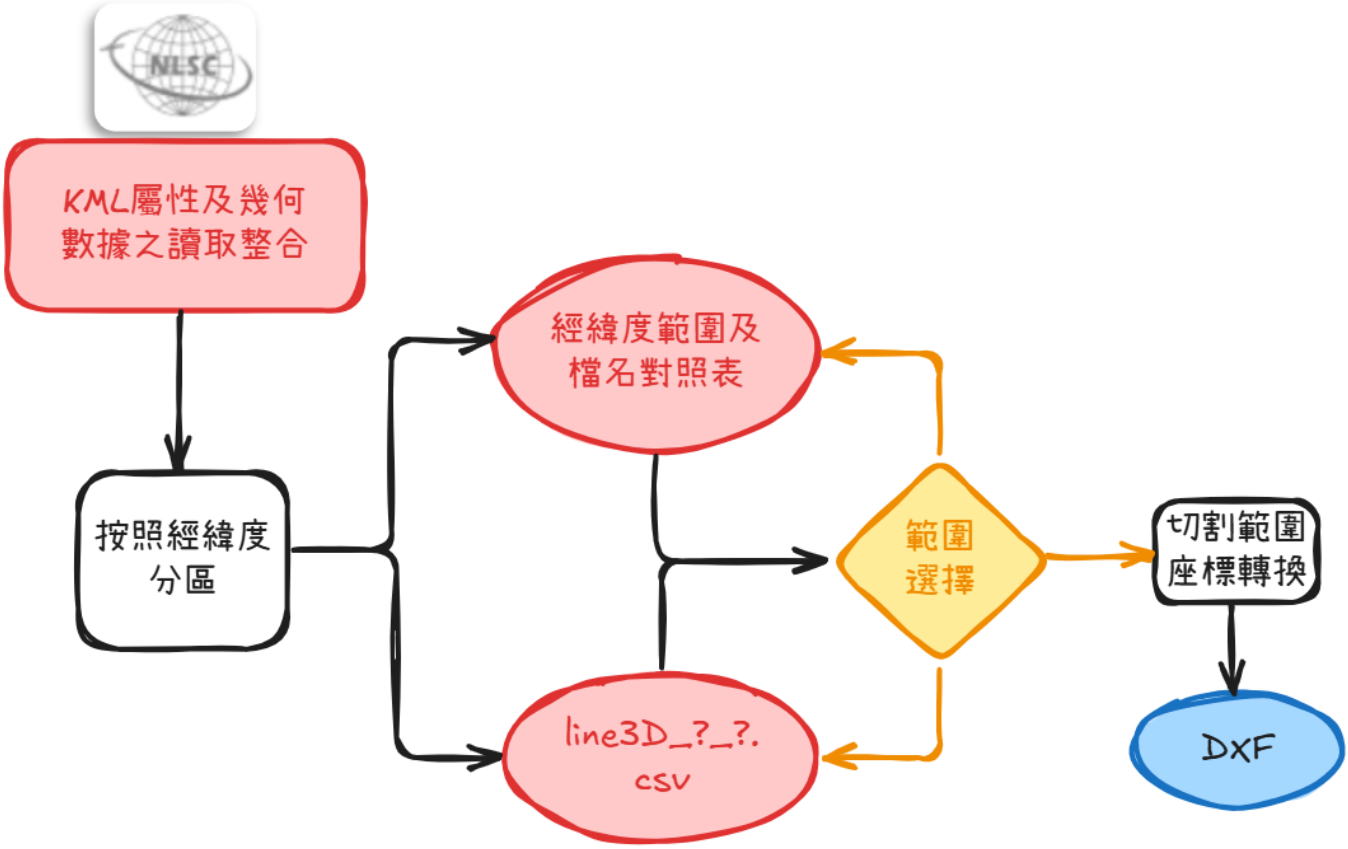
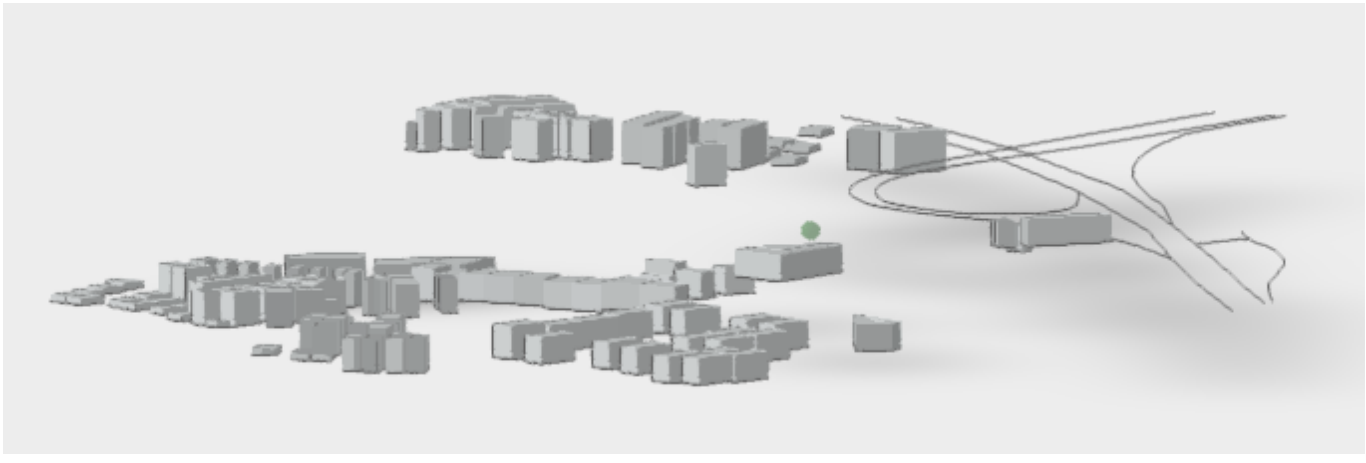
道路座標的整理與轉換

