n3gb - The NUAR hex-based spatial indexing system

The n3gb indexing system was inspired by the h3 spatial index originally developed by Uber and now open sourced. The NUAR project is working on a Generalised Data API to enable statisitics and information about buried asset data without the need to divulge the exact details and locations of said assets. While H3 provides the ideal structure for aggregating this data its global nature means that it is based on WGS84 and therefore not ideal for use in a NUAR context. Whilst we havbe created endpoints in our API to make use of H3 we decided to create our own variant that is built for the British National Grid.

In devising our approach to a spatial indexing system we based it on the following principles:

- All hex cells must have a unque identifier
- The unique identifier must be stable and consistent
- The hex cell must be algorithmically generated
- The hex cell must be different sizes at different zoom levels, with the size appropriate for analysis at that zoom level
- The hex cell areas should be approximately similar to the H3 grid
- It should be possible to determine the next child(s) hexagon from a given hexagon
- It should be possible to determine the parent hexagon from a given hxagon

This notebook provides an interactive appraisal of the algorithms used to generate and identify hex cells using the n3gb system.

In [3]:

```python
import math
import base64
import struct
from shapely import Polygon
```

Create the functions that can be used to calculate and generate hex grids.

In [17]:

```python
# Set some constants that define the grid extent and the widths of hexagons at e
GRID_EXTENTS = [0, 0, 750000, 1350000]
CELL_RADIUS = [1281249.9438829257, 48304.58762201923, 182509.65769514776, 68979.
CELL_WIDTHS = [2219190, 83666, 316116, 119476, 45154, 17060, 6443, 2424, 917, 34

def generate_identifier(easting, northing, zoom_level):
    # Create the identifier string that will be encoded
    identifier = f"E{easting}N{northing}Z{zoom_level}"

    # Multiply easting and northing to preserve 4 decimal places and convert to
    easting_int = int(round(easting * 10000))
    northing_int = int(round(northing * 10000))

    # Pack the data into a binary representation (each as 8 bytes = 64 bits for
    binary_date = struct.pack(">QQB", easting_int, northing_int, zoom_level)
    # Explanation:
    # > : big-endian byte order
    # Q : unsigned long long (8 bytes) for easting
    # Q : unsigned long long (8 bytes) for northing
    # B : unsigned char (1 byte) for zoom level

    # Encode to URL-safe Base64 and remove any padding for compactness
    base64_encoded = base64.urlsafe_b64encode(binary_date).rstrip(b"=")

    return base64_encoded.decode("ascii")

def decode_hex_identifier(identifier):
    # Add padding back if needed for base64 decoding
    padding = '=' * (-len(identifier) % 4)
    base64_str = identifier + padding

    # Decode the base64 string to binary
    binary_data = base64.urlsafe_b64decode(base64_str)

    # Unpack the binary data to get easting, northing, and zoom level
    easting_int, northing_int, zoom_level = struct.unpack(">QQB", binary_data)

    # Convert back to original easting and northing values
    easting = easting_int / 10000.0
    northing = northing_int / 10000.0

    # Retrun the decoded values
    return easting, northing, zoom_level

def create_hexagon(center_x, center_y, size):
    """
    Create a hexagon polygon centered at (center_x, center_y) with the given siz

    Args:
        center_x (float): X coordinate of the center.
        center_y (float): Y coordinate of the center.
        size (float): Size of the hexagon.

    Returns:
        shapely.geometry.Polygon: Hexagon polygon.
    """
    points = [
        (
            center_x + size * math.cos(math.radians(angle)),
            center_y + size * math.sin(math.radians(angle))
        )
        for angle in range(30, 390, 60)
    ]
    return Polygon(points)

def point_to_hex(x, y, z):
    """
    Convert a point (x, y) to the corresponding hexagon index in a grid.

    Args:
        x (float): X coordinate of the point.
        y (float): Y coordinate of the point.
        z (int): Zoom level.

    Returns:
        tuple: Row and column indices of the hexagon containing the point.
    """
    hex_width = CELL_WIDTHS[z]  # Width of the hexagon at the given zoom level
    r = CELL_RADIUS[z]  # Radius of the hexagon at the given zoom level
    dx = hex_width  # Horizontal distance between hexagon centers
    dy = 1.5 * r  # Vertical distance between hexagon centers

    qx = (x - GRID_EXTENTS[0]) / dx
    ry = (y - GRID_EXTENTS[1]) / dy

    row = int(round(ry))
    col = int(round(qx - (row % 2)))  # Adjust for odd/even rows

    return row, col

def hex_to_point(row, col, z):
    """
    Convert hexagon indices (row, col) to the corresponding point (x, y).

    Args:
        row (int): Row index of the hexagon.
        col (int): Column index of the hexagon.
        z (int): Zoom level.

    Returns:
        tuple: X and Y coordinates of the center of the hexagon.
    """
    hex_width = CELL_WIDTHS[z]  # Width of the hexagon at the given zoom level
    r = CELL_RADIUS[z]  # Radius of the hexagon at the given zoom level
    dx = hex_width  # Horizontal distance between hexagon centers
    dy = 1.5 * r  # Vertical distance between hexagon centers

    x = GRID_EXTENTS[0] + col * dx + ((row % 2) * (dx / 2))
    y = GRID_EXTENTS[1] + row * dy

    return x, y

def get_hexes_for_extent(extents, zoom_level):
    """
    Returns all hexagons that intersect the given extents at the zoom level spec

    Args:
        extents (list): The extents to intersect, expressed as [minx, miny, maxx
        zoom_level (int): The zoom level to calculate intersections at

    Returns:
        list: A list of hexagons that intersect the extents at the zoom level
    """

    # Get the grid references at each corner of the extent
    ll_row, ll_col = point_to_hex(extents[0], extents[1], zoom_level)
    lr_row, lr_col = point_to_hex(extents[2], extents[1], zoom_level)
    ur_row, ur_col = point_to_hex(extents[2], extents[3], zoom_level)
    ul_row, ul_col = point_to_hex(extents[0], extents[3], zoom_level)

    # Get the minimum and maximum row and column values
    max_row = max(ll_row, lr_row, ur_row, ul_row)
    max_col = max(ll_col, lr_col, ur_col, ul_col)
    min_row = min(ll_row, lr_row, ur_row, ul_row)
    min_col = min(ll_col, lr_col, ur_col, ul_col)
    # Create an empty list for the hexagons to be populated
    hexagons = []

    # Loop over the calculated rows and columns to determine all hexagons
    for row in range(min_row, max_row + 1):
        for col in range(min_col, max_col + 1):
            # Get the x and y centroid of the hexagon
            x, y = hex_to_point(row, col, zoom_level)

            # Create the hexagon geometry
            hexagon = create_hexagon(x, y, CELL_RADIUS[zoom_level])

            # Define the attributes for the hexagon
            hexagons.append({
                "id": generate_identifier(x, y, zoom_level),
                "easting": x,
                "northing": y,
                "zoom_level": zoom_level,
                "row": row,
                "col": col,
                "geom": hexagon
            })

    # Return the intersected hexagons
    return hexagons
```

