CS 513 Assignment 3

Aerial/Satellite Imagery Retrieval

By: Theo Guidroz, A20426895
Hunter Negron, A20417545
Zhengcong Xiao, A20453141

Contents

Introduction

Problem Statement

- Understand Mercator projection
- Understand tile coordinates and quadkeys
- Understand how to retrieve a satellite box
- Given a latitude-longitude bounding box, retrieve a satellite image at the highest resolution

Motivation

With the rise of technologies such as autonomous vehicles, geospatial has gain enormous importance in our everyday lives. One of the earliest features made possible in the geospatial field is aerial views where every pixel of an image corresponds to a precise geographic location (longitude and latitude). The goal of this assignment is directly related to this feature since it is to retrieve an aerial image.

Background

Mercator projection

The Mercator projection is a cylindrical map projection presented by the Flemish geographer and cartographer Gerardus Mercator in 1569. It significantly distorts scale and are but has two main advantages:

1. It preserves the shape of small objects (conformal)
2. The north and south are always straight up and down (cylindrical)

Mercator projection

Ground Resolution and Map Scale

At the lowest level of detail (Level 1), the map is 512 x 512 pixels. At the highest level of detail (Level 23), the map is 2,147,483,648 x 2,147,483,648 pixels.

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

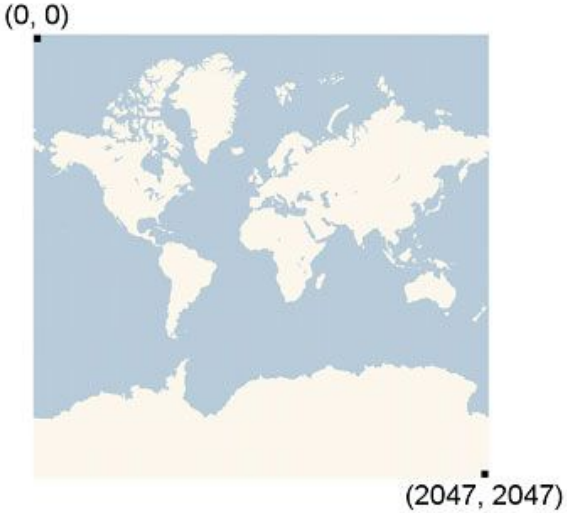
The ground resolution indicates the distance on the ground that's represented by a single pixel in the map.

ground resolution = cos(latitude * pi/180) * earth circumference / map width

Pixel Coordinates

To retrieve an image, geographic coordinates have to be converted to graphic coordinates. The pixel at the upper-left corner of the map always has pixel coordinates (0, 0). The pixel at the lower-right corner of the map has pixel coordinates (width-1, height-1).

The pixel coordinates can be calculated:




```
sinLatitude = sin(latitude * pi/180)
```

```
pixelX = ((longitude + 180) / 360) * 256 * 2
```

level

```
pixelY = (0.5 - log((1 + sinLatitude) / (1 - sinLatitude)) / (4 * pi)) * 256 * 2
```

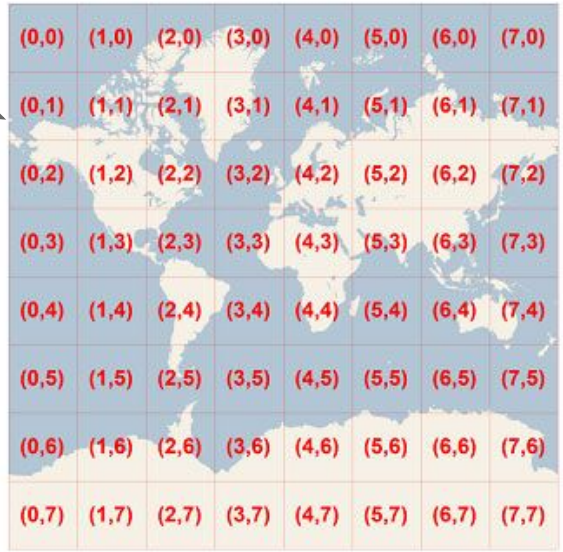
level

Tile Coordinates

The rendered map is cut into tiles of 256 x 256 pixels each. The number of tiles can be calculated using:

