End-to-End Analysis of Large 3D Geospatial Datasets in RAPIDS

John Murray

Fusion Data Science/University of Liverpool

@MurrayData

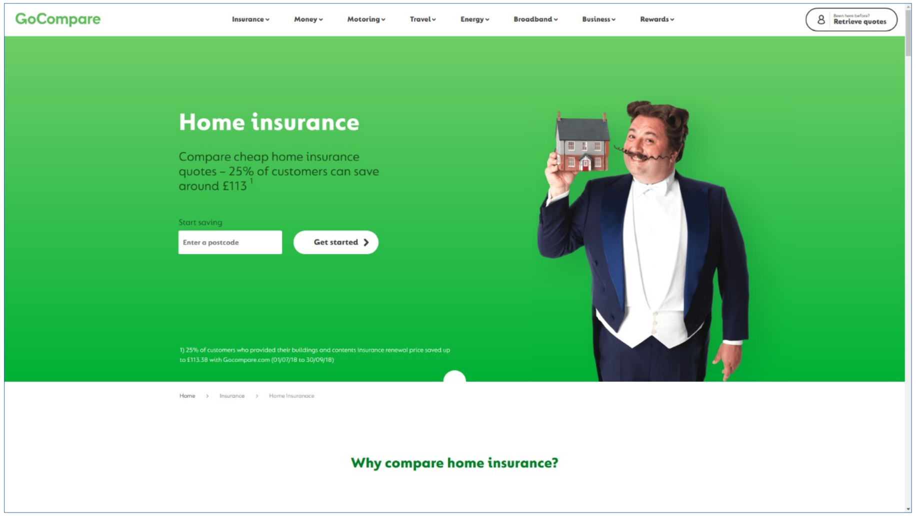
What do we mean by ‘End-to-End’ Analysis?

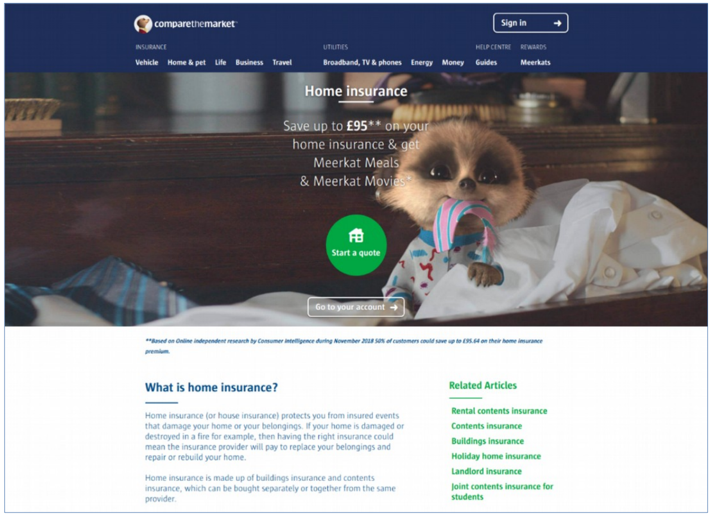
- Processing of raw data sources & ETL:
  - Data calibration
  - Conversion and standardisation
  - Load to repositories
- Combining data sources for:
  - Augmentation to enhance data
  - Attach labels to training data
- Train models
- Infer models
- Interpret and deploy results of inference
- Potentially multistage process

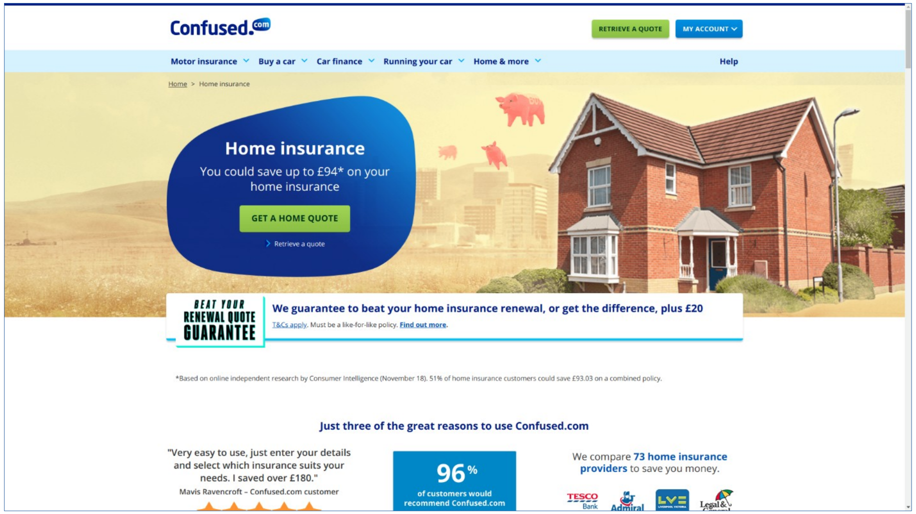
“AI is a disruptive technology”

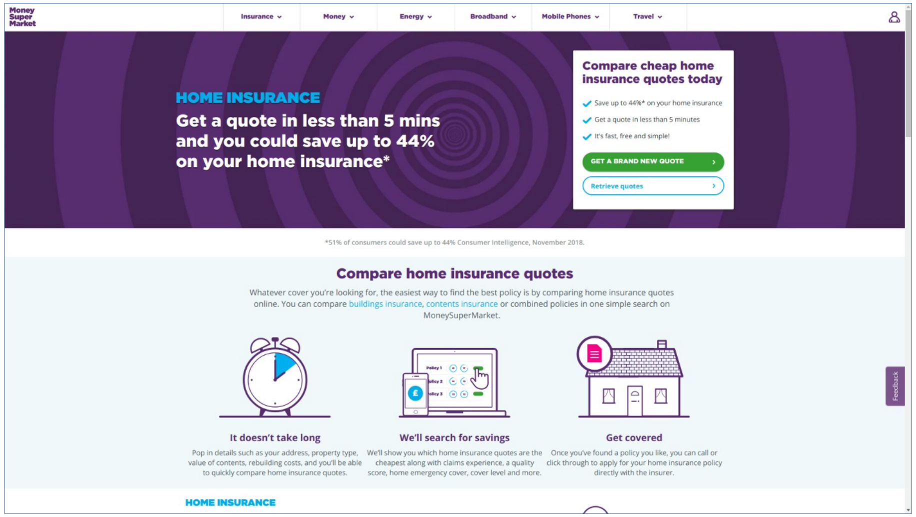
AI is also a response to other disruptive technology.

Disruption of Property Insurance Market by Price Comparison websites









The Challenge for Insurers

- Customers will not complete lengthy application forms online
- Difficult for insurer to ask customer for further information
- Customers expect instant quotations
- Potential financial loss from underwriting high risk propoerties
- Potential loss of low risk customer to a competitor
- Traditional underwriting methods no longer work

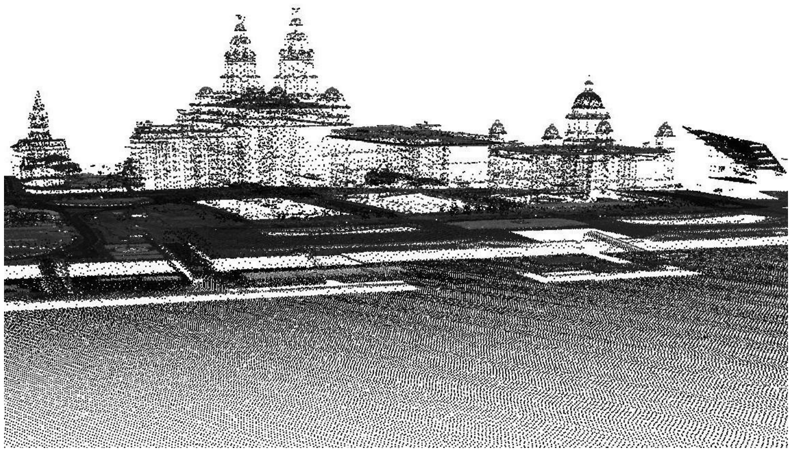
The Response

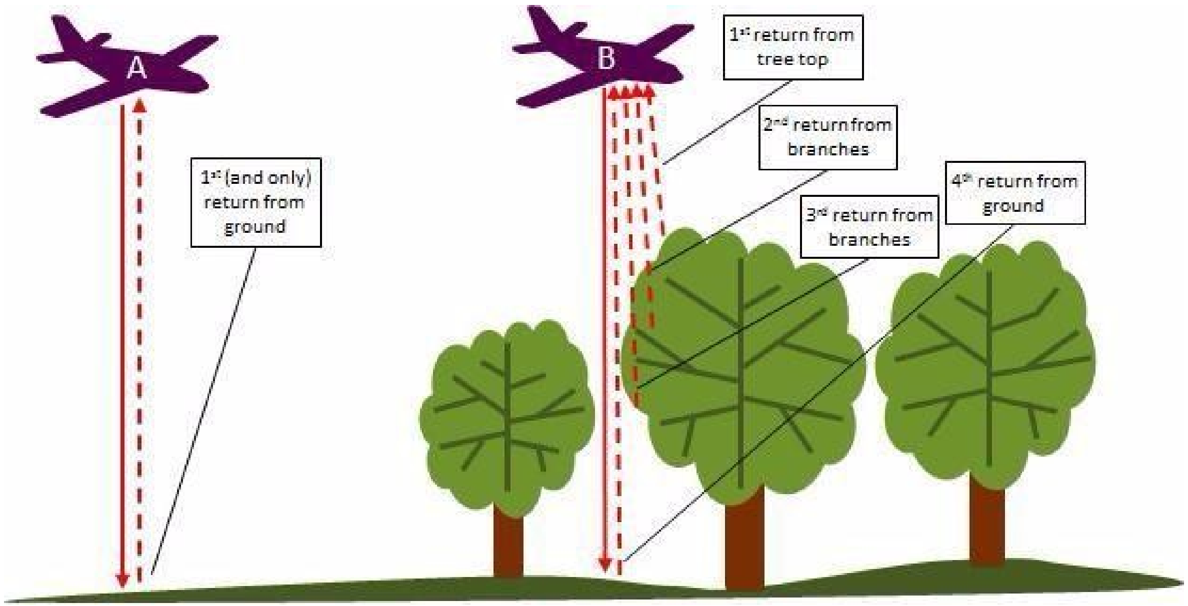
- Use AI to classify property attributes and detect risks
- Use alternative data sources to assess underwriting risk
- Minimise application form by pre-populating answers
- Move away from area based risk analysis to individual properties
- Take a 3-dimensional view of a property and its immediate environment

Geospatial Data – Data Sources

- Traditional data sources
  - Census and National Statistics
  - Mapping data as vectors and rasters
- New Data Sources
  - Satellite & Aerial Imagery
  - LiDAR data from aircraft and vehicles
  - Sensor data, e.g. SAR
  - Social Media
  - Cellphone Apps
  - Government Open Data
  - Crime location data
  - Field surveying