Generating and using Hex Cells

We can use the functions above to calculate and generate hex grid cells, for example:

- Provide easting, northing, zoom level and get the intersecting hex grid cell
- Provide a bounding box and zoom level and return all hex grid cells that intersect the bounds
- Provide a hex grid cell identifier and have it decoded
- Create a hexagon given the centre point (easting, northing) and the zoom level

Encoding and decoding hex grid identifiers

This example shows how coordinates (easting and northing) and a zoom level can be encoded to the unique identifier used by the grid system. Then how that identifier can be decoded in order to get the corresponding easting, northing, and zoom level.

In [7]:
```
# Encode easting, northing, and zoom level to a hex grid cell
identifier = generate_identifier(252086.1234, 847702.1234, 10)
print(f"Generated hex identifier: {identifier}")

# Decode the identifier back to easting, northing, and zoom level
easting, northing, zoom_level = decode_hex_identifier(identifier)
print(f"Decoded hex identifier: E{easting} N{northing} Z{zoom_level}")
```

```
Generated hex identifier: AAAAAJZBSjIAAAAB-UUUMgo
Decoded hex identifier: E252086.1234 N847702.1234 Z10
```

Generate the Hex Cells that intersect a bounding box

This example shows how a bounding box extent (minx, miny, maxx, maxy) can be used to generate all hex grid cells that intersect the bounding box at a given zoom level. The bounding box is a sample area of data in Nottingham.

