 件と実証成果を取りまとめた各種マニュアル・技術資料等 10 件が公開されました。

3 次元都市モデルは、フィジカル空間（実世界）の都市を、サイバー空間（仮想的な世界）に再現した 3 次元の都市空間情報プラットフォームで、都市空間そのものを再現するデジタルツインを実現するが可能です。PLATEAU では 3 次元都市モデルを整備することでその利活用を創出しオープンデータとして公開することで、誰もが自由にデータが活用できるようになりました。今後は 3 次元都市モデルを全国展開していく予定です。

セマンティクス・モデル

PLATEAU の 3 次元都市モデルは見た目の形を画像データにしたものではなく、建物や街路、道路、橋梁といった地物に、名称や用途、建設年、行政計画といった都市構造物全般に対して情報を付与することで意味をもたせています。これをセマンティクス・モデル(semantic model)(意味モデル)といいます。このセマンティクスにより、フィジカル空間とサイバー空間の高度の融合が可能となり都市のシミュレーション分析が可能となるのです。

デジタルツイン ≠ メタバース

PLATEAU の 3 次元都市モデルのデータソースは、下記のような地方自治体により定期的に収集・作成されているデータから整備されています。

1．都市計画基本図（基盤地図情報）

2．都市計画基礎調査

3．公共測量成果（航空写真又は LP)の「3 点セット」

これは実際の測量法に基づく公共測量成果として位置づけられている測量や調査から作成されているデータでありフィジカル空間のデータです。

デジタルツインである、PLATEAU の 3 次元都市モデルは原寸(縮尺 1/1)の世界でありメタバースとは違いがあることになります。

- デジタルツイン = シミュレーションの世界
  - 現実空間をコピーした仮想空間を再現すること
  - シミュレーションやフィードバックを可能にする仕組み
- メタバース = コンピューターの世界
  - 必ずしも既存の現実空間を再現する必要はない

- コンピュータの中に構築された 3 次元の仮想空間
- コミュニケーションが行えるサービス・プロダクト

CityGML

PLATEAU では、3 次元都市モデルのデータフォーマットとして、地理空間情報分野における国際標準化団体である OGC(Open Geospatial Consortium)が国際標準として策定した CityGML 2.0（https://www.ogc.org/standards/citygml)を採用しています。

参考：3D 都市モデルと CityGML とは | PLATEAU [プラトー](https://www.mlit.go.jp/plateau/learning/)

CityGML は GML（Geography Markup Language：ISO 19136-1:2020）(https://www.ogc.org/standards/gml)という、OGC によって開発された地理的なデータ構造を XML ベースで表現するマークアップ言語で表記された、高い拡張性を有しており独自の情報を拡張可能なデータフォーマットです。PLATEAU では、都市計画に特化して拡張された ADE(Application Domain Extension)である i-都市再生技術仕様案（i-UR）(https://www.chisou.go.jp/tiiki/toshisaisei/itoshisaisei/iur/index.html)の拡張をし必要な情報を建築物等の属性として追加したり、行政界や区域、統計グリッド等の概念的な地物を追加したりした日本版標準データモデルとして策定しています。

CityGML は確立された国際標準規格であるため、リッチなデータを規則的に格納することが可能ですがデータとしては若干取り扱いが不便なため、3 次元都市空間情報の中間フォーマットと定義しています。これはデータの各利用者が「必要な情報」を「使いやすいフォーマット」に変換した上で利用することを想定しています。ただし、今後は、CityGML データセットの WebAPI、データベース、RDF(Resource Description Framework)等を整備することで、オープンデータとしての利便性をさらに向上させる計画もあるそうです

また、建築・建設系の国際規格である IFC（Industry Foundation Classes: ISO16739-1:2018）(https://www.iso.org/standard/70303.html)や、建物の内部のナビゲーションに使われることを想定したデータモデルや IndoorGML(http://www.indoorgml.net/)などへの拡張も予想されます。

東京 23 区における 3D 都市モデルのための拡張製品仕様によると、データの分類は下記となります。

- bldg 建築物、建築物部分、建築物付属物、及びこれらの境界面
- tran 道路、通路
- luse 土地利用
- urf 都市計画区域、区域区分、地域地区
- dem 地形(起伏)
- fld 洪水浸水想定区域
- tnm 津波浸水想定
- lsld 土砂災害警戒区域
- brid 橋梁
- frn 設置物

参考：3D 都市モデル標準製品仕様書（第 2.3 版）
（https://www.mlit.go.jp/plateaudocument/）

参考：東京 23 区における 3D 都市モデルのための拡張製品仕様

(https://www.geospatial.jp/ckan/dataset/plateau-tokyo23ku-citygml-2020/resource/e5edb01e-8e87-4540-a64e-53dcc0f31f43?inner_span=True)