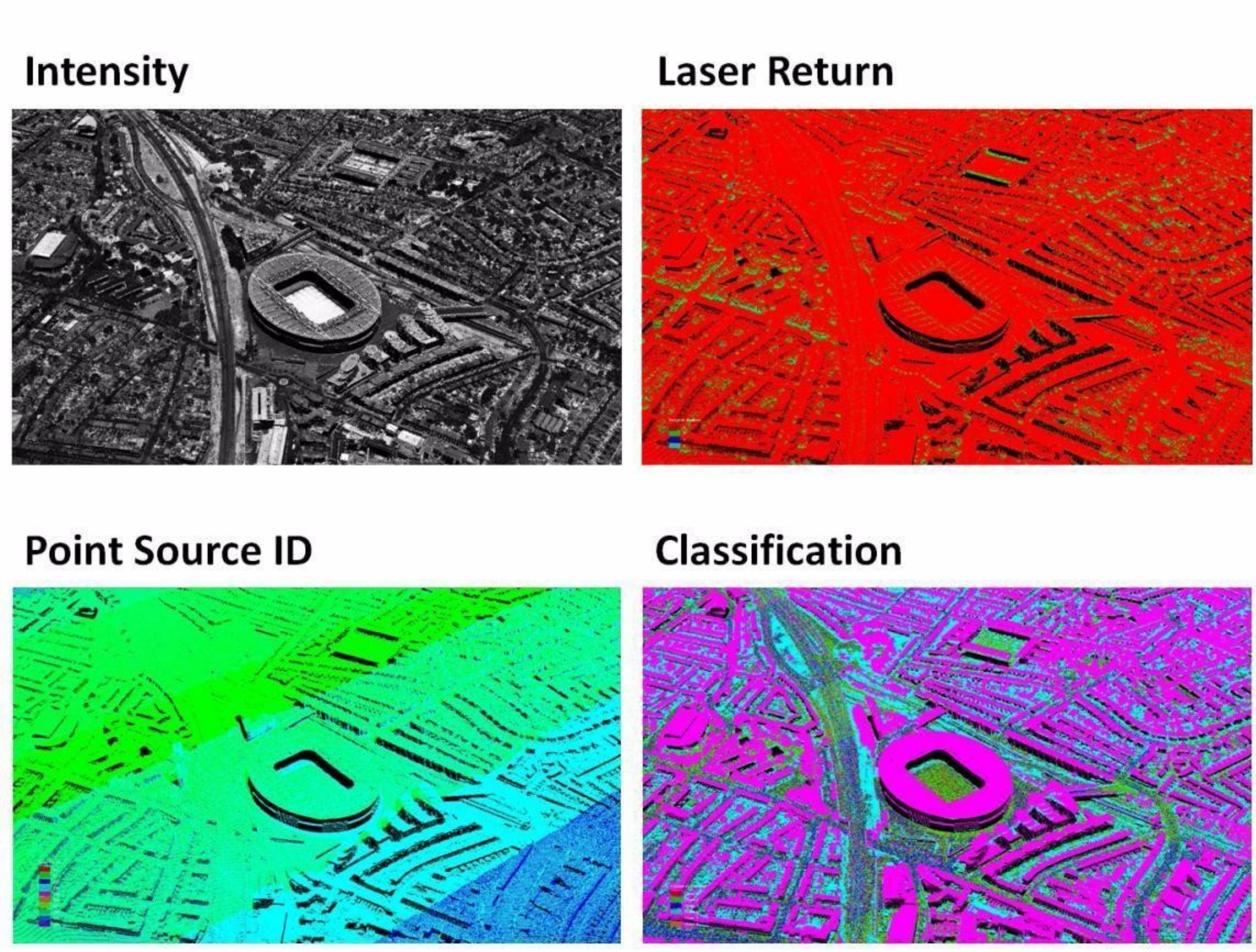
LiDAR Data





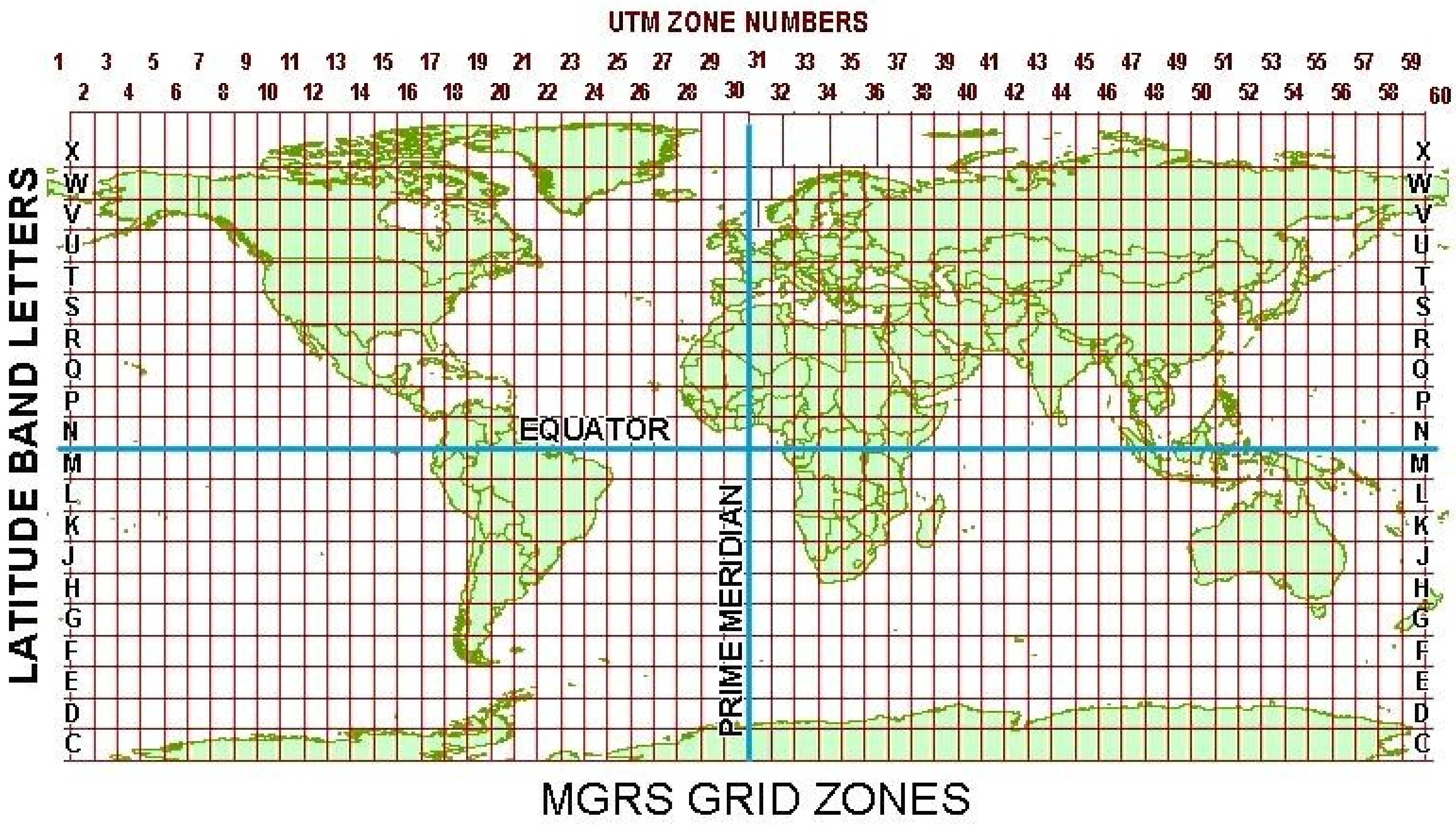
Open Data Images UK Environment Agency

LiDAR Point Cloud Data



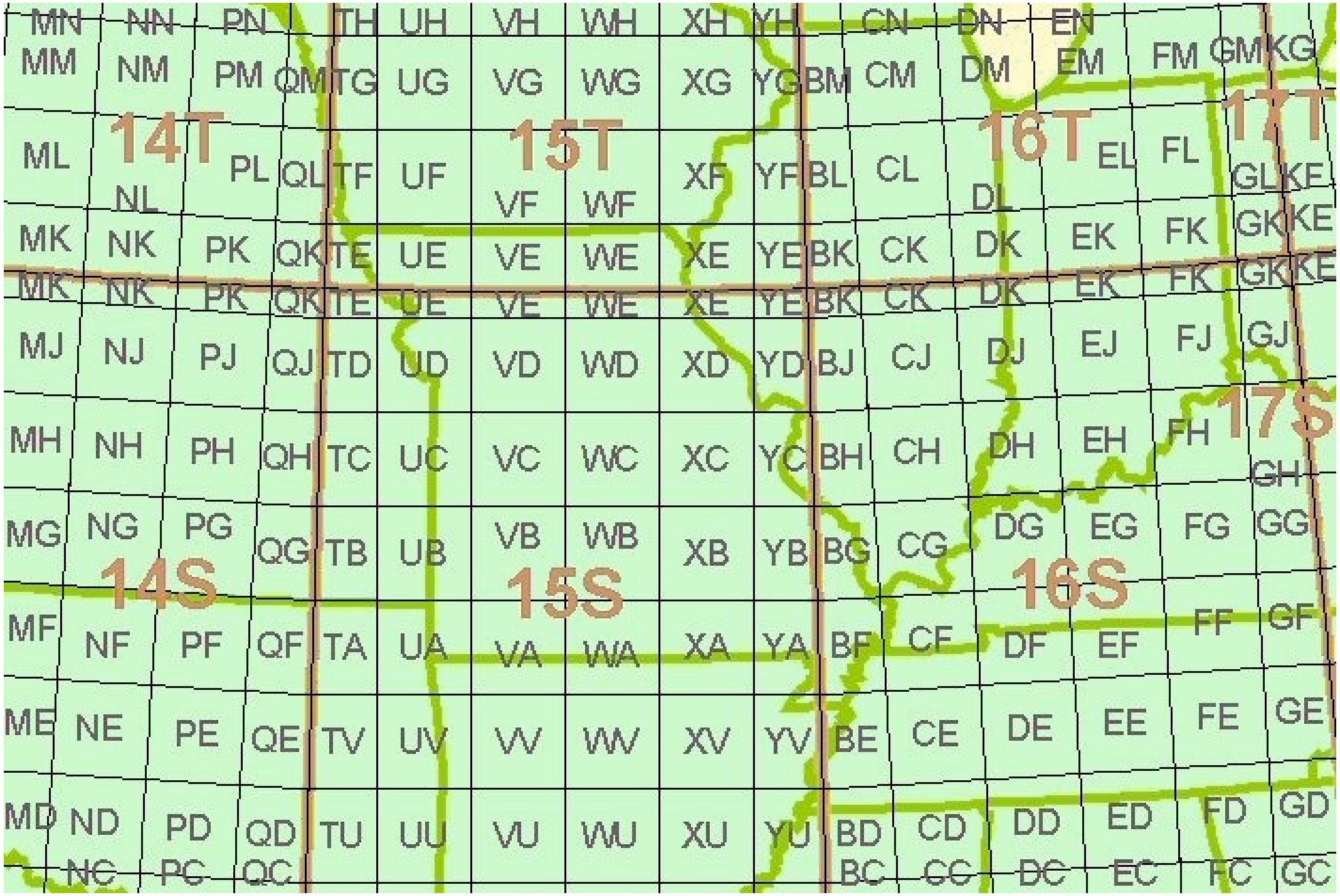
Open Data Images UK Environment Agency

Coordinate Systems



Open Data Image US NGA Office of Geomatics

Coordinate Systems



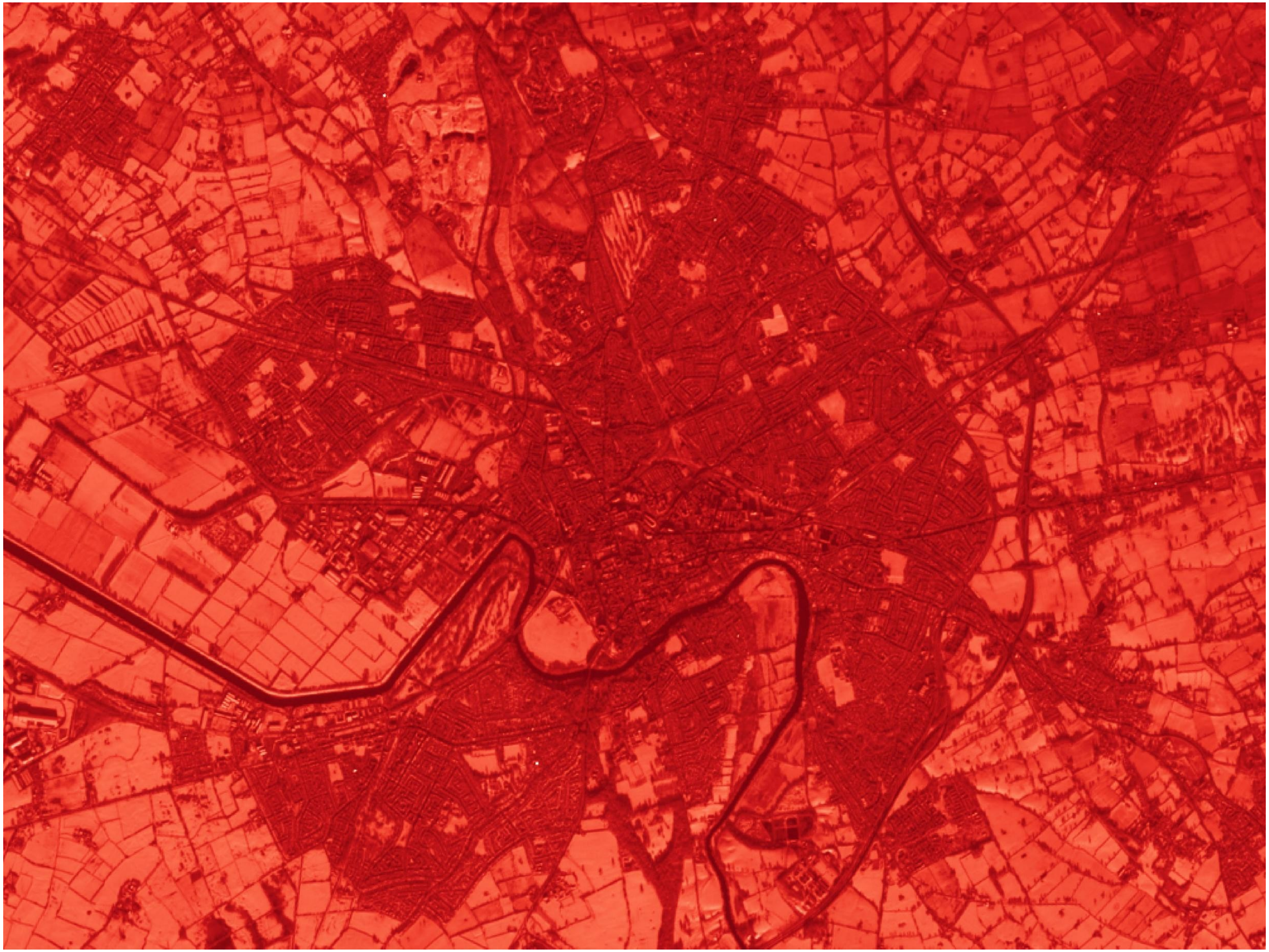
Open Data Image US NGA Office of Geomatics

