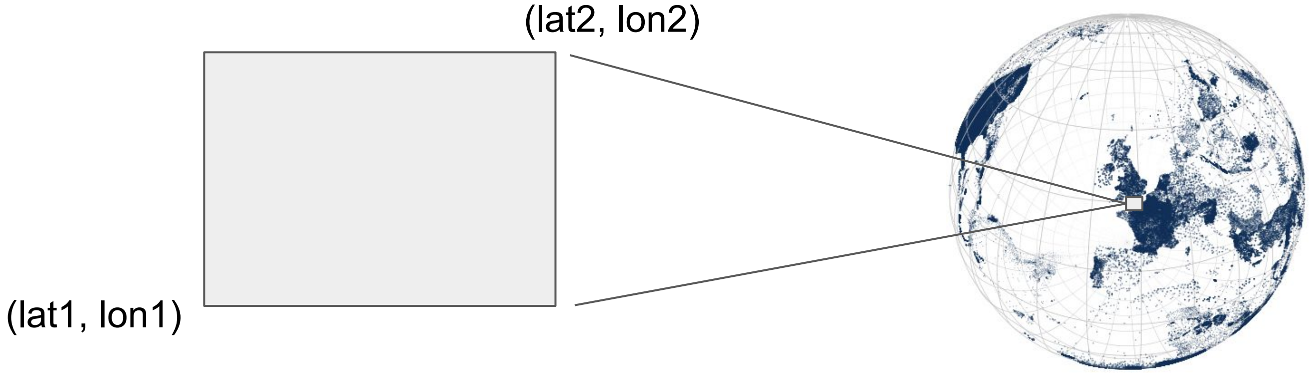
Aerial/Satellite Imagery Retrieval

Feiyu Chen, Zunran Guo

Introduction

Geospatial data has become ever more important in our lives for its ever growing economic value as companies like HERE Technologies are pushing high definition maps to cover the entire globe digitally. One of the most straight sources of high definition maps is from aerial/satellite imagery.

A common question raised in this realm is: Given a lat-lon bounding box, how can we retrieve a aerial/satellite image?



Overview

In this assignment, we leveraged Bing Maps' Tile System to retrieve a satellite imagery given a lat-lon bounding box.

Bing Maps is based on Mercator projection which are conformal and cylindrical:

1. Conformal: preserves the shape of relatively small objects
2. Cylindrical: north and south are always straight up and down, and west and east are always straight left and right

Bing Maps also follows a level system. At the lowest level of detail (Level 1), the map is 512 x 512 pixels. At each successive level of detail, the map width and height grow by a factor of 2: Level 2 is 1024 x 1024 pixels, Level 3 is 2048 x 2048 pixels, Level 4 is 4096 x 4096 pixels, and so on until **Level 23** (highest level).

Map Size and Ground Resolution

Map size is defined as:

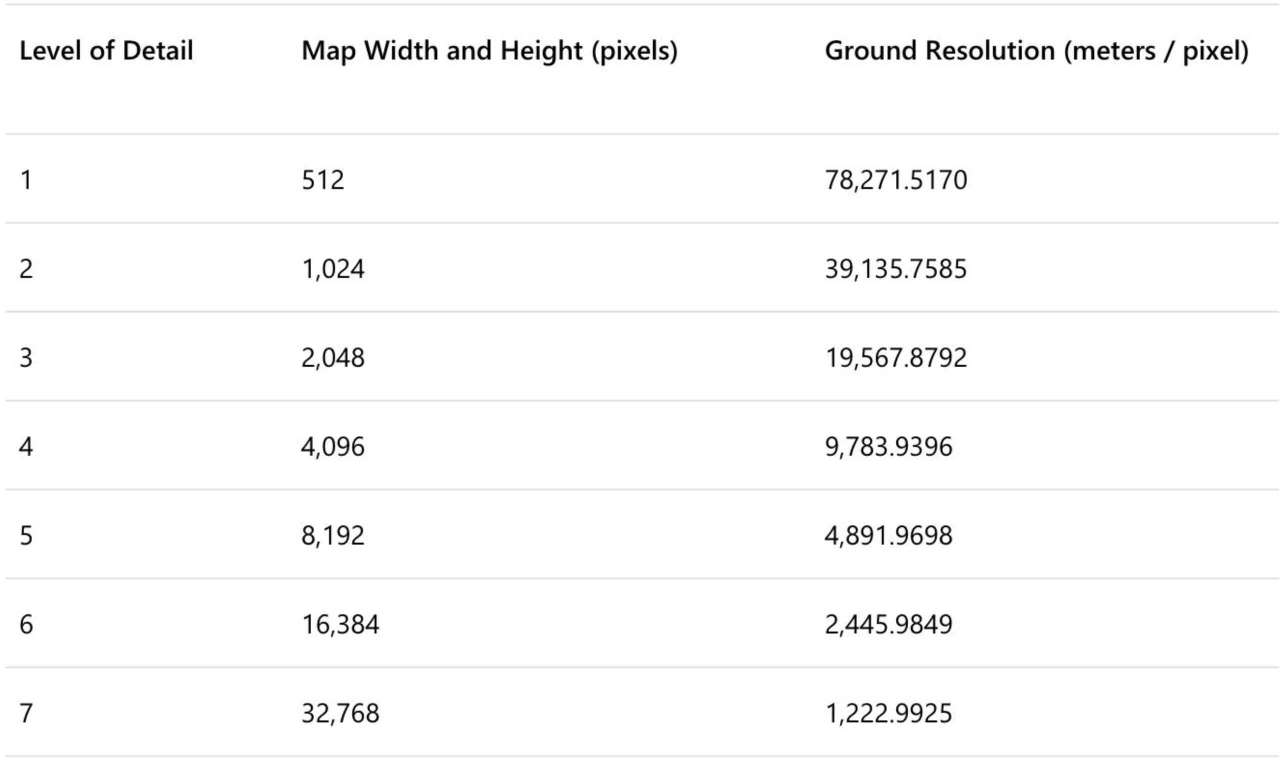
map_size = map width = map height = 256 * 2 ^{level} pixels