- 整體工作流程如圖所示，區分為2大區塊：讀取整合分區，以及座標轉換及DXF之轉檔。



Reference



內政部高架道路3維線段之讀取及處理

背景

- 內政部檔案格式是KML。

- 雖然geopandas可以直接讀取KML，但是只能擷取第一層，其他層會略過，最後只能放棄，改用`xml.etree`來讀取，還是比較實際有效。

主程式

- 主程式管理目錄下所有`*LINE*.kml`，執行`read_kml_to_gdf`副程式。
- 屬性資料儲存後予以刪除，以備整合成為一個大檔`'line3D.csv'`。
- 雖然這個檔案讀取的時間還在可接受的範圍，但畢竟單一很大（~1GB）的檔案有壞掉的風險，還是有必要發展更要效率、更安全的服務方案（）。

```
import geopandas as gpd
from shapely.wkt import loads
import xml.etree.ElementTree as ET
import os
import pandas as pd
from shapely.geometry import LineString

# Example usage
kml_dir ='.'
gdf_list = []

# Loop through all the KML files in the directory
for filename in os.listdir(kml_dir):
    if filename.endswith('.kml') and 'LINE' in filename:
      gdf = read_kml_to_gdf(filename)
      if len(gdf)==0:continue
      root=filename.replace('kml','')
      gdf.to_csv(root+'csv',index=False)
      for c in gdf.columns:
        if c!='geometry':del gdf[c]
      gdf_list.append(gdf)
if len(gdf_list)==0:sys.exit('fail')
all_gdf = pd.concat(gdf_list, ignore_index=True)
all_gdf.to_csv('line3D.csv', index=False)
```