Coordinate Systems

Research – Satellite imagery

Open Data Images ESA EU Copernicus Sentinel Mission

Satellite imagery



Open Data Images ESA EU Copernicus Sentinel Mission

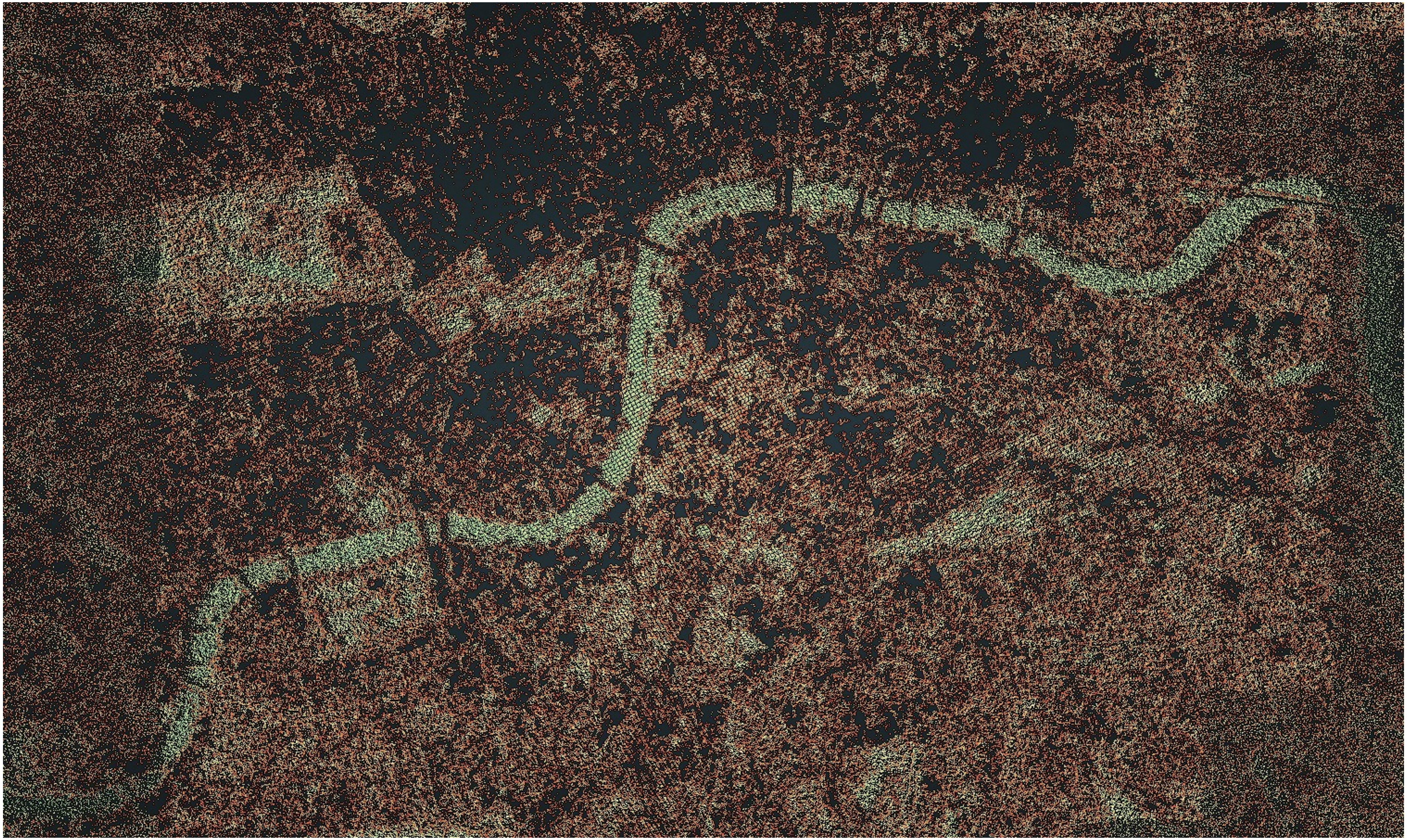
Autoencoder Upscaled Satellite imagery

Open Data Images ESA EU Copernicus Sentinel Mission

Single Aperture Radar (SAR) Data

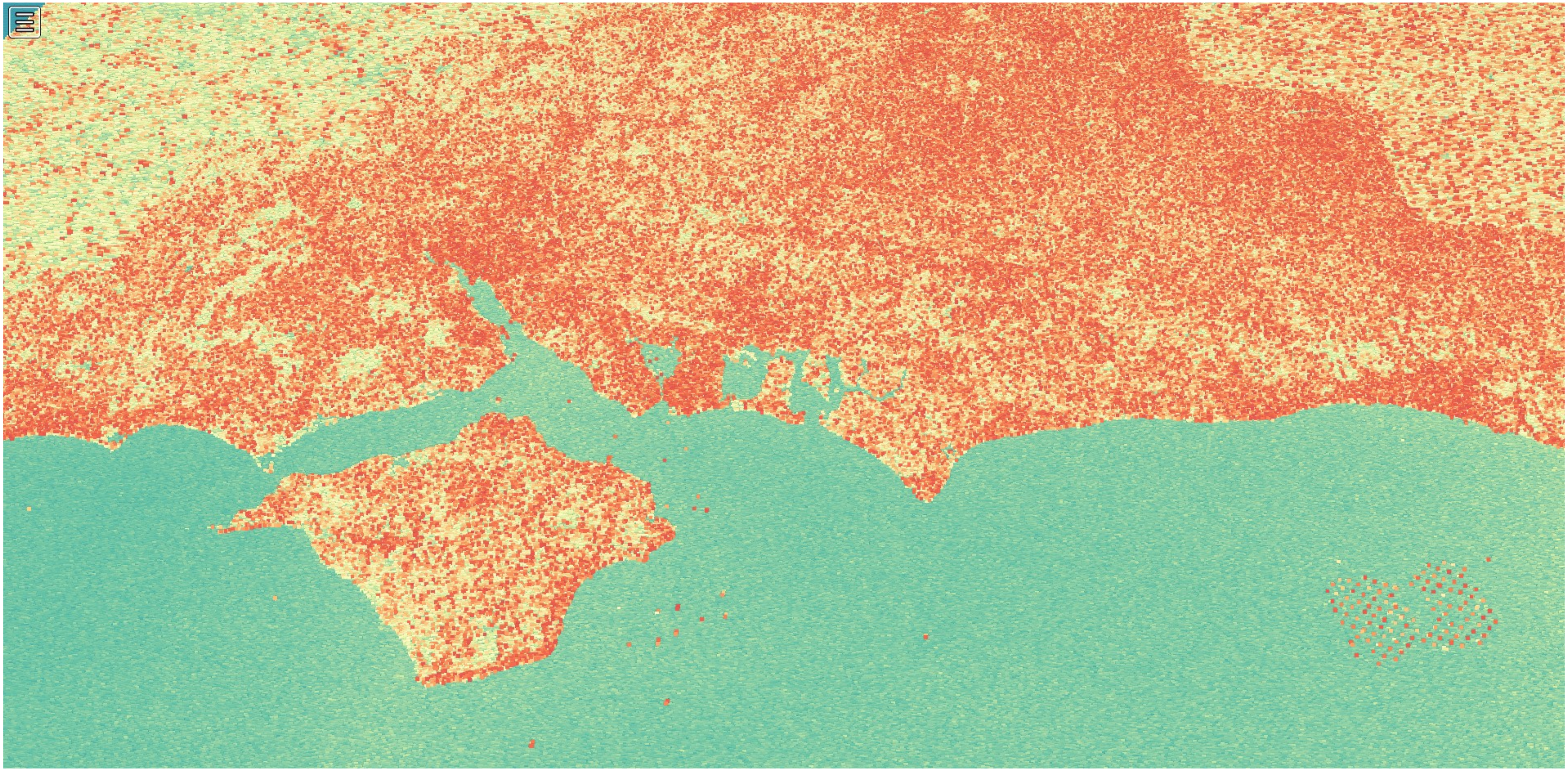
Open Data Image ESA EU Copernicus Sentinel Mission / UK Ordnance Survey

Single Aperture Radar (SAR) Data



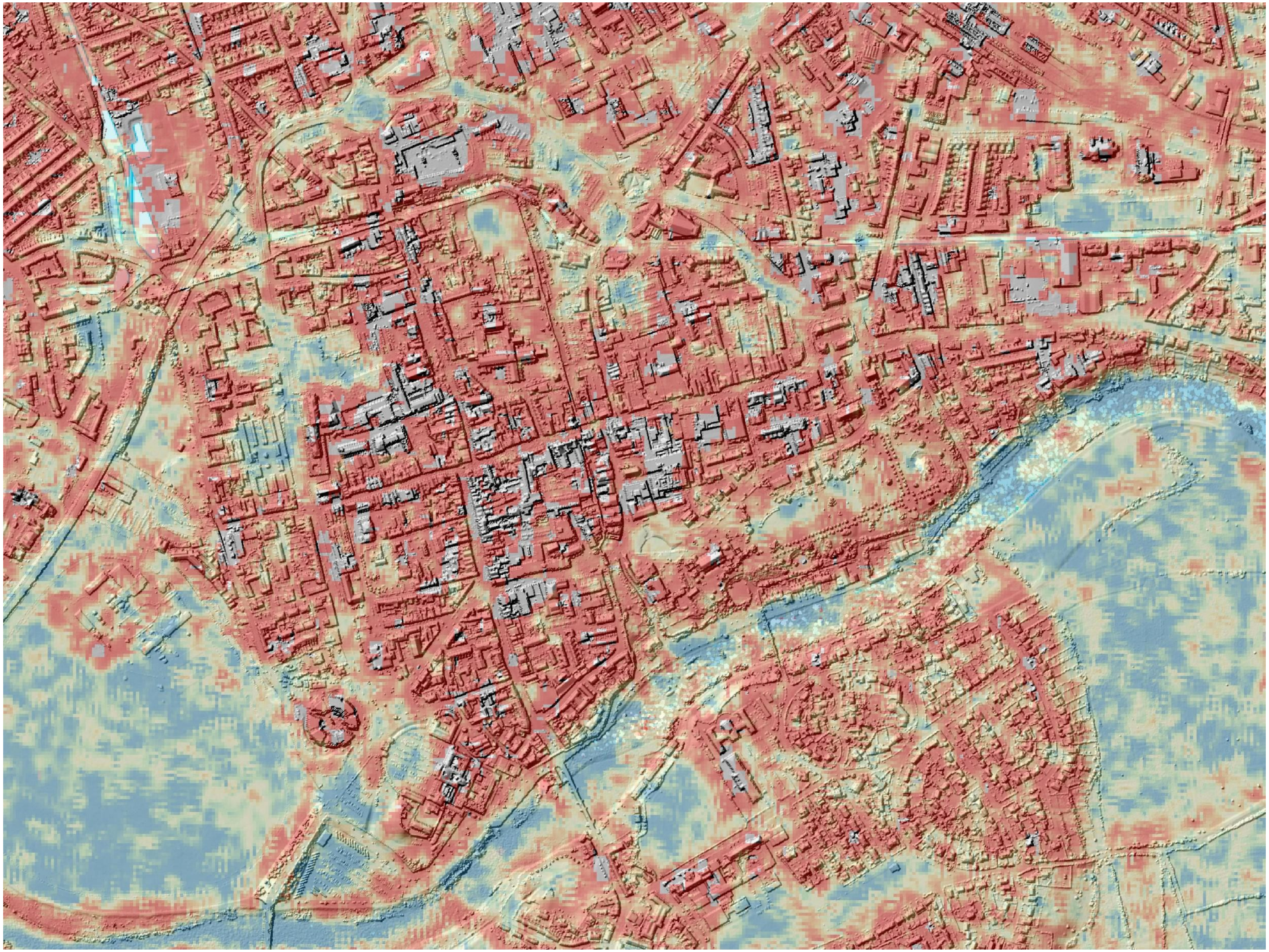
Open Data Derived Image ESA EU Copernicus Sentinel Mission