LOD (Level of Detail)

PLATEAU の CityGML の特徴の 1 つが、詳細さを LOD(Level of Detail）で定義されています。LODはデータの広さ、大きさ、詳細さを 5 つのレベルがあります。

Level	モデルスケール定義	位置 / 高さ 正確度
LOD0	地域、ランドスケープ（地形）	LOD1 以下
LOD1	都市、地域	5m/5m
LOD2	都市、市街地、プロジェクト（建築等）	2m/2m
LOD3	市街地、建築モデル（外観）、ランドマーク	0.5m/0.5m
LOD4	建築モデル（屋内）、ランドマーク	0.2m/0.2m

表 1　LOD の定義及び正確度（案）

3 次元地理空間データ CityGML/IndoorGML に関する国際標準化活動

https://www.jstage.jst.go.jp/article/jjca/52/3/52_3_29/_article/-char/ja/

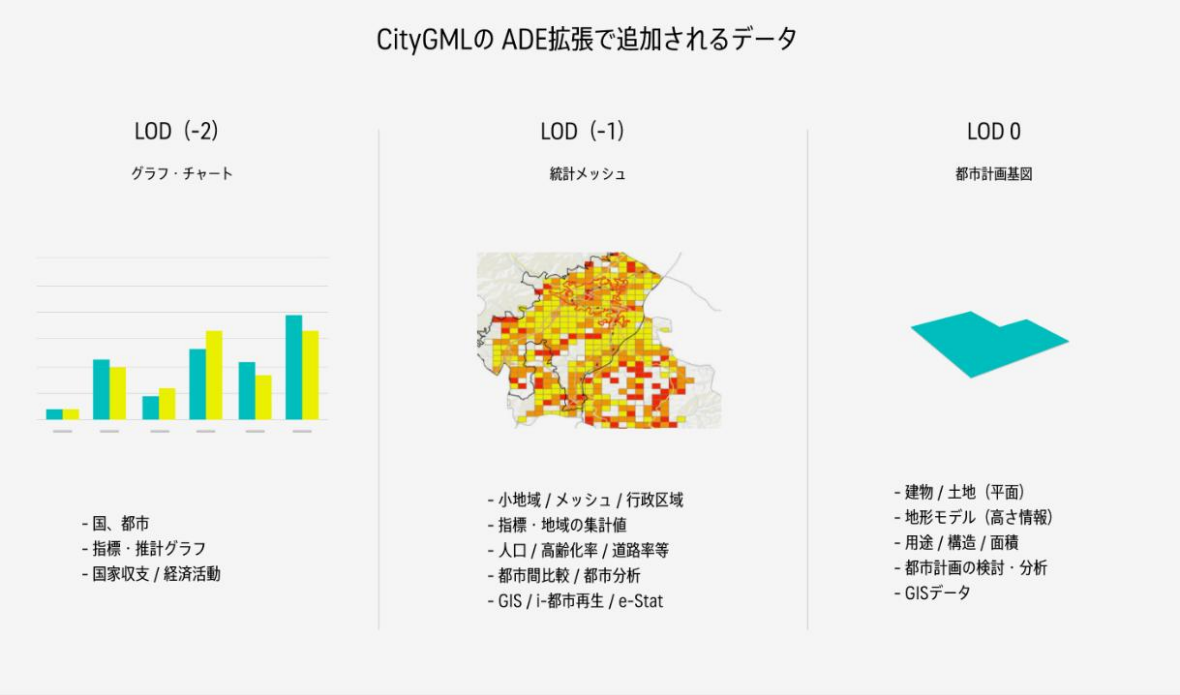
LOD のモデルの概念

- LOD0：地形モデルレベル
- LOD1：四角い箱が立ち上がっているレベル。各箱の高さが何となく分かる
- LOD2：建物の何となくの形、屋根の形が分かるレベル。建物の外見のみ
- LOD3：建物外形ランドマーク。建物の外見のみ
- LOD4: 建物内で歩行空間が認 識できるレベル。 ※建物内も対象とする