In [28]:
```
import geopandas as gpd

# Get hexagons that intersect the given bounding box at zoom level 10
hexagons = get_hexes_for_extent([457000, 339500, 458000, 340500], 10)

# Convert the hexagons to a geo data frame
gdf_hexagons = gpd.GeoDataFrame(hexagons, geometry='geom', crs='EPSG:27700')

# Show the geo dataframe table
gdf_hexagons
```

Out[28]:

	id	easting	northing	zoom_level	row	col	
0	AAAAARBdCWAAAAAAymNoUwo	456950.0	339551.240316	10	3016	3515	3
1	AAAAARBw34AAAAAAymNoUwo	457080.0	339551.240316	10	3016	3516	3
2	AAAAARCEtaAAAAAAymNoUwo	457210.0	339551.240316	10	3016	3517	3
3	AAAAARCYi8AAAAAAymNoUwo	457340.0	339551.240316	10	3016	3518	3
4	AAAAARCsYeAAAAAAymNoUwo	457470.0	339551.240316	10	3016	3519	3
...	...	...	...	...	...	...	
76	AAAAARCsYeAAAAAAyuzWmwo	457470.0	340451.906736	10	3024	3519	3
77	AAAAARDAOAAAAAAAyuzWmwo	457600.0	340451.906736	10	3024	3520	3
78	AAAAARDUDiAAAAAAyuzWmwo	457730.0	340451.906736	10	3024	3521	3
79	AAAAARDn5EAAAAAAyuzWmwo	457860.0	340451.906736	10	3024	3522	3
80	AAAAARD7umAAAAAAyuzWmwo	457990.0	340451.906736	10	3024	3523	

id	easting	northing	zoom_level	row	col
					3

81 rows × 7 columns

Identify Hex Grid Cell a point intersects

This example shows how the grid algorithms can be used to find what hex grid cell a given point location is within, this can be useful for example to identify what hex grid cell a point feature is within. These could then be aggregated up to provide a feature count per hex grid cell.

First we'll simply provide a point location and then find the identifier of the hex grid cell at zoom level 8 the point falls within.

In [25]:
```
# Set the easting, northing, and zoom level
easting = 457996
northing = 339874
zoom_level = 10

# Find what hex grid cell the point intersects at the zoom level
hex_row, hex_col = point_to_hex(easting, northing, zoom_level)

# Now we have the cell row and column we can get the centre point of the hexagon
hex_cell_x, hex_cell_y = hex_to_point(hex_row, hex_col, zoom_level)

# We can now use the hexagon's centre point to create a hexagon geometry
hexagon = create_hexagon(hex_cell_x, hex_cell_y, CELL_RADIUS[zoom_level])

print(f"The point [{easting}, {northing}] at zoom level [{zoom_level}] is within
print("***")
print(f"The hex grid cell [{hex_row}, {hex_col}] has a centre point of [{hex_cel
print("***")
print(f"The hexagon geometry for the cell [{hex_row}, {hex_col}] is: {hexagon.wk
```

```
The point [457996, 339874] at zoom level [10] is within hex grid cell [3019, 352
2]
***
The hex grid cell [3019, 3522] has a centre point of [457925.0, 339888.990223278
7] at zoom level [10]
***
The hexagon geometry for the cell [3019, 3522] is: POLYGON ((457990 339926.517990
77605, 457925 339964.04575827334, 457860 339926.51799077605, 457860 339851.462455
78135, 457925 339813.93468828406, 457990 339851.46245578135, 457990 339926.517990
77605))
```

Putting it all on a map

A picture paints a thousand words, or in this case makes it prettier to look at than a table!

We are going to take the hexagons calculated with our bounding box and plot them to a map with an OS Basemap and the hex grid overlayed. The hex cells will be red outlined with a semi-transparent red fill and we shall include the bounding box we used to generate the hex cells as a blue polygon.