The pixel coordinates can be calculated: $map\ width = map\ height = 2^{level}\ tiles$

This is what the tile coordinates look like at level 3



The tile in which a pixel falls can be determined using:

tileX = floor(pixelX / 256)

tileY = floor(pixelY / 256)

Quadkeys

Quadkeys combines the two-dimensional tile XY coordinates into one-dimensional strings. 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. Quadkeys have three important properties:

1. The length of a quadkey equals the level of detail of the corresponding tile.
2. The quadkey of any tile starts with the quadkey of its parent tile.
3. Quadkeys provide a one-dimensional index key that usually preserves the proximity of tiles in XY space.

Approach

Approach #1: One Retrieval and Crop

Key Idea

- Since the Bing Map API is based on the quadkey/tile system, it would make sense to figure out exactly which tile the bounding box is located in, retrieve that tile, and crop it to the exact size of the bounding box. Since Bing provides methods for converting between coordinates, pixel locations, tile locations, and quadkeys, it is possible to find the most detailed level at which the bounding box fits in a single tile and find which pixels designate the bounding box.

Approach #1: One Retrieval and Crop

Implementation

- See “src/main.py” for the code
- Get Best Level (naive): this algorithm takes two sets of coordinates, southwest and northeast, representing the bounding box. For each detail level, 1 to 23, it converts the coordinates into tiles and checks to see if the tiles match. Once the tiles are no longer the same, we can conclude that the previous level of detail was the last one to wholly contain the bounding box.
- Using the best level of detail, call the API to get the the relevant 256x256 tile (by quadkey). This tile can then be cropped to the exact bounding box.

Approach #1: One Retrieval and Crop

Limitations

- The one major issue with this approach is that there are many cases in which the resulting image will have an extremely low resolution. This happens when a relatively small bounding box crosses the boundaries of two or more relatively large (low detail) tiles. In this case, the algorithm will determine that the bounding box fits in a giant tile, when in reality its size is small enough to fit in a much smaller one. Once the retrieved tile is cropped, the result will contain only a few pixels. This certainly defeats the purpose.

Approach #2: High Resolution

Key Idea

- We learned after running into problems with our previous approach that we need to retrieve more than one tile. The first thought then is to just get all the tiles the touch the bounding box at the highest level of detail and stitch them all together.
- However, for a relatively large bounding box, this may mean millions of tiles, which is definitely not feasible. Therefore, we need to get tiles at the highest level of details that produces reasonably-sized output.

Approach #2: High Resolution

Key Idea Continued

- Knowing the coordinates of the bounding box and given a reasonable maximum size for the result, we can iterate to find the highest level of detail such that the result is less than or equal to the maximum size. Then, all the tiles that touch the bounding box can be retrieved at that level and stitched together to produce one final result.

Approach #2: High Resolution

Implementation

- See “src/main2.py” for the code
- Get Best Level: Again take two sets of coordinates, southwest and northeast, representing the bounding box. Iterate over all possible level of details starting at 23 (most detailed) and going down. For each level, get the tiles of the corners of the box. Using their tile coordinates, compute the pixel width and height of what the resulting image would be. If neither of these dimensions exceeds the maximum, 4096 as we have chosen, then we have found the best level.

Approach #2: High Resolution

Implementation Continued

- Once the best level is calculated, get the tile coordinates for the corner tiles that the bounding box touches. These are used as bounds for iteration over both the width and height of the box. Retrieve each 256x256 tile from the Bing Map by specifying the center point in the request. Add it to the final resulting image into its appropriate position.
- The final resulting image will now be a perfect composite of all of the detailed tiles that cover the bounding box.