建物の LOD の概念

https://www.mlit.go.jp/plateau/learning/

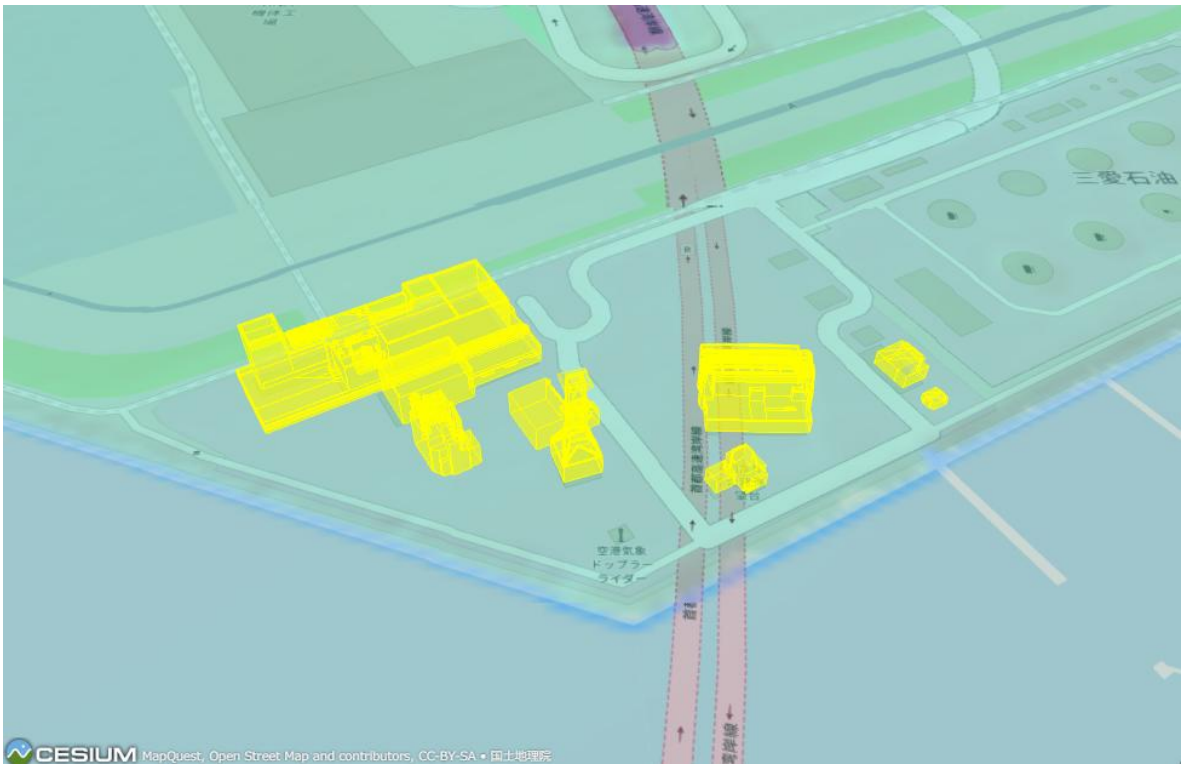
i-UR で拡張されるデータ



https://www.mlit.go.jp/plateau/learning/

PLATEAU のデータ例：東京都大田区羽田空港三丁目の *LOD0* 建物データを筆者が作成し表示

PLATEAU のデータ例：東京都大田区羽田空港三丁目の *LOD* 1 建物データを筆者が作成し表示



PLATEAU のデータ例：東京都大田区羽田空港三丁目の *LOD2* 建物データを筆者が作成し表示

PLATEAU の CityGML の地物と LOD の整備対象は下記となります。

	LOD0	LOD1	LOD2	LOD3
3D 都市モデルの対象範囲	広域	都市域	特定のエリア	限定されたエリア
建築物	○	○	○	○
道路		○	○	○
都市計画決定情報		○		
土地利用		○		
災害リスク		○		
都市設備		○	○	○
植生		○	○	○
地形		○	○	○

表 4-1　標準製品仕様が対象とする地物と LOD

4.1.1　標準製品仕様が対象とする地物と LOD

https://www.mlit.go.jp/plateaudocument/

3D 都市モデルを表示する

PLATEAU の 3 次元都市モデルのデータは、G 空間情報センター

(https://www.geospatial.jp/ckan/dataset/plateau)からダウンロードをします。

https://www.geospatial.jp/ckan/dataset/plateau

対応都市 57 都市(2022-11-05 現在)あるので対象の都市のデータをダウンロードすることが可能です。

対応都市 57(2022-11-05 現在)

東京都 23 区	東京都東村山市	静岡県掛川市	広島県福山市
北海道札幌市	神奈川県横浜市	静岡県菊川市	愛媛県松山市
福島県郡山市	神奈川県川崎市	愛知県名古屋市	福岡県北九州市
福島県いわき市	神奈川県相模原市	愛知県岡崎市	福岡県久留米市
福島県白河市	神奈川県横須賀市	愛知県津島市	福岡県飯塚市
茨城県鉾田市	神奈川県箱根町	愛知県安城市	福岡県宗像市
栃木県宇都宮市	新潟県新潟市	大阪府大阪市	熊本県熊本市

