New Jersey 2018 LiDAR dataset

https://registry.opendata.aws/nj-lidar/

180GB of LAS files resulting in a 51GB TileDB array.

In [1]:

```
!ogrinfo -al -so ~/working/nj_shapefiles/312017118_GPSC3_New_Jersey_Lidar_Index.shp
```

```
INFO: Open of `/home/jovyan/working/nj_shapefiles/312017118_GPSC3_New_Jersey_Lidar_Inde
x.shp'
      using driver `ESRI Shapefile' successful.

Layer name: 312017118_GPSC3_New_Jersey_Lidar_Index
Metadata:
  DBF_DATE_LAST_UPDATE=2018-12-26
Geometry: Polygon
Feature Count: 2843
Extent: (295000.000000, 545000.000000) - (610000.000000, 925000.000000)
Layer SRS WKT:
PROJCRS["NAD83(2011) / New Jersey (ftUS)",
    BASEGEOGCRS["NAD83(2011)",
        DATUM["NAD83 (National Spatial Reference System 2011)",
            ELLIPSOID["GRS 1980",6378137,298.257222101,
                LENGTHUNIT["metre",1]]],
        PRIMEM["Greenwich",0,
            ANGLEUNIT["degree",0.0174532925199433]],
        ID["EPSG",6318]],
    CONVERSION["SPCS83 New Jersey zone (US Survey feet)",
        METHOD["Transverse Mercator",
            ID["EPSG",9807]],
        PARAMETER["Latitude of natural origin",38.8333333333333,
            ANGLEUNIT["degree",0.0174532925199433],
            ID["EPSG",8801]],
        PARAMETER["Longitude of natural origin",-74.5,
            ANGLEUNIT["degree",0.0174532925199433],
            ID["EPSG",8802]],
        PARAMETER["Scale factor at natural origin",0.9999,
            SCALEUNIT["unity",1],
            ID["EPSG",8805]],
        PARAMETER["False easting",492125,
            LENGTHUNIT["US survey foot",0.304800609601219],
            ID["EPSG",8806]],
        PARAMETER["False northing",0,
            LENGTHUNIT["US survey foot",0.304800609601219],
            ID["EPSG",8807]]],
    CS[Cartesian,2],
        AXIS["easting (X)",east,
            ORDER[1],
            LENGTHUNIT["US survey foot",0.304800609601219]],
        AXIS["northing (Y)",north,
            ORDER[2],
            LENGTHUNIT["US survey foot",0.304800609601219]],
    USAGE[
        SCOPE["Engineering survey, topographic mapping."],
        AREA["United States (USA) - New Jersey - counties of Atlantic; Bergen; Burlingto
n; Camden; Cape May; Cumberland; Essex; Gloucester; Hudson; Hunterdon; Mercer; Middlese
x; Monmouth; Morris; Ocean; Passaic; Salem; Somerset; Sussex; Union; Warren."],
        BBOX[38.87,-75.6,41.36,-73.88]],
    ID["EPSG",6527]]
Data axis to CRS axis mapping: 1,2
```

```
Name: String (254.0)
CentroidX: Real (18.9)
CentroidY: Real (18.9)
NumVerts: Integer64 (11.0)
EntArea: Real (18.9)
AreaUnit: String (254.0)
7118: String (10.0)
```

In [2]:

```python
import tiledb

array_uri = 'tiledb://TileDB-Inc/nj_nw_2018'

config_dict = {
    "py.init_buffer_bytes": 104857600
}

ctx = tiledb.Ctx(config_dict)
```