副程式

- `read_kml_to_gdf`的任務就是將kml檔案的3維線段的座標值取出來，存成GeoPandas的數據表型態
- 其他屬性資料包括行政區名稱等，似乎也沒有特別的需求，並不會用到，此處還是將其讀取另存。

```
def read_kml_to_gdf(kml_file):
    tree = ET.parse(kml_file)
    root = tree.getroot()
    ns = {'kml': 'http://www.opengis.net/kml/2.2'}

    features = []
    for placemark in root.findall('.//kml:Placemark', ns):
        name = placemark.find('kml:name', ns).text
        geom_type = placemark.find('kml:LineString', ns) is not None
```

```
        if geom_type:
            coordinates = placemark.find('kml:LineString/kml:coordinates',
ns).text.strip()
            coords = [tuple(map(float, coord.split(','))) for coord in
coordinates.split()]
            geometry = LineString(coords)

            # Extract the extended data
            extended_data = placemark.find('kml:ExtendedData', ns)
            if extended_data is not None:
                schema_data = extended_data.find('kml:SchemaData', ns)
                if schema_data is not None:
                    attributes = {
                        elem.attrib['name']: elem.text
                        for elem in schema_data.findall('kml:SimpleData', ns)
                    }
            else:
                attributes = {}

            feature = {
                'name': name,
                'geometry': geometry,
                **attributes
            }
            features.append(feature)
        else:
            # Handle other geometry types if needed
            continue

    gdf = gpd.GeoDataFrame(features, geometry='geometry')
    return gdf
```

建築物與道路整併輸出（bld_line2dxf.py）

背景

- 高架道路在噪音模擬作業中是很重要的噪音源，但是其幾何元件相對建築物來說還算單純，似乎沒有必要另外建立一個計算流程，附加在建築物的切割與輸出，會比較單純一些。
- 未來如果要再加入地區（平面）道路，元件內容項目可能會很多，建議再另外創建新的計算流程。

程式碼說明

- 呼叫`line2dxf.py`副程式，以得到高架道路的三維GeoPandaDataFrame，詳見line2dxf.py說明。
- 內政部資料並沒有路寬，只有中心線的三維線段LineString/MuitiLineString,
- 輸出dxf時不必加地標的高程，但必須是`add_polyline3d`。
- MuitiLineString狀況：必須針對`polylin.geoms`進行迴圈、歷遍所有的線段。

```
#原來bld2dxf.py內容
...
  sliced_gdf=create_line_segments(swLL,neLL)

  ii=0
  for i in sliced_gdf.index:
    layer_name = f"Polylin_{ii}"
    layer = doc.layers.add(layer_name)
    polylin=sliced_gdf.loc[i,'geometry_twd97']
    if polylin.geom_type=="LineString":
      pnts = [Vec3(p) for p in polylin.coords]
      msp.add_polyline3d(pnts, dxfattribs={ "layer": layer_name})
    elif polylin.geom_type=="MultiLineString":
      for lin in polylin.geoms:
        pnts = [Vec3(p) for p in lin.coords]
        msp.add_polyline3d(pnts, dxfattribs={ "layer": layer_name})
    ii+=1
    ...
# 輸出成檔案
```

按照經緯度切割高架道路

背景

- 內政部檔案除了高速公路之外，其他快速道路或重要路段，是採行按照縣市進行分類。共計8個檔案。但實務上有可能會在縣市邊界上，還需要讀取鄰近縣市的檔案，過程常常會發生不流暢的作業方式。
- 讀取OSM檔案時，我們用過按照經緯度執行`osmconvert`來轉換OSM與KML檔案。解析度用0.5/0.1度2個層次。因為使用第三方程式工具，需要在`bash`上執行。
- 此處要處理的是`csv`檔案的切割，還是用python還是比較方便。