群馬県桐生市	石川県金沢市	大阪府豊中市	熊本県荒尾市
群馬県館林市	石川県加賀市	大阪府池田市	熊本県玉名市
埼玉県さいたま市	長野県松本市	大阪府高槻市	熊本県益城町
埼玉県熊谷市	長野県岡谷市	大阪府摂津市	大分県日田市
埼玉県新座市	長野県伊那市	大阪府忠岡町	沖縄県那覇市
埼玉県毛呂山町	長野県茅野市	兵庫県加古川市	
千葉県柏市	岐阜県岐阜市	鳥取県鳥取市	
東京都八王子市南大沢	静岡県沼津市	広島県呉市	

テスト用の CityGML ファイル

3D 都市モデル（Project PLATEAU）東京都 23 区（CityGML 2020 年度）

（**https://www.geospatial.jp/ckan/dataset/plateau-tokyo23ku-citygml-2020**）

テストで使用する CityGML ファイルは、533926(東京都大田区羽田空港)に入っているのでこれをダウンロードして、解凍して取り出します。

- ダウンロード先：(https://www.geospatial.jp/ckan/dataset/plateau-tokyo23ku-citygml-2020/resource/bd65212e-0467-4c6c-a28c-946192acbdfc)
- ダウンロードファイル：533926_2.7z (圧縮形式：7-Zip 形式)
- CityGML ファイル：53392633_bldg_6697_op2.gml

圧縮ファイルは、7-Zip 形式(https://sevenzip.osdn.jp/)という形式で圧縮されています。解凍ツールをインストールして圧縮ファイルを解凍してください。

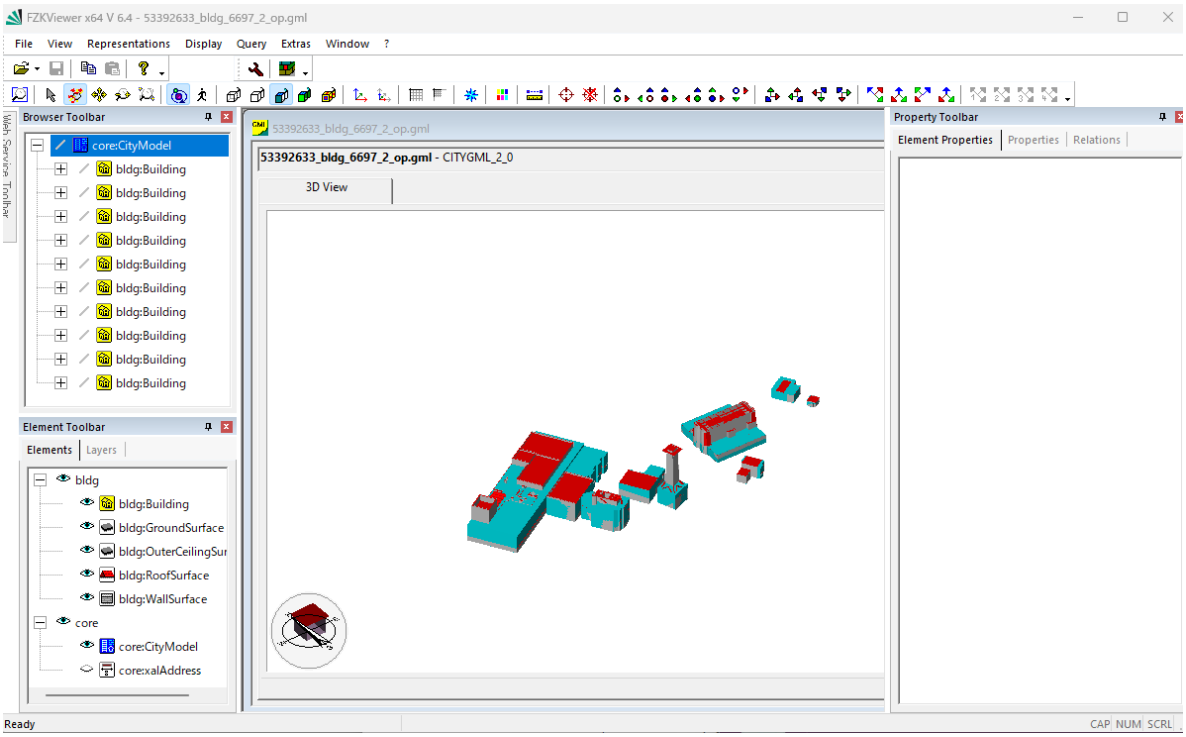
CityGML ファイルは、下記の順番で取り出してください。

533926_2.7z –(解凍) –> bldg.7z -(解凍) –> bldg/ 53392633_bldg_6697_op2.gml

CityGML ファイルを表示させる下記のツールがあります。

- Windows 版
  - FZKViewer(https://www.iai.kit.edu/1302.php)
  - FZKViewer のインストール（Windows 上）(https://www.kkaneko.jp/tools/win/fzkviewer.html)
- Mac 版
  - Azul (https://github.com/tudelft3d/azul)