Ground resolution can be calculated as:

```
ground resolution
= cos(latitude * pi/180) * earth circumference / map width
= (cos(latitude * pi/180) * 2 * pi * 6378137 meters) / (256 * 2
level pixels)
```

See the code: *def ground_resolution*

Map Size and Ground Resolution



Level of Detail	Map Width and Height (pixels)	Ground Resolution (meters / pixel)
1	512	78,271.5170
2	1,024	39,135.7585
3	2,048	19,567.8792
4	4,096	9,783.9396
5	8,192	4,891.9698
6	16,384	2,445.9849
7	32,768	1,222.9925

Pixel Coordinates

Having chosen the projection to use at each level of detail, we can convert geographic coordinates into pixel coordinates. Given latitude and longitude in degrees, and the level of detail, the pixel XY coordinates can be calculated as follows:

sinLatitude = sin(latitude * pi/180)

pixelX = ((longitude + 180) / 360) * 256 * 2 ^{level}

pixelY = (0.5 – log((1 + sinLatitude) / (1 – sinLatitude)) / (4 * pi)) * 256 * 2 ^{level}

See the code: *def lat_long_to_tile_x_y*

Tile Coordinates

To optimize the performance of map retrieval and display, the rendered map is cut into tiles of 256 x 256 pixels each. As the number of pixels differs at each level of detail, so does the number of tiles:

map width = map height = 2^{level} tiles

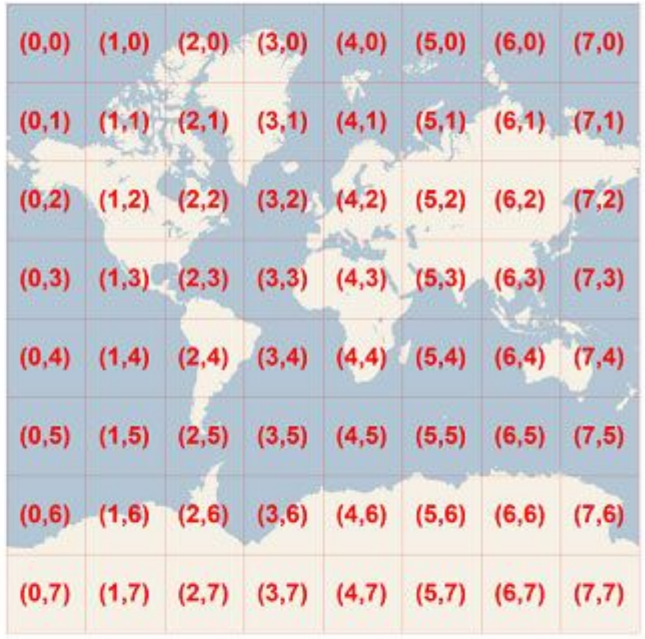
Given a pair of pixel XY coordinates, you can easily determine the tile XY coordinates of the tile containing that pixel:

```
tileX = floor(pixelX / 256)
tileY = floor(pixelY / 256)
```

See the code: *def lat_long_to_tile_x_y*

Tile Coordinates

For example, at level 3 the tile coordinates range from (0, 0) to (7, 7) as follows:



Quadkeys

To optimize the indexing and storage of tiles, the two-dimensional tile XY coordinates are combined into one-dimensional strings called quadtree keys, or “quadkeys” for short. Each quadkey uniquely identifies a single tile at a particular level of detail.