Single Aperture Radar (SAR) Data



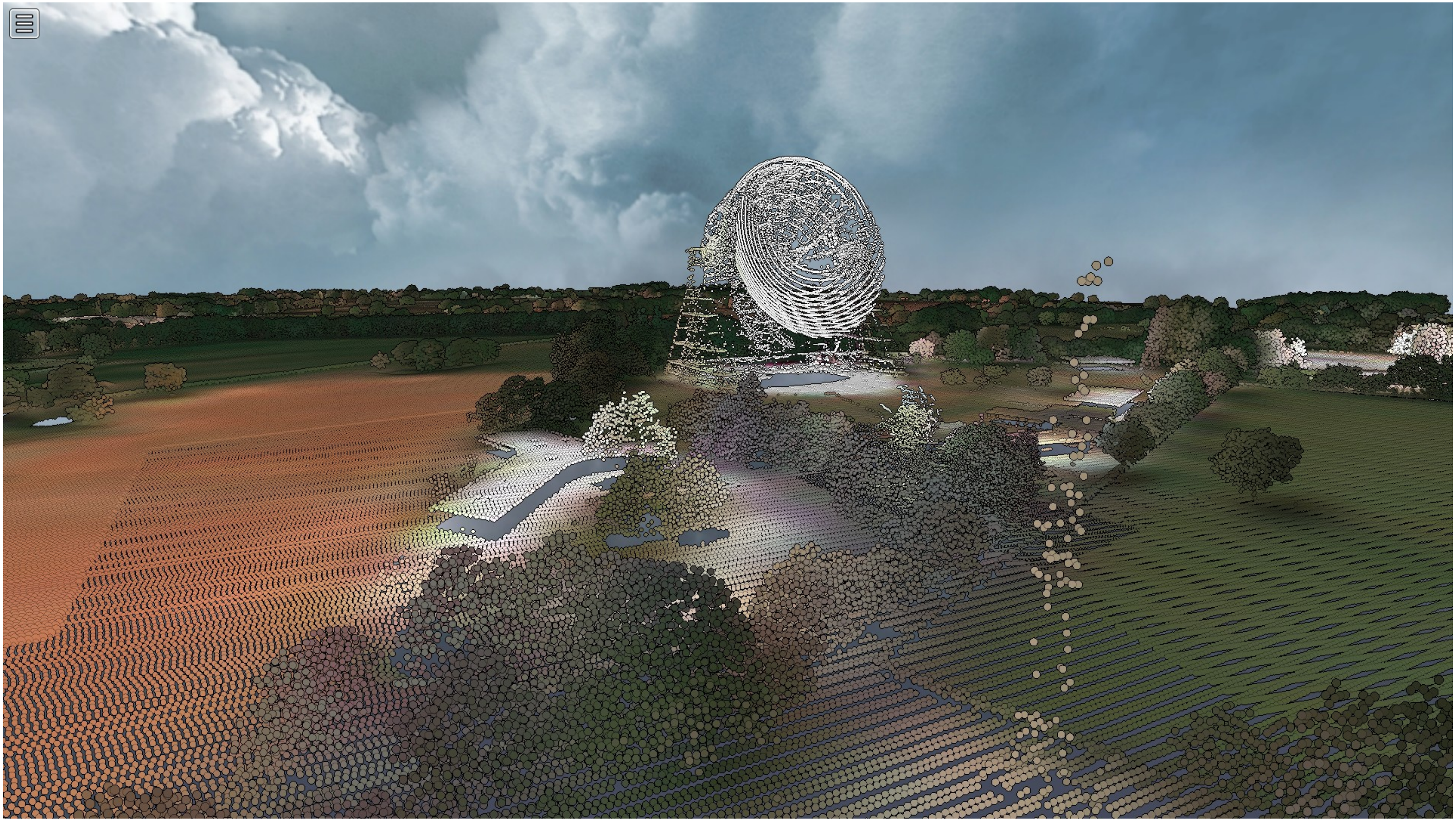
Open Data Derived Image ESA EU Copernicus Sentinel Mission

SAR and LiDAR Combined



Open Data Derived Image ESA EU Copernicus Sentinel Mission / UK Environment Agency

Satellite Imagery and LiDAR Combined



Open Data Derived Image ESA EU Copernicus Sentinel Mission / UK Environment Agency

Satellite Imagery and LiDAR Combined

Open Data Derived Image ESA EU Copernicus Sentinel Mission / USGS

Satellite Imagery and LiDAR Combined in RAPIDS

File Edit View Run Kernel Tabs Settings Help

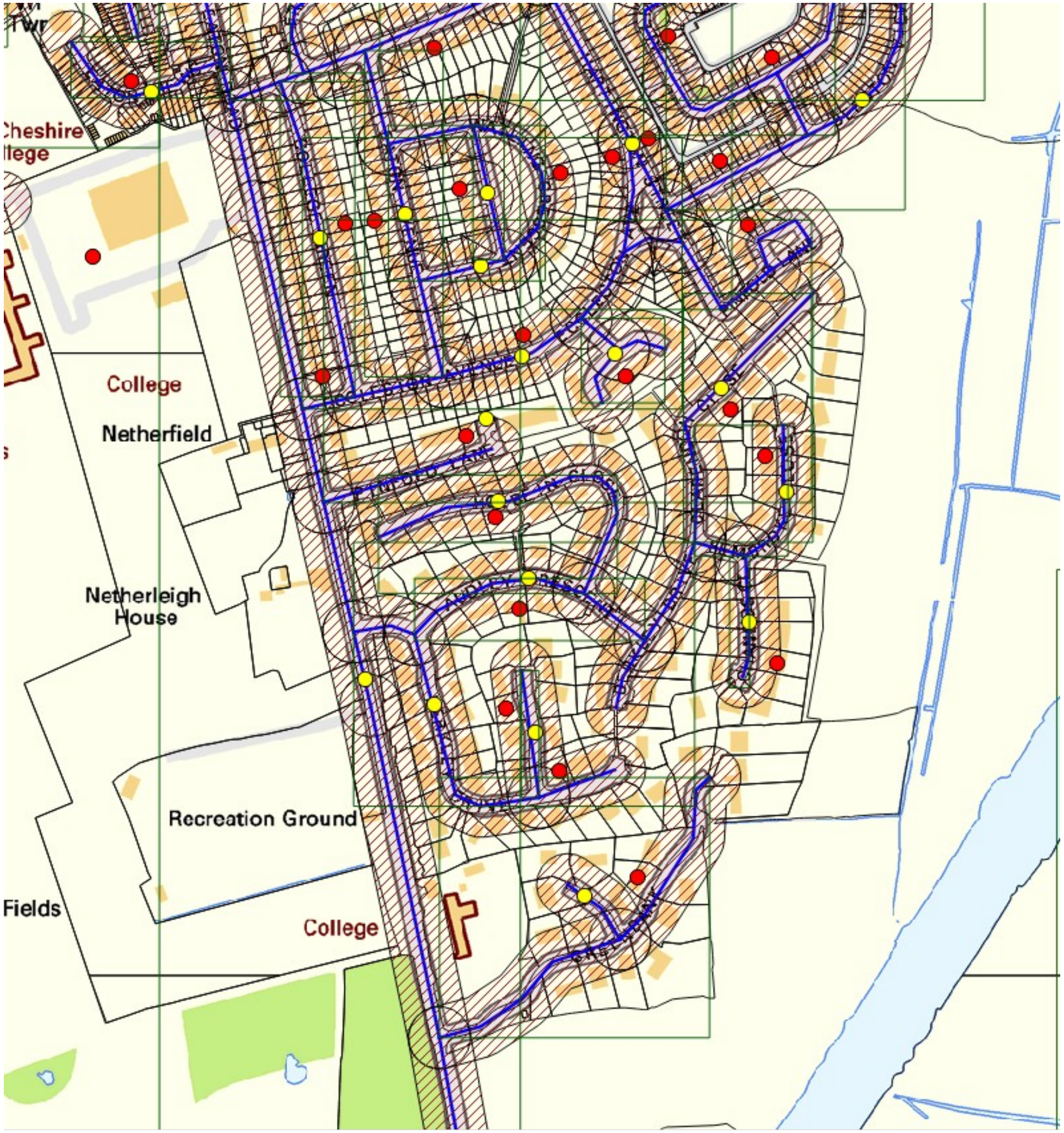
spatia_rapids_color_demo.i

Code — Python 3

```
[1]: import cudf
     from cudf.dataframe import DataFrame
     import numpy as np
     import math
     import pandas as pd
     from numba import cuda

[2]: names = ['Point_ID','ETRS89_Easting','ETRS89_Northing','ETRS89_OSGB36_EShift','ETRS89_OSGB36_NShift','ETRS89_ODN_HeightShift','Height_Datum_Flag']

[3]: dtypes = ['int64','int64','int64','float64','float64','float64','int64']

[4]: filename = '/data/ostn/OSTN15_OSGM15_DataFile.txt'

[5]: from spatia_rapids import transformations

[6]: %%time
     shift_dic = transformations.load_shifts(filename,names,dtypes,'ETRS89_Easting','ETRS89_Northing','ETRS89_OSGB36_EShift','ETRS89_OSGB36_NShift','ETRS89_ODN_HeightShift')

     CPU times: user 87.3 ms, sys: 144 ms, total: 231 ms
     Wall time: 287 ms

[7]: import laspy as lp

[8]: %%time
     lasfile = '/data/pointcloud/liv/Liverpool_Centre.las'
     inFile = lp.file.File(lasfile, mode = "r")
     print(inFile.header.min,inFile.header.max)
     print(inFile.header.get_dataformatid())

     [334000.0, 390000.0, -0.8] [335999.99, 391999.99, 139.73]
     1
     CPU times: user 2.4 ms, sys: 625 µs, total: 3.02 ms
     Wall time: 2.35 ms

[9]: origin = 496000,5913000

[10]: %%time
      point_zdf = DataFrame()
      point_zdf['x'] = inFile.x
      point_zdf['y'] = inFile.y
      point_zdf['z'] = inFile.z
      #point_zdf['n'] = inFile.num_returns[inFile.num_returns==4]

      CPU times: user 1.91 s, sys: 189 ms, total: 2.1 s
      Wall time: 911 ms

[11]: print(point_zdf)

                         x                  y     z
       0          334011.79          390001.13 -0.71
       1          334011.94          390001.26 -0.54
       2          334012.12          390001.42 -0.38
       3          334012.27          390001.54 -0.27
       4 334012.41000000003 390001.66000000003 -0.16
       5          334012.36          390001.68 -0.13
       6          334012.21          390001.55 -0.27
```