Windows ユーザであれば、FZKViewer で CityGML ファイルの構造や地物を視覚的に確認することが可能です。



FZKViewer で CityGML のファイルを表示

CityGML ファイルを可視化

CityGML ファイルを Google Colaboratory で可視化します。

CityGML ファイル名から地理空間情報を取得します

ファイル名称は下記のルールとなります

[地域メッシュコード]_[地物型]_[CRS] _[オプション].gml

- 地域メッシュコード: ファイル単位となる地域メッシュのメッシュコード
- 地物型: 格納された地物の種類を示す接頭辞 (bldg 等)
- CRS: 格納された地物に適用される座標参照系 (6697 等)
- オプション: 必要に応じてファイルを細分したい場合の識別子（オプション）

参考

- 地域メッシュコード： https://ja.wikipedia.org/wiki/地域メッシュ
- 座標系・測地系：はじめての地理空間情報:
  https://hackmd.io/5S_fL0GsT2OTkIi0owIccw
- 3D 都市モデル標準作業手順書（第 2.2 版）:
  https://www.mlit.go.jp/plateau/file/libraries/doc/plateau_doc_0002_ver02.pdf

```
# CityGML(XML)ファイルを準備
import os

# CityGML(XML)ファイル
filename = "/content/drive/MyDrive/dataset/citygml_in/53392633_bldg_66
97_2_op.gml"
# CityGML(XML)ファイルのファイル名のみを取得
basename = os.path.basename(filename)

# ファイル名を分割
basenames = basename.split('_')
# ファイル名からメッシュコードを取得
mesh_code = basenames[0]を取得
# ファイル名から地物型 (bldg)
object_name = basenames[1]
# ファイル名から CRS 空間参照 ID (SRID) (座標情報を取得)
from_srid = basenames[2]

# 出力するファイルの座標系を選択
```

```
# 座標系に関しては下記を参考にしてください
# はじめての地理空間情報: https://hackmd.io/5S_fL0GsT2OTkIi0owIccw

# 東京都大田区羽田空港のデータなので、東京都の座標・測地系で出力する
# 日本測地系 2011　平面直角座標系: 東京都の一部、福島県、栃木県、茨城県、埼玉県、千葉県、群馬県、神祭川県
to_srid = "6677"
```

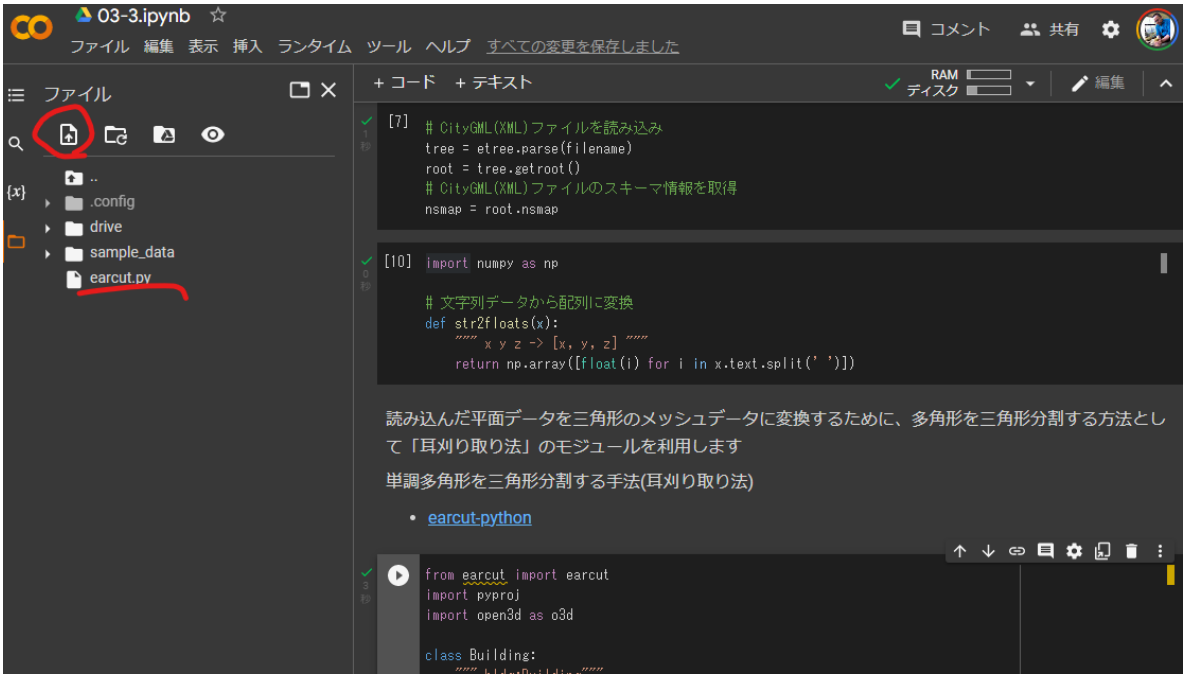
クラス Building を定義して、CityGML の平面データを三角形のメッシュデータに変換します。