相關程式說明

按經緯度切割

```
import geopandas as gpd
import pandas as pd
import os
from shapely.wkt import loads
from shapely.geometry import Polygon


# Load the 3D LineString data from the CSV file
gdf = pd.read_csv('line3D.csv')
gdf['geometry'] = gdf['geometry'].apply(loads)
gdf = gpd.GeoDataFrame(gdf, geometry='geometry', crs="EPSG:3857")

# Set the starting coordinates and the resolution
```

```
start_lon = 119.2
start_lat = 21.5
resolution = 0.5
overlap = 0.1

# Calculate the grid boundaries
min_lon, min_lat, max_lon, max_lat = gdf.total_bounds


# Create the output directory if it doesn't exist
output_dir = 'splits'
os.makedirs(output_dir, exist_ok=True)
grid_info = []


# Iterate through the grid and split the data
for lon in range(int((max_lon - min_lon) / resolution) + 1):
    for lat in range(int((max_lat - min_lat) / resolution) + 1):
        x1 = min_lon + lon * resolution - overlap
        x2 = min_lon + (lon + 1) * resolution + overlap
        y1 = min_lat + lat * resolution - overlap
        y2 = min_lat + (lat + 1) * resolution + overlap

        # Filter the data based on the grid boundaries
        dd={'geometry': [Polygon([(x1, y1), (x1, y2), (x2, y2), (x2, y1), (x1,
y1)])]}
        bounds_df=pd.DataFrame(dd)
        bounds_gdf = gpd.GeoDataFrame(bounds_df, geometry=bounds_df['geometry'],
crs="EPSG:3857")
        grid_gdf = gpd.overlay(gdf, bounds_gdf, how='intersection')

        # Save the filtered data to a new file
        output_file = os.path.join(output_dir, f'grid_{lon}_{lat}.csv')
        grid_gdf.to_csv(output_file,index=False)
        print(f'Saved file: {output_file}')

        grid_info.append({
            'file_name': f'grid_{lon}_{lat}.csv',
            'min_lon': x1,
            'min_lat': y1,
            'max_lon': x2,
            'max_lat': y2
        })

# Save the grid information to a CSV file
grid_info_df = pd.DataFrame(grid_info)
grid_info_df.to_csv('grid_info.csv', index=False)
```

切割結果之調用

- line2dxf.py中的open_grid_files副程式
- 會需要split1.py之中另存的範圍座標值與檔案名稱的綁定數據組。

```
def open_grid_files(bounds_gdf):
    """
    Open the grid files that intersect with the given bounding box, and return the
one with the smallest length.

    Args:
    bounds_gdf (GeoDataFrame): The GeoDataFrame containing the bounding box.

    Returns:
    GeoDataFrame or None: The GeoDataFrame with the smallest length data, or None
if no matching files are found.
    """
    # Load the grid information from the CSV file
    root_dir = '/nas2/kuang/MyPrograms/CADNA-A/112roads'
    file_name = f"{root_dir}/grid_info.csv"
    grid_info_df = pd.read_csv(file_name)
    grid_info_df['geometry'] = [f"POLYGON(({i} {j},{k} {j},{k} {l}, {i} {l}, {i}
{j}))" for i, j, k, l in zip(grid_info_df.min_lon, grid_info_df.min_lat,
grid_info_df.max_lon, grid_info_df.max_lat)]
    grid_info_gdf = gpd.GeoDataFrame(grid_info_df,
geometry=grid_info_df['geometry'].apply(loads), crs="EPSG:3857")

    # Find the grid files that are fully contained within the given bounding box
    covering_grids =
grid_info_gdf[bounds_gdf.geometry[0].within(grid_info_gdf.geometry)]

    # Open the corresponding grid files
    grid_gdfs = []
    for _, row in covering_grids.iterrows():
        file_name = row['file_name']
        file_path = os.path.join(f"{root_dir}/splits", file_name)
        grid_gdfs.append(pd.read_csv(file_path))

    if grid_gdfs:
        return min(grid_gdfs, key=lambda x: len(x))
    else:
        sys.exit('wrong bounds_gdf:', bounds_gdf)
```