Extra feature

Extra feature

- In addition to returning the highest quality image, the program prints the places of interest found within the bounding box of coordinates.
- It lists the most popular parks, art centers, observation points, and museum
- The program finds the center of the bounding box, the radius of the maximum circle that could fit in the box and searches for the points of interest within the radius

Results

sampleResults

- ./main2.py 41.860034 -87.628431 41.862233 -87.625650 sample.png #(1)(1.1)

(1)

```
Most popular Park in the area is:
Coliseum Park ,  1466 S Wabash Ave
Fred Anderson Park ,  1629 S Wabash Ave
Cottontail Park, Chicago ,  1500 S Dearborn St
Most popular Art Center in the area is:
Daystar Center ,  1550 S State St
Piano Forte ,  1335 S Michigan Ave
Reggie's Music Joint ,  2105 S State St
Most popular Obsevation Point in the area is:
Museum Park ,
Mississippi Blues Trail Marker ,
Most popular Museum in the area is:
Chicago Art Exchange ,  1530 S State St
Art Media Resources ,  1507 S Michigan Ave
American Police Center Museum ,
```

(1.1)

When input a correct coordination, then the png will show up in the src directory of (1).
The (1.1) will be shown in the console of the popular places in this area.

sampleResults

- ./main2.py 42.765423 -88.634399 41.760132 -88.632542 illinois_tech.png #(2)

```
PS E:\Education\MyIIT\CS513\CS513-main\hw3\src> ./main2.py 42.765423 -88.634399 41.760132 -88.632542 illinois_tech.png
level:  21
Tiles:  -94956
Traceback (most recent call last):
  File "E:\Education\MyIIT\CS513\CS513-main\hw3\src\main2.py", line 113, in <module>
    getImage(*coords, outFile)
  File "E:\Education\MyIIT\CS513\CS513-main\hw3\src\main2.py", line 49, in getImage
    final = Image.new('RGB', (true_width*256, true_height*256))
  File "C:\Python39\lib\site-packages\PIL\Image.py", line 2614, in new
    _check_size(size)
  File "C:\Python39\lib\site-packages\PIL\Image.py", line 2593, in _check_size
    raise ValueError("Width and height must be >= 0")
ValueError: Width and height must be >= 0
```

(2)

The (2) is a test that we are setting a large bound to be subtracted by a small bound, then we can see that the tiles is negative.

Results-IIT

- ./main2.py 41.830781 -87.630025 41.838600 -87.623159 illinois_tech.png #(1) (1.1)
- For the extra feature, it will show the popular places in the console. However, if there is no places, such as the museum shown in (1.1), it will nothing in the console.

(1)

```
Most popular Park in the area is:
Armour Square Park ,  3309 S Shields Ave
Dunbar Park ,  300 E 31st St
Williams Park ,  2710 S Dearborn St
Most popular Art Center in the area is:
We Care Role Model Program ,  3510 S Michigan Ave
We Care Role Model Program ,  3618 S State St
Most popular Obsevation Point in the area is:
Most popular Museum in the area is:
```

(1.1)

Results-Chicago

- ./main2.py 41.86141480683211, -87.65674715469648 41.89992289326928, -87.57318069078742 chicago.png #(2)(2.1)
- (3.1) is showing the popular places of Chicago.

(2)

```
Most popular Park in the area is:
Maggie Daley Park ,
Carl Milles Sculpture-Mckinlock Court ,
Millennium Park ,  201 E Randolph St
Most popular Art Center in the area is:
James C Petrillo Music Shell ,
Jay Pritzker Pavilion ,
Chicago Opera Theater ,  205 E Randolph St
Most popular Obsevation Point in the area is:
Prudential Rooftop Patio ,
Park Millennium Sun Deck & BBQ ,
Millennium Park Plaza Rooftop ,
Most popular Museum in the area is:
C R Advertising Arts Studio ,  400 E Randolph St
Chicago Stock Exchange Room ,
Art Institute of Chicago - the Modern Wing ,  159 E Monroe St
```

(2.1)

Results-Australia

- ./main2.py -49.685612 90.971393 9.351771 179.216107 australia.png #(3)(3.1)
- (3.1) is showing the popular places of Australia.