平面データを三角形のメッシュデータに変換するために、多角形を三角形分割する方法として「耳刈り取り法」のモジュールを利用します

単調多角形を三角形分割する手法(耳刈り取り法)

* earcut-python (https://github.com/joshuaskelly/earcut-python)

「セッションストレージにアップロード」を選択して「earcut.py」をアップロードします



建物データを保持する為にクラス Building を定義します

```
from earcut import earcut
import pyproj
import open3d as o3d

class Building:
    """ bldg:Building"""

    def __init__(self, from_srid, to_srid):
        # 元の平面データ
        self.polygons = []

        # 平面データの変換した座標系のデータ
        self.vertices = []
        # 三角形の頂点情報
        self.triangles = []
        # 三角形のメッシュデータ
        self.triangle_meshes = []

        # 座標変換の定義をします: 変換元座標系 → 変換先座標系
        self.transformer = pyproj.Transformer.from_crs(f"epsg:{from_sr
id}", f"epsg:{to_srid}")

    def get_triangle_mesh(self):
        # 三角形のメッシュデータを返す
        return self.triangle_meshes

    def transform_coordinate(self, latitude, longitude, height):
        # 座標変換: 変換元座標系 → 変換先座標系
        xx, yy, zz = self.transformer.transform(latitude, longitude, h
eight)
        return np.array([xx, yy, zz])

    def create_triangle_meshes(self, polygons):
        # 三角形のメッシュデータ作成
        for plist in polygons:
            # 平面データ座標変換
            vertices = [self.transform_coordinate(*x) for x in plist]
            # 単調多角形を三角形分割(耳刈り取り法)
```

```
            vertices_earcut = earcut(np.array(vertices).flatten(), dim
=3)
            if len(vertices_earcut) > 0:
                # 三角形の頂点情報を設定
                # 三角形の頂点情報は、平面データ座標のインデックス番号
                vertices_length = len(self.vertices)
                self.vertices.extend(vertices)
                triangles = np.array(vertices_earcut).reshape((-1, 3))
                triangles_offset = triangles + vertices_length
                self.triangles.extend(triangles_offset)

        # 三角形のメッシュデータ作成
        triangle_meshes = o3d.geometry.TriangleMesh()
        triangle_meshes.vertices = o3d.utility.Vector3dVector(self.ver
tices)
        triangle_meshes.triangles = o3d.utility.Vector3iVector(self.tr
iangles)

        # 三角形のメッシュの法線取得
        triangle_meshes.compute_vertex_normals()

        # データを保存
        self.triangle_meshes = triangle_meshes
        self.polygons = polygons
```

CityGML ファイルから LOD2 の平面データを取得し、建物別に三角形のメッシュデータを作成します

```
# LOD2 のデータを取得
def lod2(tree, nmap, from_srid, to_srid):
  """ lod2 """
  obj_buildings = []

  # bldg:Building 建物データを検索
  buildings = tree.xpath('/core:CityModel/core:cityObjectMember/bldg:B
uilding', namespaces=nsmap)

  # 建物データでループ
  for building in buildings:
    # 建物クラスを作成
```

```
    obj_building = Building(from_srid, to_srid)

    # bldg:GroundSurface, bldg:RoofSurface, bldg:RoofSurface
    # LOD2 の屋根、壁、床の面データのパスを定義
    polygon_xpaths = ['bldg:boundedBy/bldg:GroundSurface/bldg:lod2Mult
iSurface/gml:MultiSurface/gml:surfaceMember/gml:Polygon',
                      'bldg:boundedBy/bldg:RoofSurface/bldg:lod2MultiS
urface/gml:MultiSurface/gml:surfaceMember/gml:Polygon',
                      'bldg:boundedBy/bldg:WallSurface/bldg:lod2MultiS
urface/gml:MultiSurface/gml:surfaceMember/gml:Polygon']

    # LOD2 の屋根、壁、床の面データの位置情報を取得
    vals_list = []
    for polygon_xpath in polygon_xpaths:
      poslist_xpaths = building.xpath(polygon_xpath, namespaces=nsmap)
      for poslist_xpath in poslist_xpaths:
        vals = poslist_xpath.xpath('gml:exterior/gml:LinearRing/gml:po
sList', namespaces=nsmap)
        vals_list.extend(vals)

    # [X, Y, Z]の座標配列を作成
    polygons = [str2floats(v).reshape((-1, 3)) for v in vals_list]
    # 三角形のメッシュデータ作成
    obj_building.create_triangle_meshes(polygons)
    # 建物クラスデータを保存
    obj_buildings.append(obj_building)

  return obj_buildings
```