In [3]:

```python
with tiledb.open(array_uri) as arr:
    print(arr.schema)
    print(arr.nonempty_domain())
```

```
ArraySchema(
  domain=Domain(*[
    Dim(name='X', domain=(295000.0, 610000.0), tile='None', dtype='float64'),
    Dim(name='Y', domain=(545000.0, 925000.0), tile='None', dtype='float64'),
    Dim(name='Z', domain=(0.0, 5000.0), tile='None', dtype='float64'),
  ]),
  attrs=[
    Attr(name='Intensity', dtype='uint16', var=False, nullable=False, filters=FilterList
([Bzip2Filter(level=5), ])),
    Attr(name='ReturnNumber', dtype='uint8', var=False, nullable=False, filters=FilterLi
st([ZstdFilter(level=7), ])),
    Attr(name='NumberOfReturns', dtype='uint8', var=False, nullable=False, filters=Filte
rList([ZstdFilter(level=7), ])),
    Attr(name='ScanDirectionFlag', dtype='uint8', var=False, nullable=False, filters=Fil
terList([Bzip2Filter(level=5), ])),
    Attr(name='EdgeOfFlightLine', dtype='uint8', var=False, nullable=False, filters=Filt
erList([Bzip2Filter(level=5), ])),
    Attr(name='Classification', dtype='uint8', var=False, nullable=False, filters=Filter
List([GzipFilter(level=9), ])),
    Attr(name='ScanAngleRank', dtype='float32', var=False, nullable=False, filters=Filte
rList([Bzip2Filter(level=5), ])),
    Attr(name='UserData', dtype='uint8', var=False, nullable=False, filters=FilterList
([GzipFilter(level=9), ])),
    Attr(name='PointSourceId', dtype='uint16', var=False, nullable=False, filters=Filter
List([Bzip2Filter(level=-1), ])),
    Attr(name='GpsTime', dtype='float64', var=False, nullable=False, filters=FilterList
([ZstdFilter(level=7), ])),
    Attr(name='ScanChannel', dtype='uint8', var=False, nullable=False),
    Attr(name='ClassFlags', dtype='uint8', var=False, nullable=False),
  ],
  cell_order='hilbert',
  tile_order='NA',
  capacity=100000,
  sparse=True,
  allows_duplicates=True,
  coords_filters=FilterList([ZstdFilter(level=7)]),
)
```

```
((array(430000.), array(469999.999)), (array(560995.706), array(920205.119)), (array(13.
397), array(2399.015)))
```

In [4]:

```
%%time

minx = 445000.0
maxx = 449999.0
miny = 905000.0
maxy = 909999.999
minz = 1239.0
maxz = 2021.0
with tiledb.open(array_uri, ctx=ctx) as arr:
    df = arr.query(dims=['X', 'Y', 'Z'], attrs=[]).df[minx:maxx, miny:maxy, minz:maxz]
```

```
CPU times: user 3.09 s, sys: 1.79 s, total: 4.88 s
Wall time: 19.8 s
```

In [5]:

```
print(len(df['X']))
```

```
15187665
```

In [6]:

```
df
```

Out[6]:

	X	Y	Z
0	445000.117	907126.546	1303.673
1	445000.413	907134.641	1303.633
2	445000.509	907136.045	1303.632
3	445001.059	907140.195	1303.613
4	445007.251	907145.253	1303.672
...	...	...	...
15187660	449553.900	905006.844	1247.961
15187661	449555.864	905003.398	1248.061
15187662	449549.334	905005.641	1247.641
15187663	449554.105	905004.637	1247.731
15187664	449552.129	905008.064	1248.151

15187665 rows × 3 columns

In [7]:

```
import pybabylonjs

min_height = df['Z'].min()
max_height = df['Z'].max()
rng = max_height - min_height

interval = 50
data = {
    'X': df['X'][::interval],
```

```
    'Y': df['Y'][::interval],
    'Z': df['Z'][::interval],
    'Red': (df['Z'][::interval] - min_height) / rng,
    'Green': (df['Z'][::interval] - min_height) /rng,
    'Blue': (df['Z'][::interval] - min_height) / rng
}
```

In [8]:

```
babylon = pybabylonjs.BabylonJS()
babylon.value = data
babylon.z_scale = .25
babylon.width = 1000
babylon.height = 1000
```

In [9]:

```
babylon
```

In [10]:

```
# apply rgb values from a WMS service
```

In [11]:

```
!wget "https://img.nj.gov/imagerywms/Natural2019?request=getcapabilities&service=wms&ve
```

```
--2021-07-19 20:27:02--  https://img.nj.gov/imagerywms/Natural2019?request=getcapabiliti
es&service=wms&version=1.3
Resolving img.nj.gov (img.nj.gov)... 199.20.100.65
Connecting to img.nj.gov (img.nj.gov)|199.20.100.65|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 4016 (3.9K) [text/xml]
Saving to: ‘capabilities.xml’

capabilities.xml    100%[===================>]   3.92K  --.-KB/s    in 0s

2021-07-19 20:27:02 (330 MB/s) - ‘capabilities.xml’ saved [4016/4016]
```

In [12]:

```
import rasterio
from rasterio import MemoryFile
from rasterio.plot import show
from rasterio import logging
from urllib.request import urlopen

log = logging.getLogger()
log.setLevel(logging.ERROR)
```

In [13]:

```
w = 800
h = 800
get_map = f"https://img.nj.gov/imagerywms/Natural2017?SERVICE=WMS&VERSION=1.1.1&REQUEST
```