Demonstration LiDAR Processing in RAPIDS

Property Attribute Classification in LiDAR



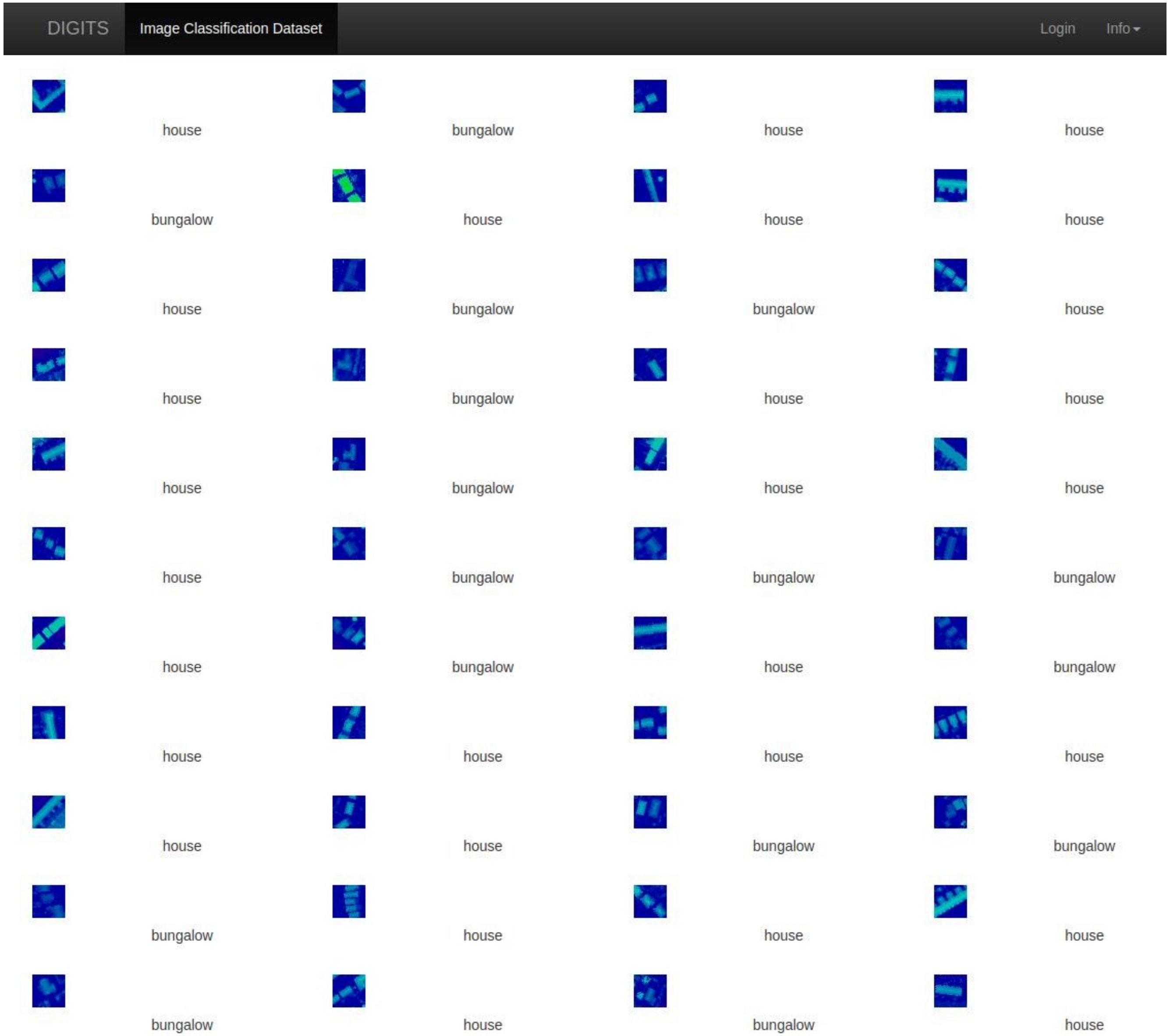
Open Data Derived Image UK Ordnance Survey

Property Attribute Classification in LiDAR

LMK_KEY	BUILDING_REFERENCE_NUMBER	ADDRESS1	POSTCODE	PROPERTY_TYPE	BUILT_FORM
337587020090803100814	1966065668	32, Commondale Drive	TS25 2AN	House	Semi-Detached
673146320110905030952	7035879868	6, Miers Avenue	TS24 9HL	House	Mid-Terrace
733339620120120100139	5400804968	13 Poppy Close	TS26 0YX	House	Detached
547496820100930060918	9899530868	30, Laurel Gardens	TS25 4NZ	Flat	*NULL*
279303520090512090529	3210151668	53, Mariners Point	TS24 0FB	House	End-Terrace
100721020080509110549	9252586468	3, Elderslie Walk	TS25 4BP	Flat	Detached
681296520110922070956	2432530968	72, Ridlington Way	TS24 9QB	House	Semi-Detached
586920720110211120216	8555633868	14, Fernville Close	TS25 4LN	Bungalow	Semi-Detached
690270320111018041032	4393990968	6, Barnard Grove	TS24 9SD	House	Semi-Detached
422193520100118040117	0699551768	15, Salisbury Place	TS26 0XJ	Flat	Mid-Terrace
554540820101019111021	1688680868	11, Rockpool Close	TS24 0TJ	House	Semi-Detached
1138040420140509020507	5633903278	5, Celandine Gardens	TS26 0ZJ	House	End-Terrace
74140820080215050255	0740894468	26, Burn Valley Road	TS26 9BS	House	End-Terrace
198577720090105050109	0223755568	22, Brimston Close	TS26 0QA	Bungalow	Detached
720599120111105121136	2968313968	76, Murray Street	TS26 8RQ	Flat	*NULL*
558106120101027081043	5364011868	77, Lime Crescent	TS24 8JW	House	Mid-Terrace
656450520110719030722	9025958868	6, Phoenix Close	TS25 3DH	Flat	*NULL*
639986220110609060653	5171247868	28, Lister Street	TS24 7QF	House	End-Terrace
512747420100713100718	5502987768	33, Commondale Drive	TS25 2AN	Bungalow	Semi-Detached
1467410320160801010848	2709636478	44, Northgate	TS24 0LJ	House	Mid-Terrace
616461120110412120413	0480475868	263a Raby Road	TS24 8HF	Flat	Mid-Terrace
1445200120160519100511	3080974478	50, Penarth Walk	TS26 0TW	Bungalow	Mid-Terrace
180353420081114091105	0074234568	48, Irvine Road	TS25 3HS	House	End-Terrace
948719520160915090924	4897769078	7, Regent Square	TS24 0QW	House	Mid-Terrace
190254220081125081148	9792684568	75, Challoner Road	TS24 8HY	House	Semi-Detached

Open Data Derived Image UK DCLG

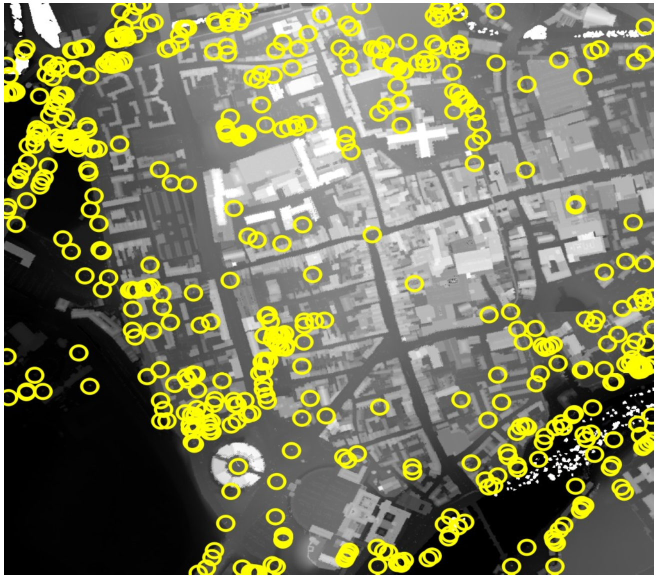
Property Attribute Classification in LiDAR

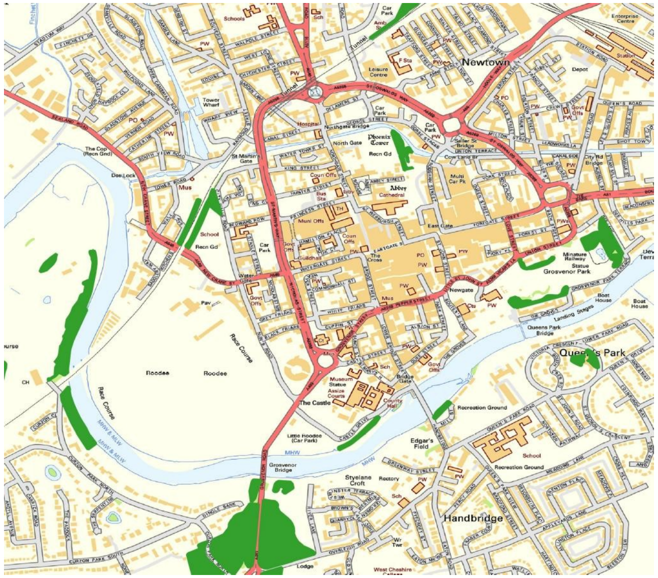


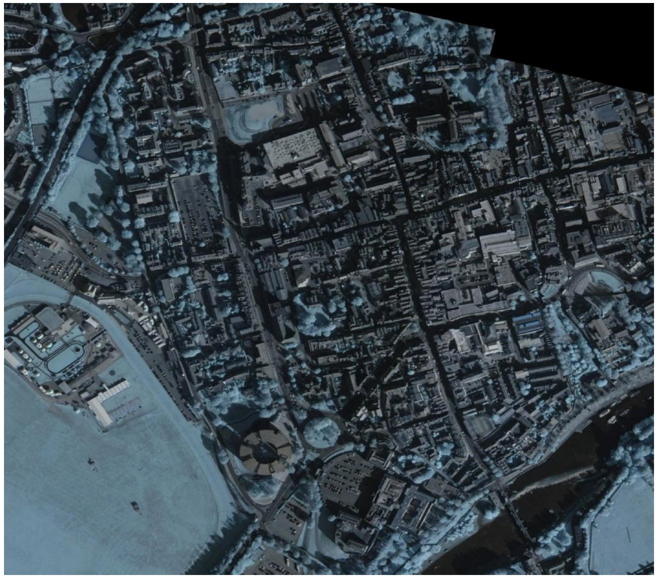
Open Data Derived Image UK Environment Agency