建物別に三角形のメッシュデータを PLY ファイル形式で保存をします

```
# PLY ファイルに保存
def write_ply(output_filename, obj_buildings):
  # 保存用ファイル名を作成
  basedir = os.path.dirname(output_filename)
  basename_without_ext = os.path.splitext(os.path.basename(output_file
name))
  basename = basename_without_ext[0]
  extname = basename_without_ext[1]

  # 建物クラスでループ
```

```
  for index, obj_building in enumerate(obj_buildings):
    # 三角形のメッシュデータを取得
    triangle_mesh = obj_building.get_triangle_mesh()
    # PLY ファイルに保存
    pathname = os.path.join(basedir, f"{basename}_{index:02}{extname}"
)
    o3d.io.write_triangle_mesh(pathname, triangle_mesh, write_ascii=Tr
ue)
```

保存した PLY ファイルは、Open3d と Plotly で 3 次元表示をします。（前回のコードを改造して利用します）

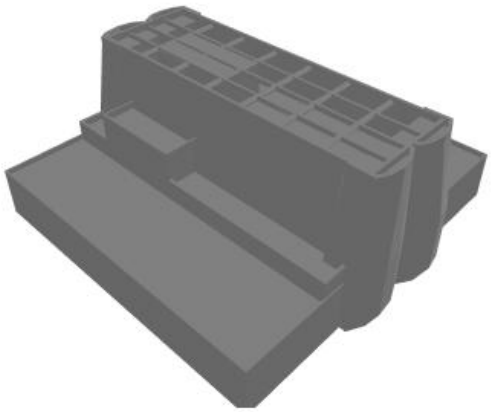
```
 Plotly をインポート
import plotly.graph_objects as graph_objects

def show_ply(show_filename):
  # メッシュタイプ（PLY ファイル）のデータを読み込み
  mesh = o3d.io.read_triangle_mesh(show_filename)

  # 法線を計算
  mesh.compute_vertex_normals()     # 頂点の法線を計算
  mesh.compute_triangle_normals()   # 三角形の法線を計算

  # 頂点と三角形のデータを np.array 形式に変換
  triangles = np.asarray(mesh.triangles)
  vertices = np.asarray(mesh.vertices)

  # Plotly で表示。出力結果はマウスでインタラクティブに動かすことが可能です
  fig = graph_objects.Figure(
      data=[
          graph_objects.Mesh3d(
              x=vertices[:,0],    # 頂点の X 座標
              y=vertices[:,1],    # 頂点の Y 座標
              z=vertices[:,2],    # 頂点の Z 座標
              i=triangles[:,0],   # 三角形の 1 番目の座標
              j=triangles[:,1],   # 三角形の 2 番目の座標
              k=triangles[:,2],   # 三角形の 3 番目の座標
              facecolor= (0.5, 0.5, 0.5),   # 面の色を設定
              opacity=1.0)       # 表面の不透明度を設定
      ],
```

```
        layout=dict(
            scene=dict(
                xaxis=dict(visible=False), # X 軸非表示
                yaxis=dict(visible=False), # Y 軸非表示
                zaxis=dict(visible=False)  # Z 軸非表示
            )
        )
    )
    fig.show()  # 表示
```

No.1 の建物を表示

```
# PLY ファイル表示(LOD2 - No.1 の建物)
show_ply("/content/drive/MyDrive/dataset/citygml_out/53392633_bldg_6677_lod2_01.ply")
```

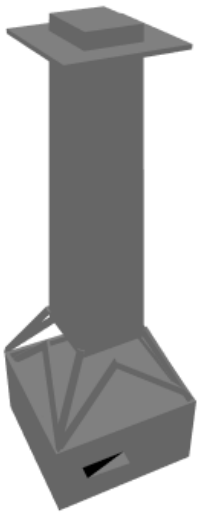
No.1 の建物の表示例



No.5 の建物を表示

```
# PLY ファイル表示(LOD2 - No.5 の建物)
show_ply("/content/drive/MyDrive/dataset/citygml_out/53392633_bldg_6677_lod2_05.ply")
```