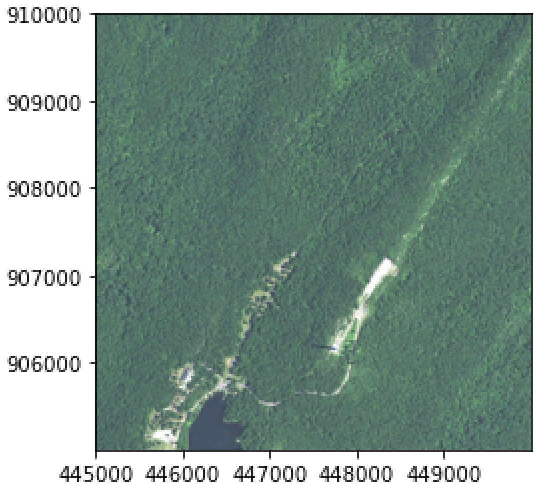
In [14]:

```
tif_bytes = urlopen(get_map % (minx, miny, maxx, maxy, w, h)).read()

with MemoryFile(tif_bytes) as memfile:
    with memfile.open() as src:
```

```
        print(src.profile)
        show(src)
```

{'driver': 'GTiff', 'dtype': 'uint8', 'nodata': 255.0, 'width': 800, 'height': 800, 'count': 3, 'crs': CRS.from_epsg(6527), 'transform': Affine(6.248750000000023, 0.0, 444999.88467548403,
       0.0, -6.249998749999964, 910002.518125334), 'tiled': False, 'interleave': 'pixel'}



In [15]:

```
# prototype
pdf = df[:10000]
```

Running this computation is too long and intensive for a single notebook server, we will prototype a function for the colorization of this point cloud

In [16]:

```
def compute_rgb(df_sample):
    import numpy as np
    import pandas as pd
    import rasterio
    from rasterio import MemoryFile
    import requests
    from requests.adapters import HTTPAdapter
    from requests.packages.urllib3.util.retry import Retry

    pd.options.mode.chained_assignment = None

    s = requests.Session()
    retries = Retry(total=3, backoff_factor=1, status_forcelist=[ 502, 503, 504 ])
    s.mount('https://', HTTPAdapter(max_retries=retries))

    # for tutorial
    if not isinstance(df_sample, pd.DataFrame):
        df_sample = pd.DataFrame.from_dict(df_sample)

    bounds = (df_sample['X'].min(), df_sample['Y'].min(), df_sample['X'].max(), df_samp

    get_map = f"https://img.nj.gov/imagerywms/Natural2017?SERVICE=WMS&VERSION=1.1.1&REQ

    def find_rgb(row):
        sample = next(src.sample([(row['X'], row['Y'])]))
        return pd.Series(sample[0:3])
```

```
    resp = s.get(get_map % bounds)
    if resp.status_code == 200:
        tif_bytes = resp.content
        with MemoryFile(tif_bytes) as memfile:
            with memfile.open() as src:
                df_sample[['Red', 'Green', 'Blue']] = df_sample.apply(find_rgb, axis=1)
                return df_sample
    else:
        # for the tutorial if we are bounced upstream we will use a cached tiledb array
        cache_uri = 'tiledb://TileDB-Inc/nj_cache_array'
        with rasterio.open(cache_uri) as src:
            df_sample[['Red', 'Green', 'Blue']] = df_sample.apply(find_rgb, axis=1)
            return df_sample
```

In [17]:

```
%%time
rdf = compute_rgb(pdf)
rdf
```

```
CPU times: user 2.36 s, sys: 0 ns, total: 2.36 s
Wall time: 2.77 s
```

Out[17]:

	X	Y	Z	Red	Green	Blue
0	445000.117	907126.546	1303.673	87.0	125.0	99.0
1	445000.413	907134.641	1303.633	100.0	148.0	109.0
2	445000.509	907136.045	1303.632	104.0	154.0	112.0
3	445001.059	907140.195	1303.613	108.0	165.0	114.0
4	445007.251	907145.253	1303.672	112.0	165.0	120.0
...	...	...	...	...	...	...
9995	445074.543	907011.147	1304.590	99.0	138.0	116.0
9996	445074.881	907013.353	1304.551	101.0	141.0	118.0
9997	445076.616	907014.369	1304.551	104.0	145.0	119.0
9998	445075.212	907017.973	1304.600	99.0	138.0	114.0
9999	445075.335	907015.101	1304.592	100.0	140.0	116.0

10000 rows × 6 columns

In [18]:

```
data = {
    'X': rdf['X'],
    'Y': rdf['Y'],
    'Z': rdf['Z'],
    'Red': rdf['Red'] / 255.0,
    'Green': rdf['Green'] / 255.0,
    'Blue': rdf['Blue'] / 255.0
}
```

In [19]:

```
babylon = pybabylonjs.BabylonJS()
```

```
babylon.value = data
babylon.width = 1000
babylon.height = 1000
babylon.z_scale = .25
```