To convert tile coordinates into a quadkey, the bits of the Y and X coordinates are interleaved, and the result is interpreted as a base-4 number (with leading zeros maintained) and converted into a string.

For instance, given tile XY coordinates of (3, 5) at level 3, the quadkey is determined as follows:

- **`tileX`** `= 3 = 011 (binary)`
- **`tileY`** `= 5 = 101 (binary)`
- **`quadkey`** `= 100111 = 213 (base-4)`

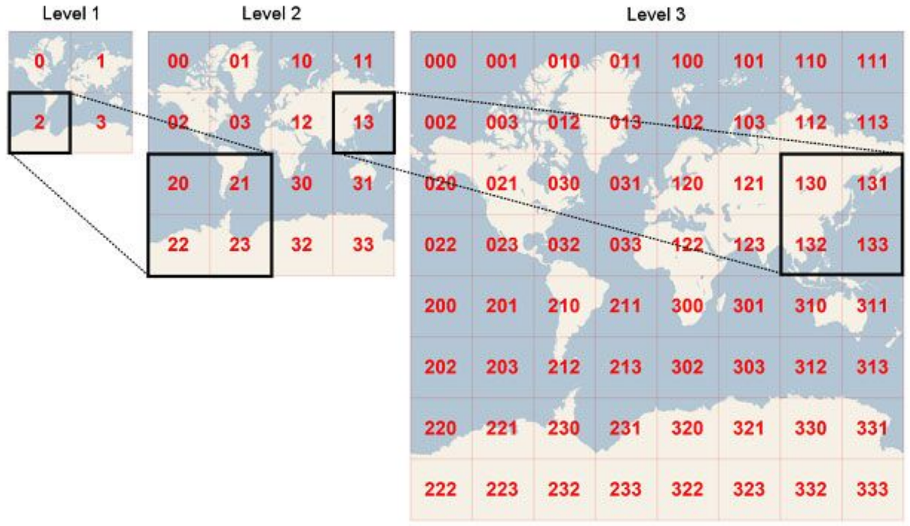
See the code: *def lat_long_to_tile_x_y, def tile_x_y_to_quadkey,* and *def lat_lon_to_quadkey*

Quadkeys

Quadkeys have several interesting properties:

1. The length of a quadkey (the number of digits) equals the level of detail of the corresponding tile.
2. The quadkey of any tile starts with the quadkey of its parent tile (the containing tile at the previous level).

As shown in the example to the right, tile 2 is the parent of tiles 20 through 23, and tile 13 is the parent of tiles 130 through 133:

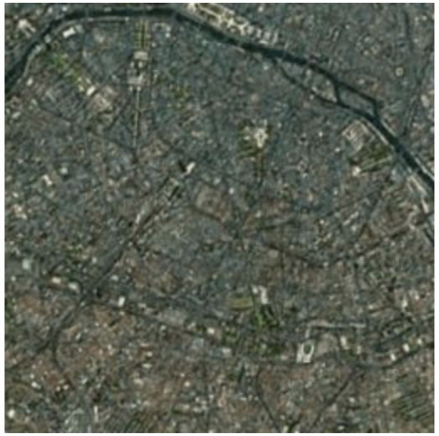


Quadkeys and Imagery Retrieval

Given a lat-lon bounding box, we identify the two pair of coordinates: (lat1, lon1) and (lat2, lon2). We then loop through levels from the smallest **until the maximum level is reached** so that the two pairs are still in the same tile coded by a quadkey.

Once we find the maximum level of detail, we can conveniently calculate the quadkey and retrieve the tile imagery via Bing Maps' API in meta format.

```
level of detail: 12
quadkey: 120220011012
ground resolution (meters/pixel) 25.1450099326402
```



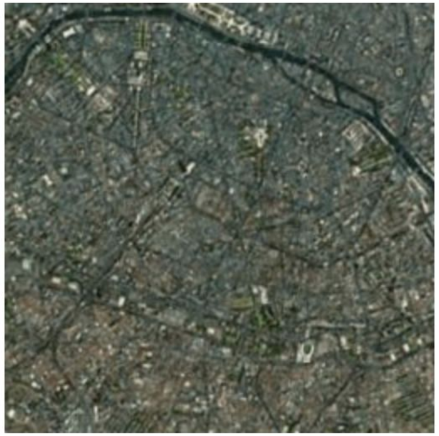
See the code: def *find_max_level* and *def retrieve_satellite_image_from_quadkey*