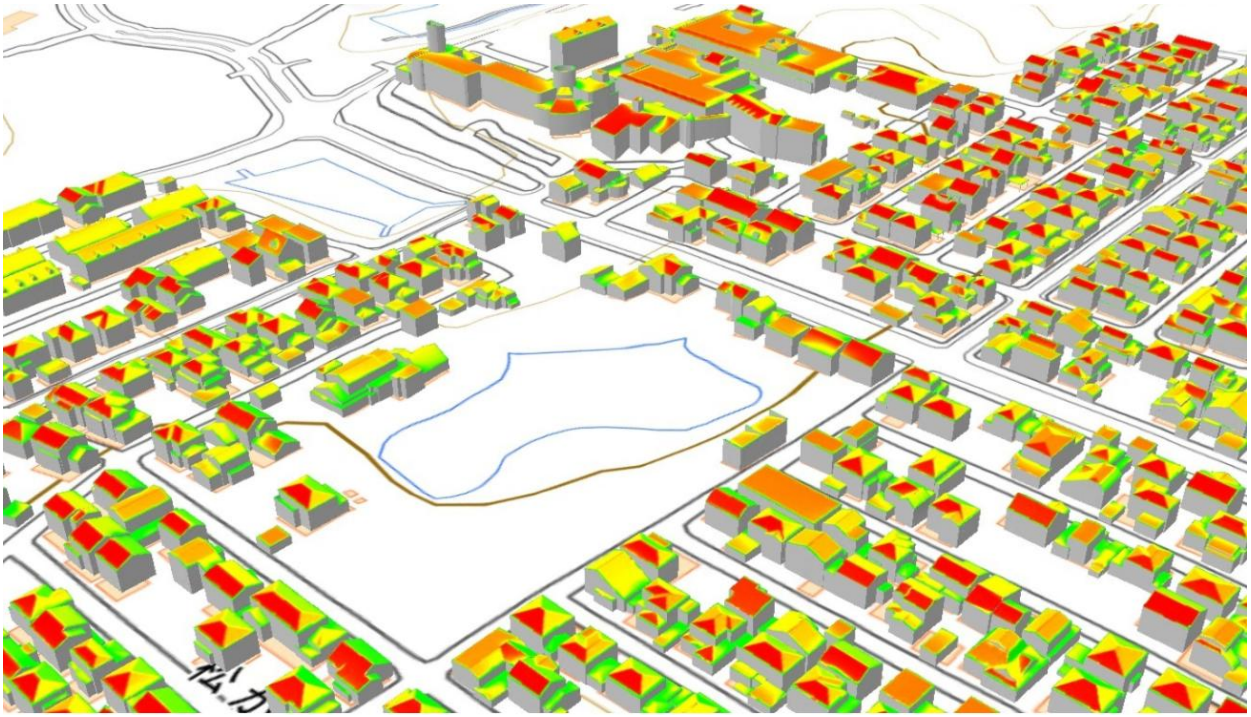
No.5 の建物の表示例



参考コード

- 技術的特異点 - 「open3 次元」一覧 (http://tecsingularity.com/category/open3 次元/)
- Open3d Visualize in Google Colab(https://colab.research.google.com/drive/1CR_HDvJ2AnjJV3Bf5vwP70K0hx3RcdMb?usp=sharing)
- earcut-python (https://github.com/joshuaskelly/earcut-python)
- AcculusSasao/plateaupy (https://github.com/AcculusSasao/plateaupy)
- ksasao/PlateauCityGmlSharp (https://github.com/ksasao/PlateauCityGmlSharp/)

事例：太陽光発電のシミュレーションに使ってるよ



UC_ID_3-006: 太陽光発電のポテンシャル推計及び反射シミュレーション

https://www.mlit.go.jp/plateau/use-case/smart-planning/3-006/

近年、世界的に地球温暖化対策が喫緊の課題とされており、温室効果ガスの排出を全体としてゼロにするカーボンニュートラル(脱炭素社会)の実現を目指し、脱炭素を進める動きが加速している。3 次元都市モデルの LOD2 が持つ建物の屋根面積、傾き、形状と、日陰影響の情報などを活用することで、建物の屋根などに設置した太陽光パネルの発電量を精緻にシミュレーションすることが可能になる。

参考：3D 都市モデルを活用した太陽光発電施設の設置シミュレーション 技術検証レポート

https://www.mlit.go.jp/plateau/file/libraries/doc/plateau_tech_doc_0001_ver01.pdf