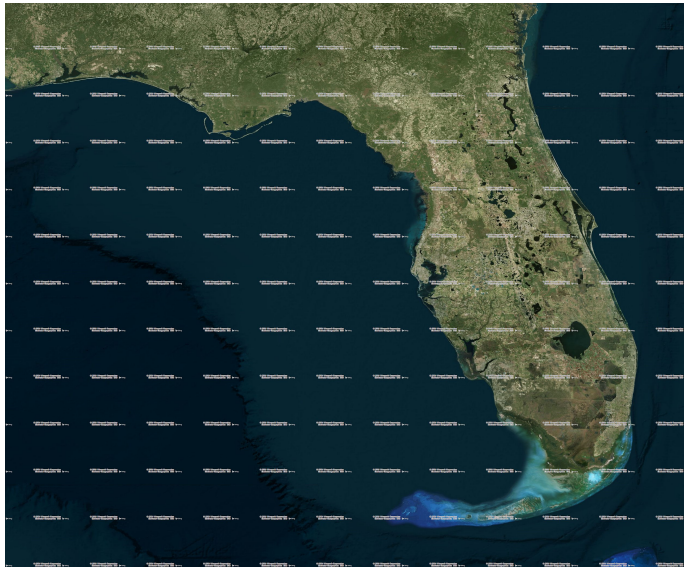
(3)

```
Most popular Park in the area is:
Davenport Ranges National Park ,
Karlu Karlu/Devils Marbles Conservation Reserve ,
The Devils Marbles Conservation Reserve ,
Most popular Art Center in the area is:
Winanjjikari Music Centre ,  52 Peko Road
Central Dance Theatre ,  45 Mulara Street
Baggage Claims ,  18 Bath Street
Most popular Obsevation Point in the area is:
Bill Allen Lookout ,
Anzac Hill ,
Cassia Hill Lookout ,
Most popular Museum in the area is:
Tennant Creek Museum at Tuxworth Fullwood House ,
Julalikari Arts ,
Arlpwe Art & Culture Centre ,  160 Kinjurra Road
```

(3.1)

Results-Florida

- ./main2.py 24.350755 -87.867736 31.133314 -79.507140 florida.png #(4)(4.1)
- For the extra feature, we can see that the most popular places of Florida, such as names, addresses and so on, will be shown in the console.



(4)

```
Most popular Park in the area is:
Indian Shores Municipal Park 1 ,
Chief Chic-A-Si Park ,
Indian Rocks Beach Nature Preserve ,  915 Gulf Blvd
Most popular Art Center in the area is:
Largo ,  12046 Indian Rocks Rd
Largo ,  1001 W Bay Dr
Largo Cultural Center ,  65 4th St NW
Most popular Obsevation Point in the area is:
Swing on Gulf Boulevard ,
Town of Indian Shores Town Square Nature Park ,
Randolph Farms ,
Most popular Museum in the area is:
Indian Rocks Beach Historical Museum ,  203 4th Ave
Impressive Florida Wines ,  311 1st St
Gulf Beach Art Center ,  1515 Bay Palm Blvd
```

(4.1)

Results-Care

- ./main2.py 41.880142 -87.627599 41.884423 -87.617342 #(1)
  - If no file name is inputed, then error message will show up, see picture (1).
- ./main2.py 41.880142 -87.627599 41.884423 -87.617342 illinois_tech
  - If we input the file name, but we do not input the format of the file, the code will still run it and generate a file named illinois_tech with no format.

```
PS E:\Education\MyIIT\CS513\CS513-main\hw3\src> ./main2.py 41.880142 -87.627599 41.884423 -87.617342
Incorrect number of arguments.
Usage: ./main2.py lat1 lon1 lat2 lon2 outFile.png
lat1 = South Latitude
lon1 = West Longitude
lat2 = North Latitude
lon2 = East Longitude
```

(1)

Conclusion

Conclusion

- The code can calculate the map width/height.
- Accepting the bounding box coordinates and level of detail.
- If the coordinates make the tiles as negative then there will be a error message showing up.
- All equations show in the Bing Maps Tile System work.
- When input the right coordinates, given the name “xxx.png”, then the xxx.png will be generated in the src directory.
- Extra features, popular places, shows in the console

References

https://docs.microsoft.com/en-us/bingmaps/articles/bing-maps-tile-system