Demonstration Deep Learning

Deep Learning – Object Detection in LiDAR

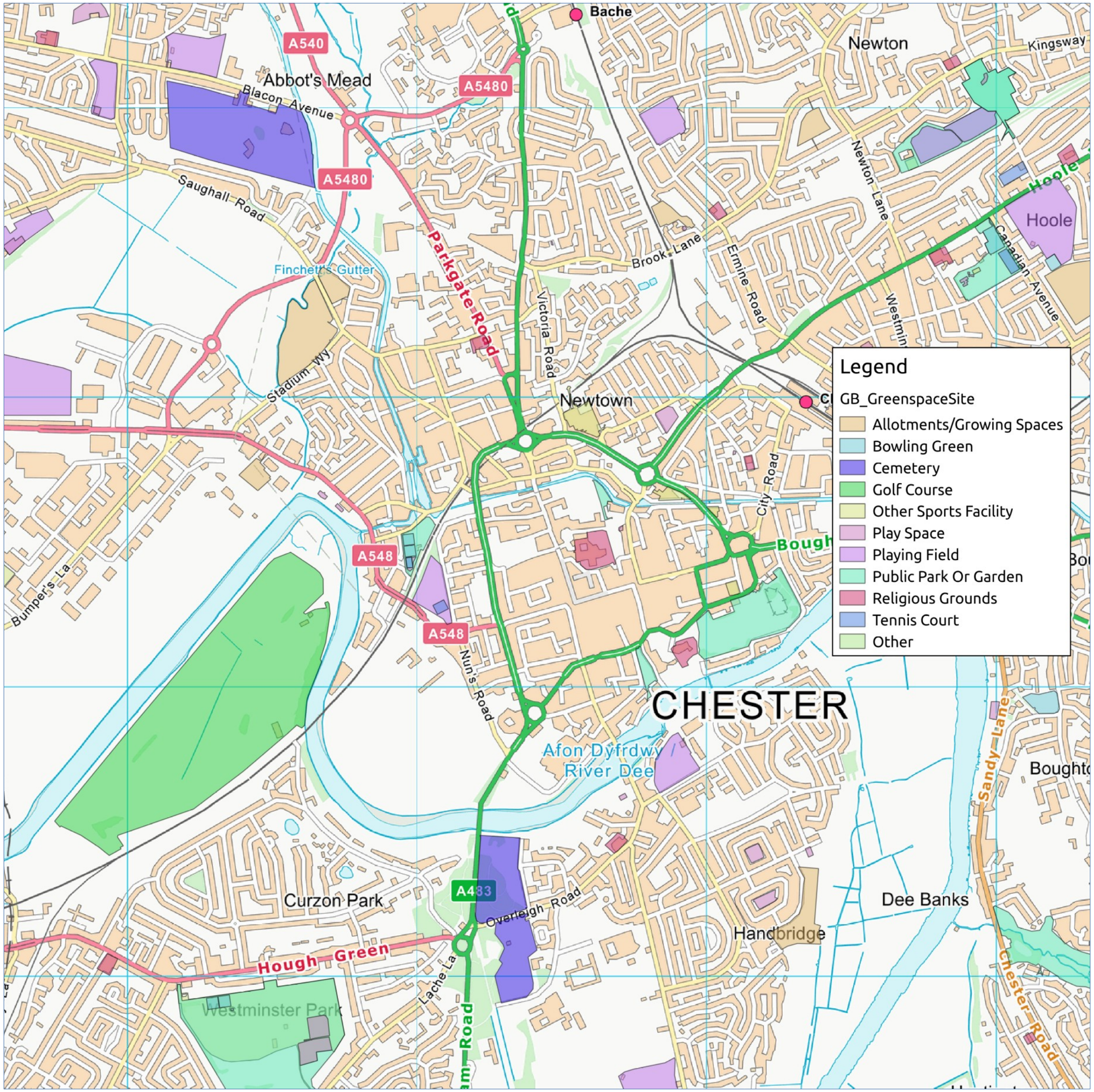






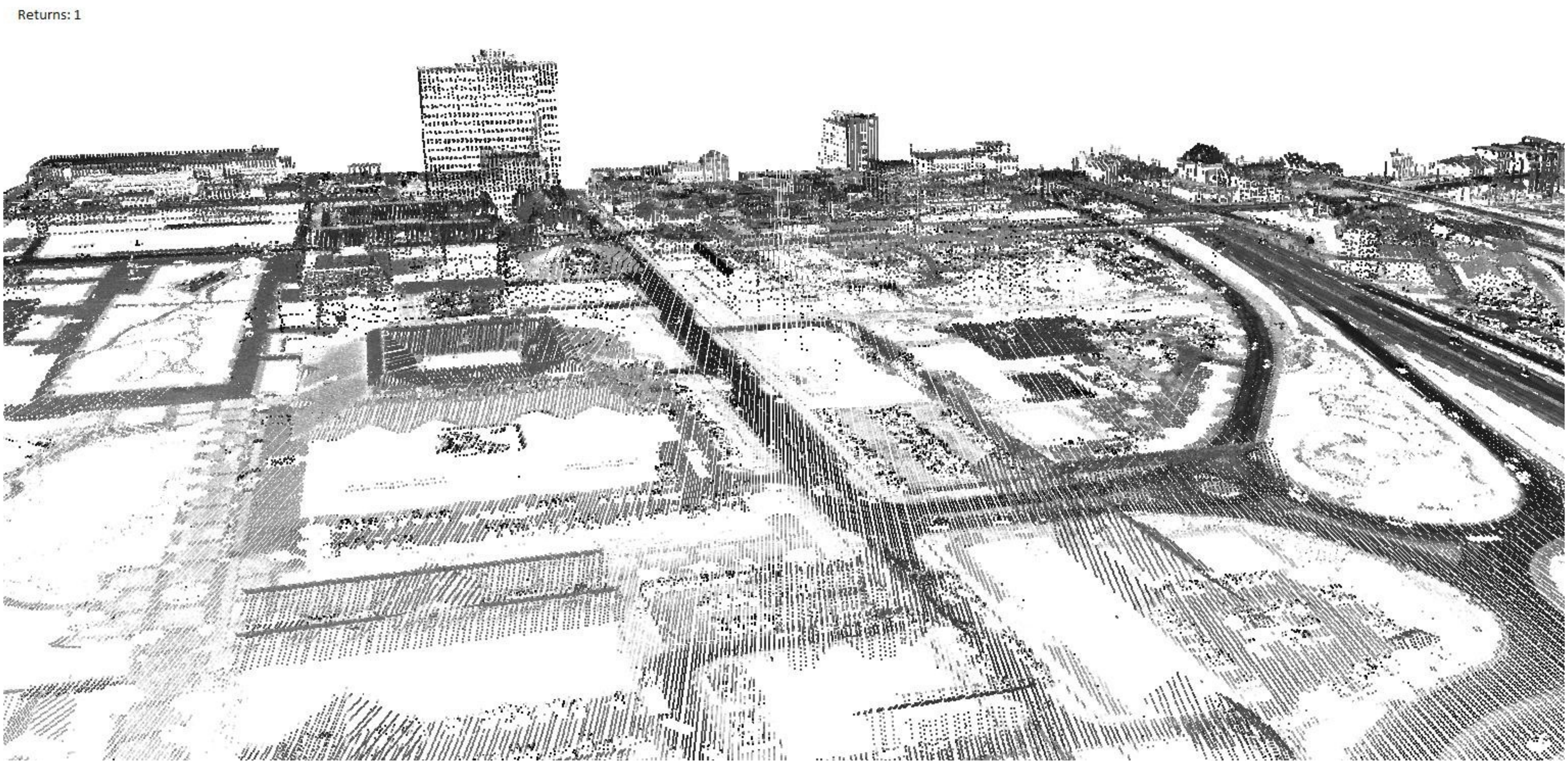
Open Data Derived Images UK Ordnance Survey / UK Environment Agency

Deep Learning – OS Greenspace



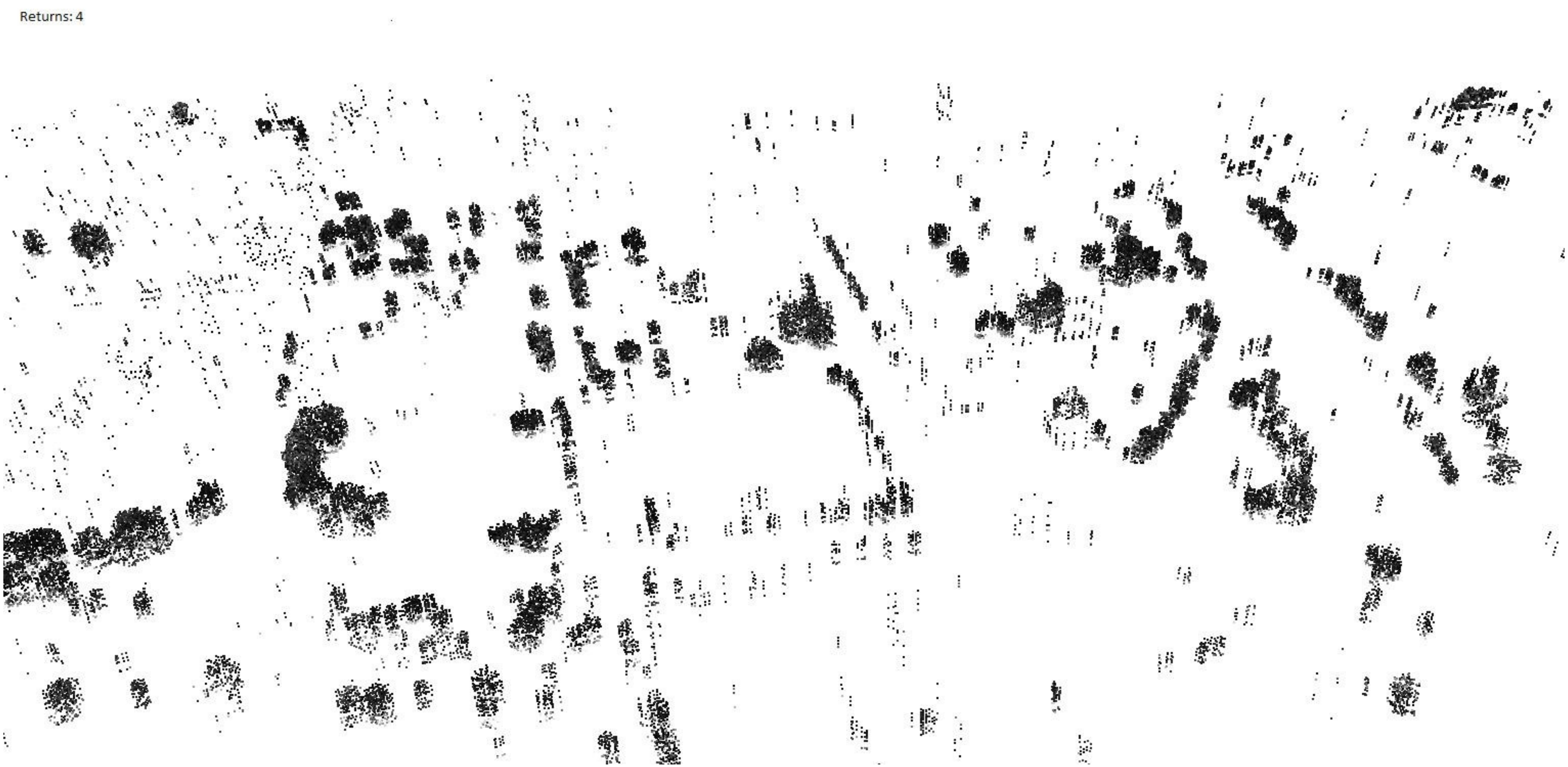
Open Data Derived Images UK Ordnance Survey

LiDAR Segmentation



Open Data Derived Image UK Environment Agency

LiDAR Segmentation



Open Data Derived Image UK Environment Agency

LiDAR Tree Detection in RAPIDS with CuML

File Edit View Run Kernel Tabs Settings Help

spatia_rapids_color_trees.ip

Code   Python 3

[1]:
```python
import cudf
from cudf.dataframe import DataFrame
import numpy as np
import math
import pandas as pd
from numba import cuda
from cuml import DBSCAN as cumlDBSCAN
```

[2]:
```python
import laspy as lp
```

[3]:
```python
%%time
lasfile = '/data/pointcloud/liv/Liverpool_Centre.las'
inFile = lp.file.File(lasfile, mode = "r")
```

```
CPU times: user 0 ns, sys: 2.91 ms, total: 2.91 ms
Wall time: 2.42 ms
```

[4]:
```python
%%time
point_zdf = DataFrame()
point_zdf['x'] = inFile.x[inFile.return_num==4]
point_zdf['y'] = inFile.y[inFile.return_num==4]
point_zdf['z'] = inFile.z[inFile.return_num==4]
```

```
CPU times: user 1.83 s, sys: 172 ms, total: 2 s
Wall time: 860 ms
```

[5]:
```python
eps = 3
min_samples = 2
```

[6]:
```python
%%time
clustering_cuml = cumlDBSCAN(eps = eps, min_samples = min_samples)
clustering_cuml.fit(point_zdf)
```

```
CPU times: user 629 ms, sys: 779 µs, total: 629 ms
Wall time: 187 ms
```

[7]:
```python
point_zdf["l"] = clustering_cuml.fit_predict(point_zdf)
```

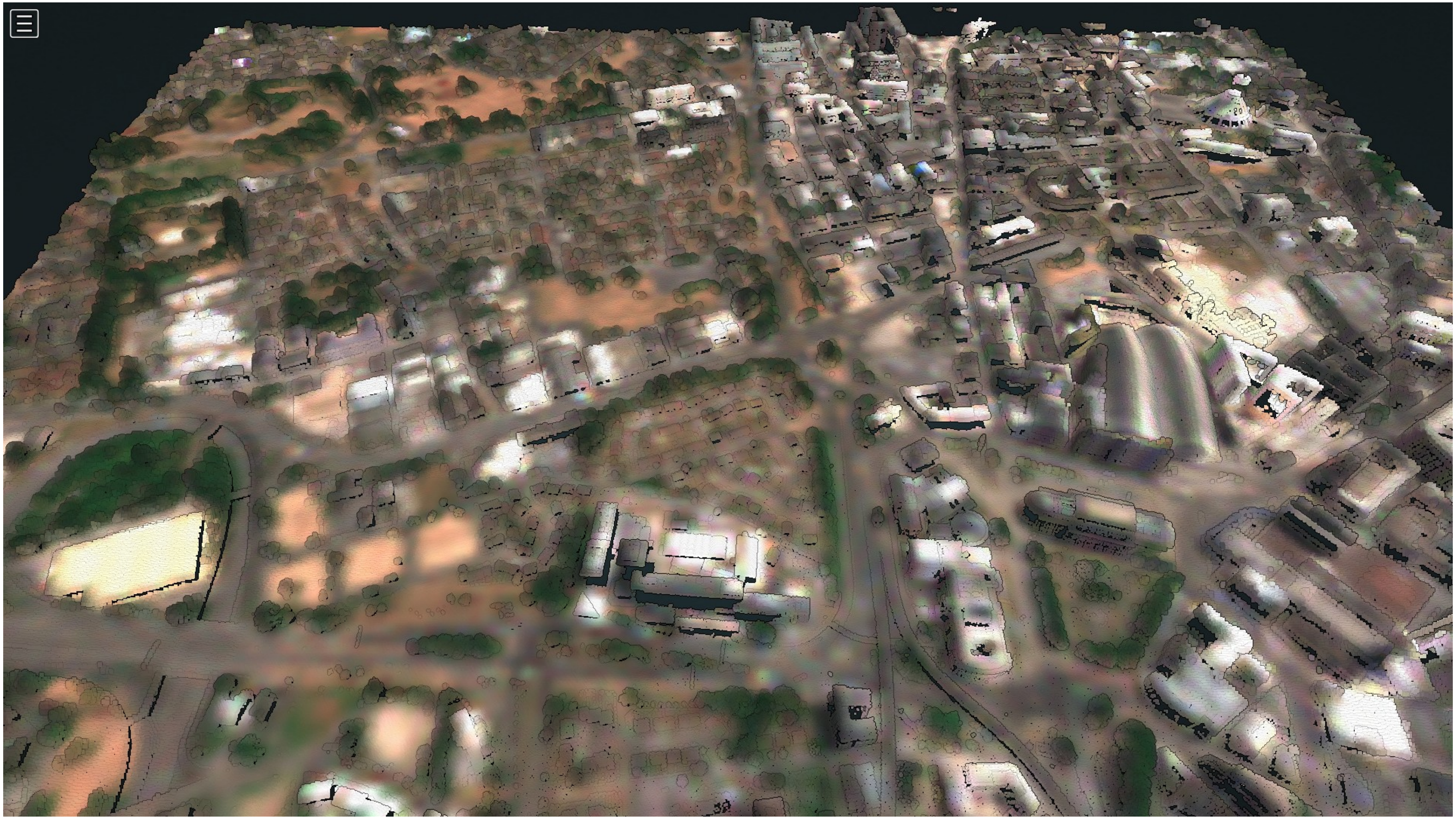
[8]:
```python
trees = point_zdf.query("l >= 0")
```