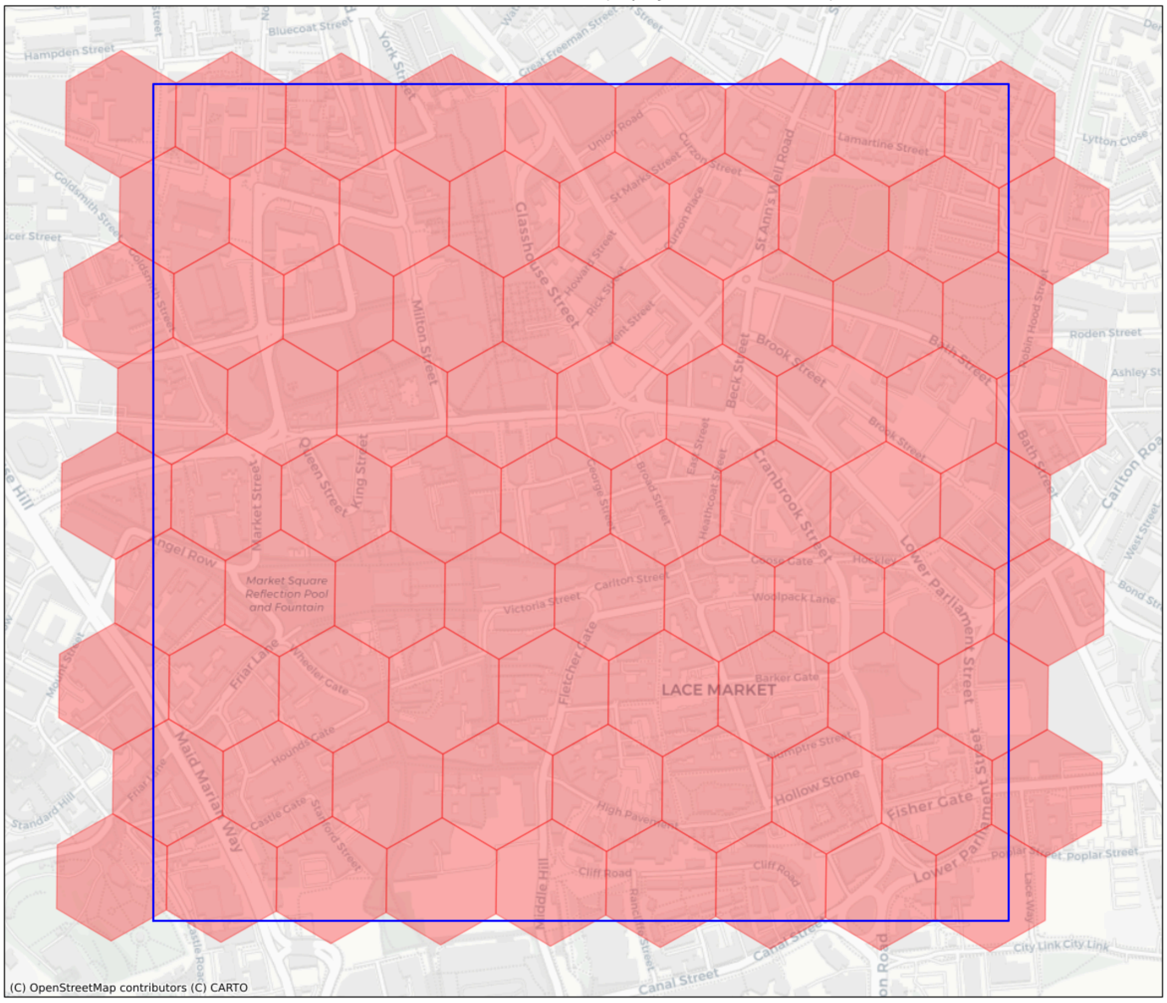
In [45]:
```python
import contextily as ctx
import xyzservices.providers as xyz
import cartopy.crs as ccrs
import matplotlib.pyplot as plt
from pyproj import Transformer

# Plot the GeoDataFrame with a basemap, we need to transform the GeoDataFrame to
fig, ax = plt.subplots(1, 1, figsize=(15, 15), subplot_kw={'projection': ccrs.Me
hexagons_3857 = gdf_hexagons.to_crs(epsg=3857)
hexagons_3857.plot(ax=ax, edgecolor='red', facecolor='red', alpha=0.3)
ctx.add_basemap(ax, crs="EPSG:3857", source=xyz.CartoDB.Positron)

# Add the bounding box of the area of interest to the plot
minx, miny, maxx, maxy = 457000, 339500, 458000, 340500
transformer = Transformer.from_crs("EPSG:27700", "EPSG:3857", always_xy=True)
bbox_minx, bbox_miny = transformer.transform(minx, miny)
bbox_maxx, bbox_maxy = transformer.transform(maxx, maxy)
plt.plot(
    [bbox_minx, bbox_maxx, bbox_maxx, bbox_minx, bbox_minx],
    [bbox_miny, bbox_miny, bbox_maxy, bbox_maxy, bbox_miny],
    'b-'
)

# Show the plot
plt.title("Hex Grid Cells at Zoom Level 10 (Reprojected to EPSG:3857)")
plt.show()
```

Hex Grid Cells at Zoom Level 10 (Reprojected to EPSG:3857)



LACE MARKET
Market Square Reflection Pool and Fountain
(C) OpenStreetMap contributors (C) CARTO