geometry.py – Geometric Calculations & Spatial Operations

Overview

This module provides geospatial utility functions for converting between real-world measurements and pixel coordinates. It handles buffer zone calculations and spatial panel selection.

Logic

Functions Overview

Function	Purpose
get_meters_per_pixel()	Calculate ground resolution at given location
buffer_radius_to_pixels()	Convert buffer area (sq.ft) to pixel radius
find_best_panel()	Select optimal panel within buffer zone
encode_polygon()	Convert polygon to JSON string

How It Works

1. Meters Per Pixel Calculation

```
def get_meters_per_pixel(lat, zoom, scale):
    return 156543.03392 * math.cos(math.radians(lat)) / (2 ** zoom) / scale
```

The Formula Breakdown:

Component	Value	Meaning
156543.03392	constant	Earth's circumference / 256 (tile size at zoom 0)
cos(lat)	0-1	Latitude correction (Mercator distortion)
2 ** zoom	2^20	Zoom level scaling
/ scale	÷2	High-DPI scale factor

Example at zoom=20, scale=2, lat=23°:

```
156543.03392 × cos(23°) × (1/1048576) × (1/2) ≈ 0.069 m/px
```

2. Buffer Radius Conversion

```
┌──────────────────────────────────────────────────────────────────────────────┐
│  Square Feet → Square Meters → Circle Radius → Pixels  │
└──────────────────────────────────────────────────────────────────────────────┘
```

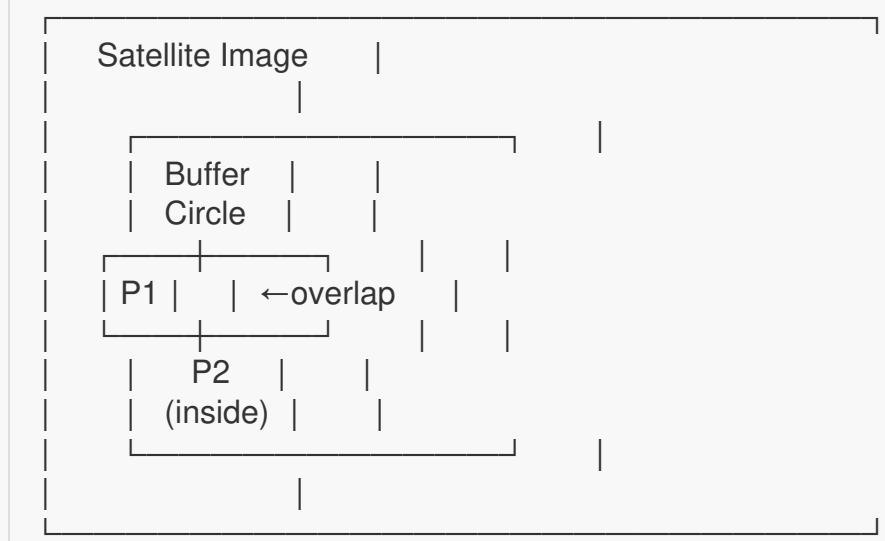
```python
def buffer_radius_to_pixels(buffer_sqft, sqft_to_sqm, lat, zoom, scale):
    # Convert sq.ft to equivalent circle radius in meters
    radius_meters = math.sqrt(buffer_sqft * sqft_to_sqm / math.pi)

    # Convert meters to pixels
    meters_per_px = get_meters_per_pixel(lat, zoom, scale)
    return radius_meters / meters_per_px
```

Example for 1200 sq.ft buffer:

```
1200 sq.ft × 0.092903 = 111.48 sq.m
radius = √(111.48 / π) = 5.96 meters
5.96 m ÷ 0.069 m/px ≈ 86 pixels
```

3. Best Panel Selection



```python
def find_best_panel(polygons, center_px, buffer_radius_px, min_overlap=10):
    center_point = Point(center_px)
    buffer_circle = center_point.buffer(buffer_radius_px)

    for poly, conf in polygons:
        if buffer_circle.intersects(poly):
            intersection = buffer_circle.intersection(poly)
            overlap_area = intersection.area

            if overlap_area > max_overlap:
                best_poly = poly
                max_overlap = overlap_area

    return best_poly, best_conf, max_overlap
```

Selection Criteria:

1. Panel must intersect buffer circle
2. Overlap area must exceed min_overlap (10 pixels)
3. Panel with **maximum overlap** wins

4. Polygon Encoding

```python
def encode_polygon(poly):
    if poly is None:
        return ""
    coords = list(poly.exterior.coords)
    return json.dumps([[round(x, 2), round(y, 2)] for x, y in coords])
```

Output format:

```
[[512.00, 480.00], [580.00, 480.00], [580.00, 540.00], [512.00, 540.00], [512.00, 480.00]]
```

Why It Works

Web Mercator Math

The constant `156543.03392` comes from:

```
Earth circumference (meters) / pixels at zoom 0
40075016.686 m / 256 px = 156543.03392 m/px
```

Each zoom level halves the resolution, hence `2 ** zoom`.

Latitude Correction

The Mercator projection stretches landmasses at high latitudes. `cos(lat)` compensates:

- Equator (lat=0°): cos(0°) = 1.0 → no correction
- Poles (lat=90°): cos(90°) = 0.0 → maximum stretch

Circle Buffer from Area

Buffer zones are defined in square feet (property boundaries), but spatial matching uses circular regions:

```
Area = π × r²  →  r = √(Area / π)
```

This creates an equivalent circular search zone around the property center.

Minimum Overlap Filter

Small incidental touching (< 10 pixels) is rejected as noise. This prevents:

- Edge artifacts
- Tiny overlaps from neighboring properties
- False positives from detection edge cases

Usage in Main Pipeline

Resolution Calculation

```
# In pipeline.py _process_sample()
meters_per_px = get_meters_per_pixel(lat, Config.ZOOM_LEVEL, Config.MAP_SCALE
```

Buffer Zone Setup

```
radius_1_px = buffer_radius_to_pixels(
    Config.BUFFER_RADIUS_1_SQFT,  # 1200 sq.ft
    Config.SQFT_TO_SQM,           # 0.092903
    lat, Config.ZOOM_LEVEL, Config.MAP_SCALE
)

radius_2_px = buffer_radius_to_pixels(
    Config.BUFFER_RADIUS_2_SQFT,  # 2400 sq.ft
    ...
)
```

Detection Matching

```
# Try smaller buffer first (higher confidence)
poly_1200, conf_1200, overlap = find_best_panel(
    all_polygons, center_px, radius_1_px, Config.MIN_OVERLAP_AREA
)

if poly_1200 is None:
    # Fall back to larger buffer
    poly_2400, conf_2400, overlap = find_best_panel(
        all_polygons, center_px, radius_2_px, Config.MIN_OVERLAP_AREA
    )
```

Result Encoding

```
output_record = {
    ...
    "bbox_or_mask": encode_polygon(result_data['polygon']),
    ...
}
```

Complete Pipeline Flow

```
Coordinates (lat/lon)
        |
        ▼
get_meters_per_pixel() → Resolution context
        |
        ▼
buffer_radius_to_pixels() → Search areas (1200/2400 sq.ft)
        |
        ▼
find_best_panel() → Selected panel + confidence
        |
        ▼
encode_polygon() → JSON for output
```