[9]:
```python
trees.drop_column("l")
```

[10]:
```python
print(trees)
```

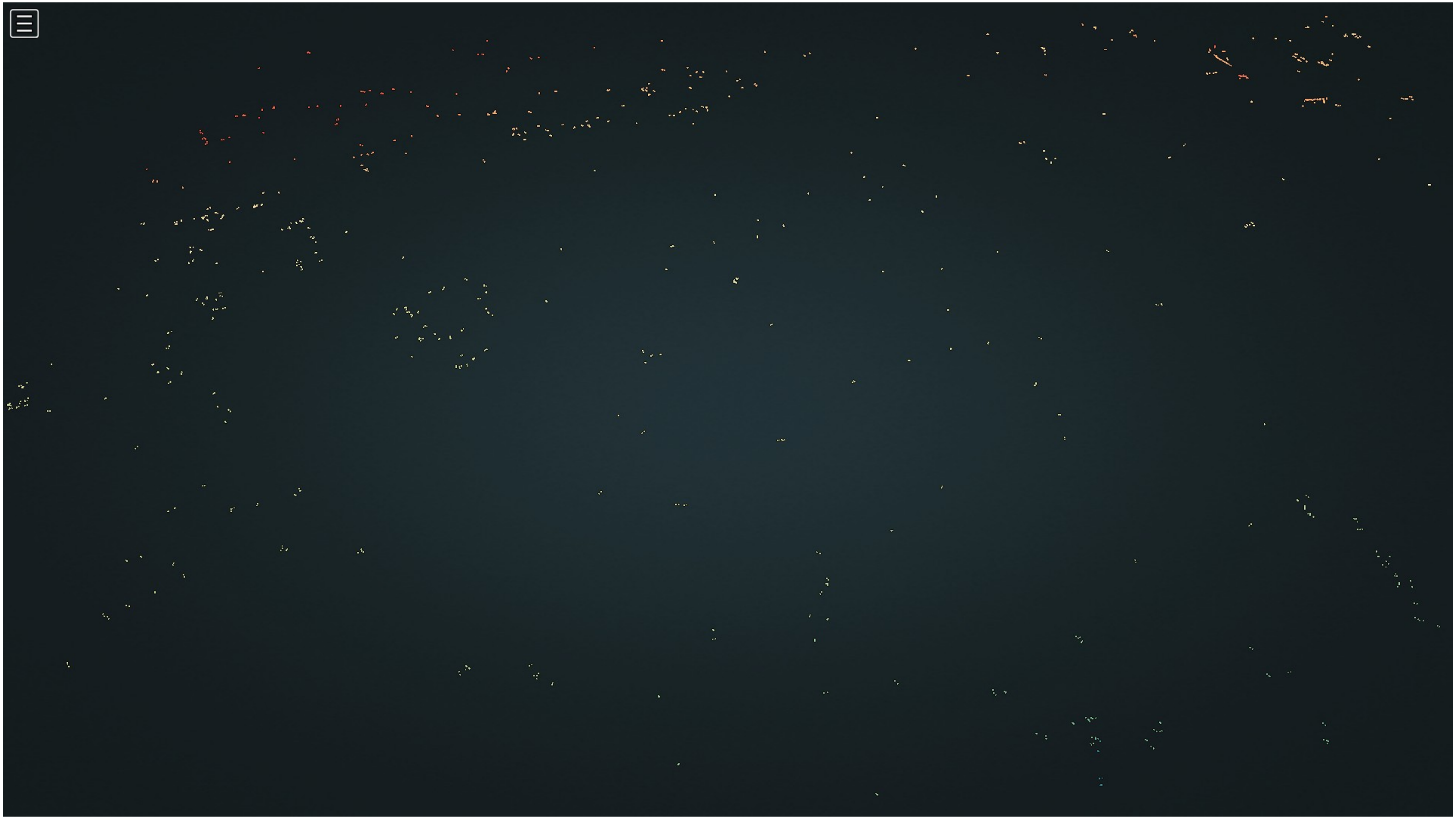
```
                   x                 y                z
 1         334021.23         391927.01            13.52
 2         334021.21         391926.95            13.61
 3         334071.26         391837.87 14.290000000000001
 5         334015.74 391688.41000000003 16.740000000000002
 6          334015.7 391688.10000000003            16.75
14         334144.01         391663.71 19.580000000000002
15         334143.48         391663.32            19.56
25 334195.41000000003         391622.94            20.32
27         334294.52         391614.95            20.28
28 334294.85000000003         391615.19            20.32
[2087 more rows]
```

LiDAR Tree Detection in RAPIDS with CuML



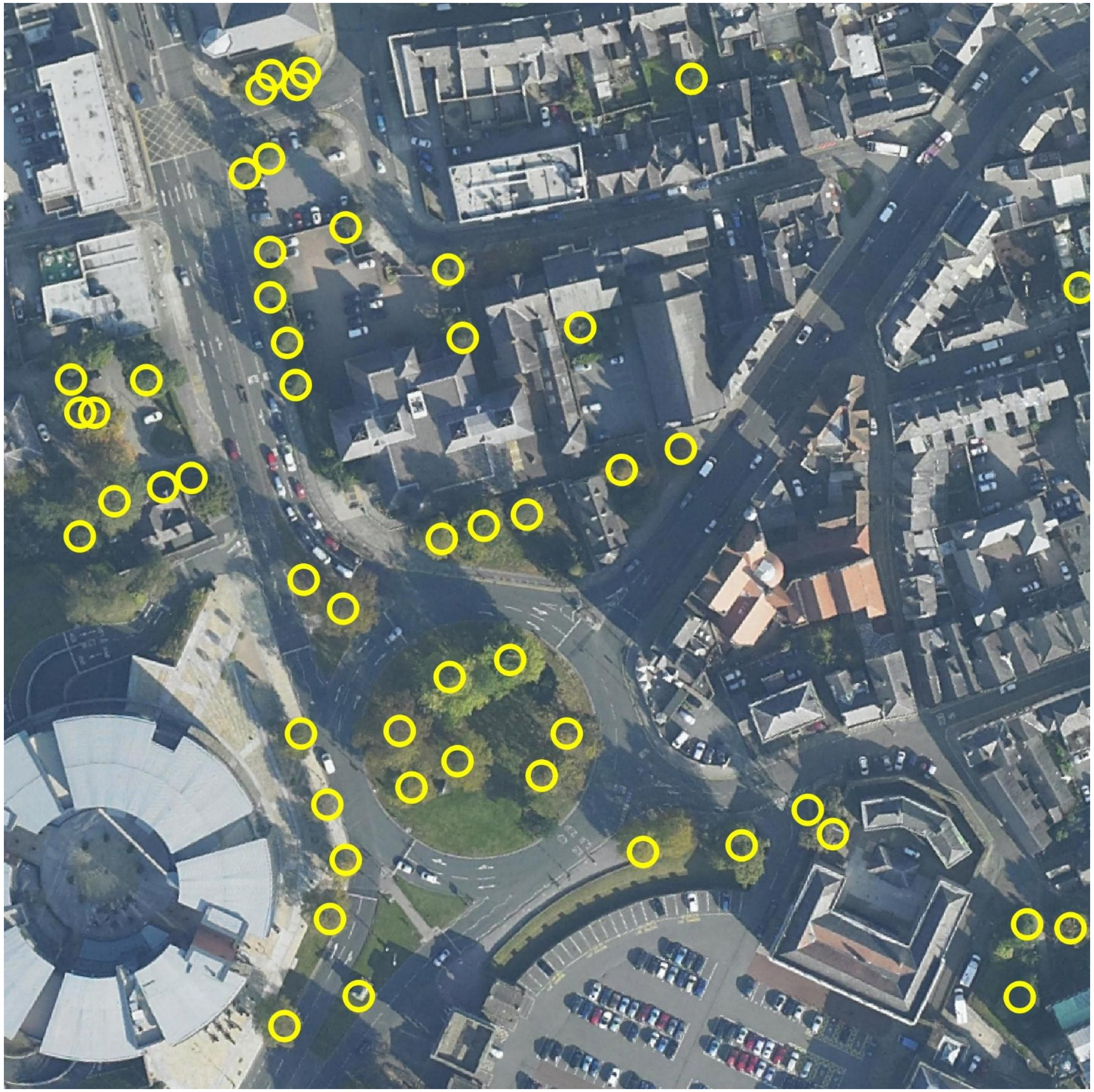
Open Data Derived Image UK Environment Agency

LiDAR Tree Detection in RAPIDS with CuML



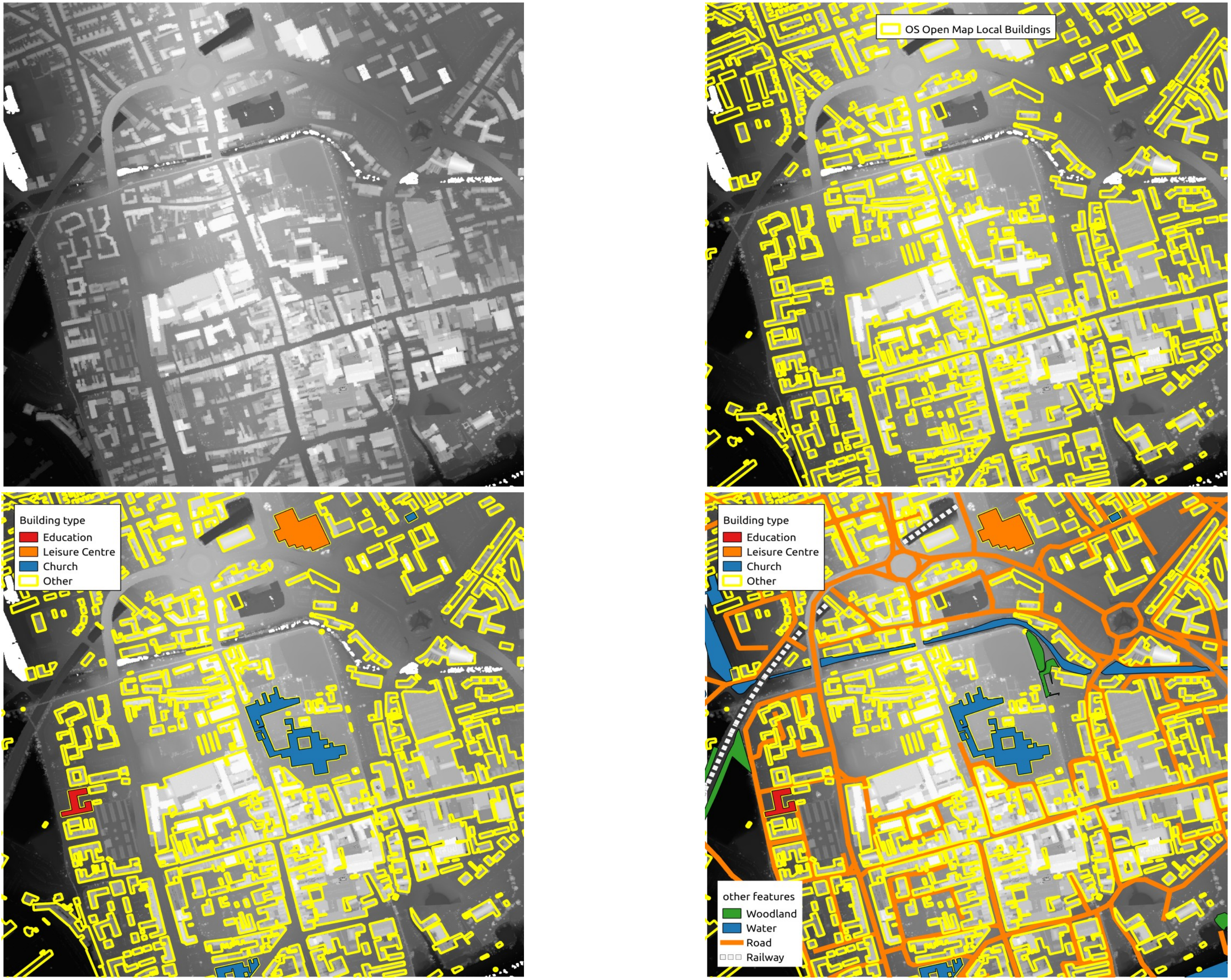
Open Data Derived Image UK Environment Agency