In [20]:

```
babylon
```

Create a task graph

In [21]:

```
df_bounds = (df['X'].min(), df['Y'].min(), df['X'].max(), df['Y'].max())
df_bounds
```

Out[21]:

```
(445000.0, 905000.0, 449999.0, 909999.998)
```

In [22]:

```
from tiledb.cloud.compute import DelayedArrayUDF, Delayed

nodes = []

y = df_bounds[1]
step = 500.0

while y < df_bounds[3]:
    x = df_bounds[0]
    while x < df_bounds[2]:
        name = "node_{}_{}".format(y, x)
        nodes.append(
            DelayedArrayUDF(array_uri,
                            compute_rgb,
                            attrs=['X', 'Y', 'Z'], image_name='3.7-geo', name=name)([(x, x
        x = x + step
    y = y + step

print(len(nodes))
nodes
```

```
100
```

Out[22]:

```
[<tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734d1ce10>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734a6a450>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734d56490>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734dc04d0>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734dc00d0>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734d1c9d0>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734d1c590>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734d1c7d0>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734d1cbd0>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734d56c90>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734d1cdd0>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734d1c890>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734d1c550>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734d1cd90>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734d1cad0>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734d1cb50>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734d1c510>,
```

```
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff737bd10d0>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff737bd1ed0>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff737bd1090>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff7372ccf90>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff7372ccd10>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff7372ccf50>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff7372ccf10>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff7372cccd0>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c5be50>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c5bfd0>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c5b850>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c5b510>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c5bdd0>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c5b550>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c5ba10>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c5b3d0>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c5be10>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff5e411a1d0>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff5e411a210>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff7349f4310>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734cf7dd0>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734cf7910>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734cf7c50>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734cf7850>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734cf7790>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734d56cd0>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734cf7cd0>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c5b450>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c5b110>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c5bd90>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c5b250>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c5b910>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c5b2d0>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c5b490>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c5b6d0>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c5b690>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c5b410>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c5b290>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c5bb50>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c5bb10>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c5bd50>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c5bf90>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c5bd10>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c5ba90>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff73738ddd0>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c5b310>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c6bd10>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c6b550>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c6ba50>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c6bfd0>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c6be50>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c6bf10>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c6b990>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c6ba90>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c6b190>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c6b310>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c6bf50>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c6b0d0>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c6b610>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c6bdd0>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c6b510>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c6b4d0>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c6b150>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c6b390>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c6bc50>,
```

```
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c6ba10>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c6b490>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c6b110>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c6be90>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734c6bc10>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734d54c10>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734d54910>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734d544d0>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734d54c50>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734d54f90>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734dc1e50>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734dc1810>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff736faf290>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff736ee5f90>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff7374793d0>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff737479110>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff737483750>,
 <tiledb.cloud.compute.delayed.DelayedArrayUDF at 0x7ff734dc1f10>]
```

In [23]:

```
def concat(results):
    import pandas as pd
    return pd.concat(results)
```

In [24]:

```
res = Delayed(concat, local=True, name="colorize")(nodes)
```

In [25]:

```
res.visualize()
```

In [26]:

```
%%time
result_df = res.compute()
result_df
```

```
CPU times: user 35.7 s, sys: 5.2 s, total: 40.9 s
Wall time: 1min 44s
```

Out[26]:

	X	Y	Z	Red	Green	Blue
0	445000.988	905495.313	1388.691	81.0	110.0	105.0
1	445001.575	905495.325	1388.720	81.0	110.0	105.0
2	445001.509	905490.050	1388.967	75.0	101.0	99.0
3	445000.297	905491.533	1389.001	78.0	104.0	102.0
4	445001.004	905491.455	1388.990	77.0	103.0	101.0
...	...	...	...	...	...	...
123199	449721.739	909526.695	1725.809	80.0	109.0	102.0
123200	449722.035	909522.411	1725.600	78.0	108.0	102.0
123201	449711.786	909532.102	1725.850	64.0	81.0	90.0
123202	449710.010	909536.516	1725.797	60.0	79.0	84.0
123203	449817.381	909566.168	1739.414	87.0	120.0	107.0

15190984 rows × 6 columns

In [27]:

```
data = {
    'X': result_df['X'][::interval],
    'Y': result_df['Y'][::interval],
    'Z': result_df['Z'][::interval],
    'Red': result_df['Red'][::interval] / 255.0,
    'Green':result_df['Green'][::interval] / 255.0,
    'Blue': result_df['Blue'][::interval] / 255.0
}
```

In [28]:

```
babylon = pybabylonjs.BabylonJS()
babylon.value = data
babylon.z_scale = .25
babylon.width = 1000
babylon.height = 1000
```

In [29]:

```
babylon
```

In [ ]:

```

```