Quadkeys and Imagery Retrieval

Code:

```
lat_1 = 48.8580 # coordinates of Eiffel Tower
lon_1 = 2.2945

lat_2 = 48.8530 # coordinates of Cathédrale Notre-Dame de Paris
lon_2 = 2.3499

img, quadkey, level = retrieve_satellite_image_from_quadkey(
   lat_1, lon_1, lat_2, lon_2)

print('level of detail:', level)
print('quadkey:', quadkey)
print('ground resolution (meters/pixel)', ground_resolution(lat_1,
level))

# show image
Image(img)
```

```
level of detail: 12
quadkey: 120220011012
ground resolution (meters/pixel) 25.1450099326402
```



Alternative Imagery Retrieval Method

Besides using quadkey to retrieve imagery using Bing Maps’ Tile System, we also find a much more straightforward way to retrieve imagery by just passing the lat-long bounding box in an API request:

```
request_url =
"https://dev.virtualearth.net/REST/v1/Imagery/Map/{imagerySet}?mapArea={m
apArea}&mapSize={mapSize}&format={format}&mapMetadata={mapMetadata}&key={
BingMapsAPIKey}".format(imagerySet='Aerial', mapArea=coordinates,
mapSize=mapSize, format='jpeg', mapMetadata=0, BingMapsAPIKey=key)
```

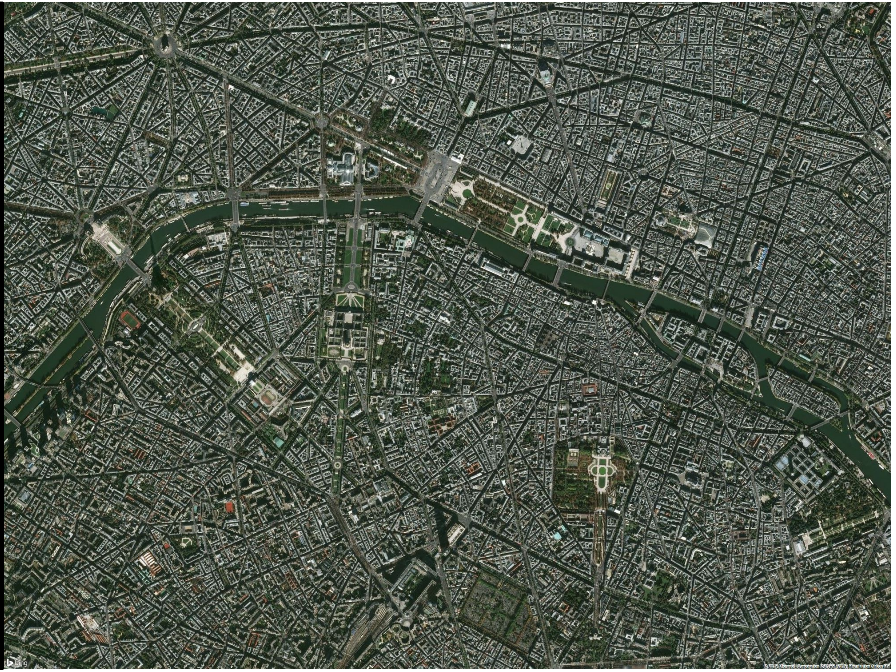
See the code: def *retrieve_satellite_image*

Alternative Imagery Retrieval Method

Code:

```python
# Define function
def retrieve_satellite_image(lat1, lon1, lat2, lon2):
   mapSize = '{},{}'.format(MAX_WIDTH, MAX_HEIGHT)
   coordinates = encode_coordinates(lat1, lon1, lat2, lon2)
   request_url = ("https://dev.virtualearth.net/REST/v1/"+
       "Imagery/Map/{imagerySet}?mapArea={mapArea}&mapSize"+
       "={mapSize}&format={format}&mapMetadata={mapMetadata}"+
       "&key={BingMapsAPIKey}").format(
       imagerySet='Aerial', mapArea=coordinates,
       mapSize=mapSize, format='jpeg',
       mapMetadata=0, BingMapsAPIKey=key)
   request = urllib.request.Request(request_url)
   response = urllib.request.urlopen(request)
   return response.read()

# Test retrieving image
lat_1, lon_1, lat_2, lon_2 = 48.8580, 2.2945, 48.8530, 2.3499
img = retrieve_satellite_image(lat_1, lon_1, lat_2, lon_2)
Image(img) # show image
```

The result is shown below:



References:

1. https://docs.microsoft.com/en-us/bingmaps/articles/bing-maps-tile-system
2. https://docs.microsoft.com/en-us/bingmaps/rest-services/directly-accessing-the-bing-maps-tiles
3. https://docs.microsoft.com/en-us/bingmaps/rest-services/imagery/get-a-static-map
4. https://docs.microsoft.com/en-us/bingmaps/rest-services/common-parameters-and-types/location-and-area-types

Thank you!