Tree Detection in LiDAR



Open Data Derived Images UK Ordnance Survey / UK Environment Agency

Object Detection from LiDAR



Open Data Derived Images UK Ordnance Survey / UK Environment Agency

Object Detection Labeling in RAPIDS

oml_churches_test.ipynb

Code

Python 3

```
[1]: import cudf
     from spatia_rapids import projections
     from spatia_rapids import transformations
     import numpy as np

[2]: ostn_names = ['Point_ID','ETRS89_Easting','ETRS89_Northing','ETRS89_OSGB36_EShift','ETRS89_OSGB36_NShift','ETRS89_ODN_HeightShift','Height_Datum_Flag']
     ostn_dtypes = ['int64','int64','int64','float64','float64','float64','int64']
     ostn_filename = '/data/ostn/OSTN15_OSGM15_DataFile.txt'

[3]: %%time
     shift_dic = transformations.load_shifts(ostn_filename,ostn_names,ostn_dtypes,'ETRS89_Easting','ETRS89_Northing','ETRS89_OSGB36_EShift','ETRS89_OSGB36_NShift','ETRS89_ODN_HeightShift')

     CPU times: user 99.1 ms, sys: 220 ms, total: 319 ms
     Wall time: 341 ms

[4]: names = ['ID','ring','point','x','y']
     dtypes = ["int64","int64","int64","float64","float64"]
     filename = "/data/vectors/oml_churches.csv"

[5]: %%time
     vdf = cudf.read_csv(filename,names=names,dtype=dtypes,skiprows=1)

     CPU times: user 12.9 ms, sys: 4.29 ms, total: 17.2 ms
     Wall time: 16.7 ms

[6]: %%time
     vdf = vdf.apply_rows(transformations.rapids_etrs2osgb,incols=['x', 'y'],outcols=dict(es=np.float64,ns=np.float64,hs=np.float64,adj=np.int8),kwargs=shift_dic)

     CPU times: user 636 ms, sys: 15 ms, total: 651 ms
     Wall time: 649 ms

[7]: vdf['east'] = vdf['x']-vdf['es']
     vdf['north'] = vdf['y']-vdf['ns']

[8]: proj_etrs = projections.get_proj_parameters(27700,ellipsoid=4258)
     proj_utm30 = projections.get_proj_parameters(25830)

     ('GRS_1980', 6378137.0, 6356752.314140356, '298.257222101', 0.003352810681182319)
     {'n': 0.0016792203946287211, 'PHI0': 0.8552113334772214, 'N0': -100000.0, 'a': 6378137.0, 'e_sqr': 0.006694380022900686, 'bF0': 6354217.697096618, 'LAMBDA0': -0.03490658503988659, 'aF0': 6375593.8562768
     23, 'E0': 400000.0, 'F0': 0.9996012717, 'b': 6356752.314140356}
     ('GRS_1980', 6378137.0, 6356752.314140356, '298.257222101', 0.003352810681182319)
     {'n': 0.0016792203946287211, 'PHI0': 0.0, 'N0': 0.0, 'a': 6378137.0, 'e_sqr': 0.006694380022900686, 'bF0': 6354209.6132147005, 'LAMBDA0': -0.05235987755982988, 'aF0': 6375585.745200001, 'E0': 500000.0,
     'F0': 0.9996, 'b': 6356752.314140356}

[9]: %%time
     vdf = vdf.apply_rows(projections.rapids_en2latlon,incols=['east', 'north'],outcols=dict(lat=np.float64,lon=np.float64),kwargs=proj_etrs)

     CPU times: user 709 ms, sys: 36 ms, total: 745 ms
     Wall time: 741 ms

[10]: print(vdf)

        ID ring point                 x            y adj               es ...                  lon
     0   1    0     0 460054.5900000001   1201094.46   1 102.93099739657141 ... -0.9022432882672774
     1   1    0     1          460066.9   1201094.96   1 102.93117743282399 ... -0.9020177824183705
     2   1    0     2         460067.81    1201072.7   1   102.930899369787 ... -0.9020079284275687
     3   1    0     3          460055.5    1201072.2   1 102.93071960710002 ... -0.9022334328822663
     4   1    0     4 460054.5900000001   1201094.46   1 102.93099739657141 ... -0.9022432882672774
```

Object Detection Labeling in RAPIDS

oml_churches_test.ipynb

```
CPU times: user 709 ms, sys: 36 ms, total: 745 ms
Wall time: 741 ms
```

[10]:
```
print(vdf)
```

```
     ID ring point                 x                  y  adj                es ...                  lon
 0    1    0     0 460054.5900000001         1201094.46    1 102.93099739657141 ... -0.9022432882672774
 1    1    0     1          460066.9         1201094.96    1 102.93117743282399 ... -0.9020177824183705
 2    1    0     2         460067.81          1201072.7    1   102.930899369787 ... -0.9020079284275687
 3    1    0     3          460055.5          1201072.2    1 102.93071960710002 ... -0.9022334328822663
 4    1    0     4 460054.5900000001         1201094.46    1 102.93099739657141 ... -0.9022432882672774
 5    6    0     0         461447.13         1208836.01    1 103.05450175515129 ... -0.8743238732757238
 6    6    0     1         461448.27 1208846.6800000004    1 103.05466216124361 ... -0.8742996054706821
 7    6    0     2         461471.36         1208844.22    1 103.05497183153919 ... -0.8738767738085924
 8    6    0     3         461470.22         1208833.55    1 103.05481118188099 ... -0.8739010428465085
 9    6    0     4         461447.13         1208836.01    1 103.05450175515129 ... -0.8743238732757238
[192755 more rows]
[5 more columns]
```

[11]:
```
%%time
vdf = vdf.apply_rows(projections.rapids_latlon2en,incols=['lat', 'lon'],outcols=dict(east=np.float64,north=np.float64),kwargs=proj_utm30)
```

```
CPU times: user 499 ms, sys: 11.3 ms, total: 510 ms
Wall time: 509 ms
```

[12]:
```
print(vdf[['east','north']])
```

```
                east              north
 0 614554.7220852895 6729927.903818589
 1 614567.0242868561 6729928.591401609
 2 614568.2734215639 6729906.345545966
 3 614555.9712196725 6729905.657963582
 4 614554.7220852895 6729927.903818589
 5 615829.2509406238 6737690.593149093
 6  615830.228243363 6737701.280419619
 7 615853.3554255705 6737699.172567375
 8 615852.3781230241  6737688.48529628
 9 615829.2509406238 6737690.593149093
[192755 more rows]
```

[13]:
```
pdf = vdf.to_pandas()
```

[14]:
```
gb = pdf.groupby(['ID'])
bb = gb.agg({'east' : [np.min, np.max],'north' : [np.min, np.max]})
```

[15]:
```
gb = pdf.groupby(['ID'])
bb.to_csv("/data/vectors/church_bb.csv")
```

Object Detection Labeling in RAPIDS

Open Data Derived Image ESA EU Copernicus Sentinel Mission / UK Environment Agency Ordnance Survey

Terrain Interpolation & Analysis in RAPIDS

```
from spatia_rapids import projections
```

```
from spatia_rapids import dem
```

```
d = dem.dem()
```

```
import os.path
import fnmatch

folder = '/data/dem/'

matches = []
for root, dirnames, filenames in os.walk(folder):
  for filename in fnmatch.filter(filenames, '*.asc'):
    matches.append(os.path.join(root, filename))
```

```
idx_array = np.full((9301), -1, dtype=int)
grid_array = []
```

```
def getOSGridRecNo(e,n):
    east_index = int(e/10000);
    north_index = int(n/10000);
    return east_index + (71 * north_index) + 1;
```

```
%%time
for i,f in enumerate(sorted(matches)):
    d.loadASCII(f)
    idx = getOSGridRecNo(d.metadata['xllcorner'],d.metadata['yllcorner'])-1
    idx_array[idx] = i
    #print(i,idx,f)
    grid_array += [d.data]
```

```
CPU times: user 15.9 s, sys: 209 ms, total: 16.2 s
Wall time: 16.1 s
```

```
from numba import cuda
cuda_grid = cuda.to_device(np.array(grid_array))
```

```
cuda_idx = cuda.to_device(idx_array)
```

```
cuda_grid.shape
```

Terrain Interpolation & Analysis in RAPIDS

```
@cuda.jit(device=True)
def cu_getOSGridRecNo(e,n):
    east_index = int(e/10000);
    north_index = int(n/10000);
    return east_index + (71 * north_index) + 1;
```

```
@cuda.jit(device=True)
def cu_calc_height(E, N, grid,idx):
    # Calculate point offset within grid square
    grid_rec = cu_getOSGridRecNo(E,N) - 1
    if grid_rec < 0 or grid_rec > idx.shape[0] or idx[grid_rec] < 0:
        return 0.0
    east_origin = math.floor((E + 25) / 50) * 50 - 25
    north_origin = math.floor((N + 25) / 50) * 50 - 25
    east_offset = int((east_origin - int(east_origin / 10000) * 10000) / 50)
    north_offset = 199 - int((north_origin - int(north_origin / 10000) * 10000) / 50)
    print (east_origin,north_origin)
    print (east_offset,north_offset)
    t = (E - east_origin) / 50
    u = (N - north_origin) / 50
    # If point is a grid corner, return the shift
    if (E + 25) % 50 == 0 and (N + 25) % 50 == 0:
        return grid[idx[grid_rec]][north_offset][east_offset]
    # Else use bilinear interpolation to estimate shift within the grid.
    else:
        height = 0.0
        # Calculate point offset within grid square
        # For each corner of the enclosing grid square
        for xi in range(0,2):
            for yi in range(0,2):
                grid_rec = cu_getOSGridRecNo(east_origin+xi*50,north_origin+yi*50) -1
                if idx[grid_rec] < 0:
                    return 0.0
                eidx = (east_offset + xi) % 200
                nidx = (north_offset - yi) % 200
                print(east_origin+xi*50,north_origin+yi*50,grid_rec,eidx,nidx)
                # Calculate bilinear adjustment factor (area of rectangle define by point and corner) and apply it to shift at relevant corner
                factor = ((1 - xi) + (2 * xi - 1) * t) * ((1 - yi) + (2 * yi - 1) * u)
                h = grid[idx[grid_rec]][nidx][eidx]
                height += factor * h
                print(grid[idx[grid_rec]][nidx][eidx])

        return (height)
```

```
def rapids_calc_heights(x, y, h, cuda_grid,cuda_idx):
    for i, (E, N) in enumerate(zip(x,y)):
        h[i] = cu_calc_height(E,N,cuda_grid,cuda_idx)
```

```
%%time
hdf = adf.apply_rows(rapids_calc_heights,incols=['x', 'y'],outcols=dict(h=np.float64),kwargs=dict(cuda_grid=cuda_grid,cuda_idx=cuda_idx))
```

```
CPU times: user 3min 13s, sys: 2min 17s, total: 5min 31s
Wall time: 5min 13s
```

Terrain Interpolation & Analysis in RAPIDS

```
print(hdf[['x','y','h']])
```

```
                    x          y                  h
 0 358263.47000000003 172798.15 14.971814348801894
 1           352967.0  181077.0  6.812000163269042
 2           352967.0  181077.0  6.812000163269042
 3           354800.0  180469.0               6.97
 4           354796.0  180460.0  6.911400001525879
 5           353473.0  180409.0  7.487519968414307
 6           352548.0  180308.0  7.708360050964355
 7           352515.0  180360.0   7.50000002861023
 8           352462.0  180401.0 7.3659998798370365
 9           354662.0  180364.0  6.717719916343689
[37961339 more rows]
```

Terrain Interpolation & Analysis in RAPIDS

Open Data Derived Image ESA EU Copernicus Sentinel Mission / UK Ordnance Survey

By using RAPIDS & TensorFlow

- We have processed large 3D datasets
- Re-projected coordinate systems with high accuracy
- Merged datasets in different formats & coordinates to enrich training data
- Labelled LiDAR & satellite images for training
- Been able to interpret and deploy the results of AI

In terms of our business objectives

- AI models have accurately predicted property attributes
- These remove the need to ask questions
- Removes the risk of incorrect information being supplied by the customer due to fraud or subjectivity
- Provided additional insights to the insurance companies
- Considerably enhance the accuracy of underwriting decisions

Other applications under development

- Spatial microsimulation of small-area statistics estimates
- Using cuML clustering on autoencoder output for unsupervised classification of imagery
- Creating 3D deep learning models with TensorFlow Sparse Tensors and Conv3D layers
- Using hybrid image and sensor data combined objects as training data

Conclusions

- RAPIDS simplifies the process of GPU acceleration of computationally intensive applications
- RAPIDS protects your legacy investments by allowing code to be reused with minimal adaptation
- RAPIDS is extensible
- RAPIDS is a versatile ~~data~~ science accelation platform