Annotating the NYC Skyline

Max Ogryzko
Columbia University
mvo2102@columbia.edu

Roop Pal
Columbia University
rmp2191@columbia.edu

Patrick Stanton
Columbia University
pws2111@columbia.edu

Abstract

Many if not most of us have looked at the awe-inspiring New York City skyline and been curious about each building and its history. We believe that people are unable to fully appreciate the skyline without knowing this information. We sought to change this. Our program attempts to identify buildings within images of the New York City skyline taking in only smartphone data and a 3D model. By converting the phone data into the model space, we were able to create a projection of what the buildings are suppose to look like at a given location. Using alignment techniques, we were then able to outline the building and provide relevant information. We believe we have set the framework for an innovative application.

1. Introduction

Every year, millions of tourists flock to New York City. Brought by the towering skyscrapers, tourists leave the city in awe of its architectural wonders. Whether it be the Art Deco of the Chrysler building or the Neo-Gothic Trinity church, New York has great diversity of style. However too often tourists are unable to fully appreciate these buildings. They see them in the skyline but are unable to identify them. With this in mind, we sought to design a computer vision program that would allow anyone to use their smartphone to identify any building in New York City.

Object detection and recognition is one of the most researched areas of computer vision. By utilizing neural networks, many researchers have been able to design programs that have significantly outperformed previous works. These programs however rely on huge datasets of labeled images for training. We sought to identify these buildings without utilizing this technique. Only using a 3D model of the city and data from our smartphone, we wanted to design a system that could identify any building in the city. We believe that through this project we have set the framework for a comprehensive New York City skyline annotator.

2. Related Works

2.1. NYC 3-D Building Model

In 2014, the New York City Department of Information Technology and Telecommunications conducted an aerial survey of every single building present at that time. They then compiled this data into a series of CityGML files that encompasses all five boroughs of the city. For this project, we decided to only use the data from the 12th file of the dataset, which encompasses Lower and Midtown Manhattan[5]

2.2. MeshLab

In order to test and visualize our dataset, we used a program called MeshLab. It is an open source system that provides a host of 3D digitization tool and devices [6]. We frequently used it to check if our conversions for real world to model space were correct. We also used it to make sure our phone rotation vectors were in fact correct. The program nonetheless has significant problems with its user interface which made it difficult to use.

3. Methods

3.1. Data

3.1.1 Parsing

The first step in our method was to parse the DOITT New York City 3D model data. The data files we used are in City Geographic Markup Language (CityGML) which is a XML-based geospatial format[4]. In this dataset, each building has unique identifier in the format: Bldg XXXXXXXXXXX. Following the identifier is a set of polygons, each representing a side or surface of the building. Each polygon, called a ”posList” in the file, is a set of 3D points that make up the surface. When visualizing all of these polygons, a fairly precise facade of the building is created. In order to extract these polygon in our code, we designed a GML parser such that the building identification and corresponding polygons are extracted from the file and put into a python dictionary. During the parse, we also extract the minimum and maximum x, y, and z positions in order to get a sense of the area encompassed by the building, which we later use to identify the building.

3.1.2 Sensors

Modern smart-phones are equipped with a powerful kit of sensors that can be utilized to run our application. In order to model where the individual took the picture, we need locate them and understand where they are pointing their camera. Although we focus on Android devices [1] because they make up a majority of the smartphone market, iOS devices have very similar arrays of sensors. In order to get user position we collect latitude and longitude, and to get direction we collect x, y and z axis rotation. With this sensor data we now have an (x,y) location, as well as enough info to determine direction. We still need to determine our z location, but this is more difficult as smartphone GPS systems have a high degree of inaccuracy in terms of altitude. Therefore, we have to assume street-level altitude.

The sensor data returns a rotation vector of Euler angles which informs how to rotate the phone to sit in an orientation on the North-East plane, normalized to [-1, 1]. However, in practice, the rotation vector was often inconsistent with its own readings and had high uncertainty. We followed the examples of applications like Google Maps by having users do a figure-8 motion with the phone to calibrate these sensors to lower inaccuracy.

3.1.3 Annotation Dataset

In order to annotate buildings, we manually generate a dataset with building IDs and information about the buildings. We use Wikipedia[2] to gather information about the buildings and manually paired them with building ids. In total, we scrape data for 93 buildings including name, number of floors, height, date completed, and paragraph descriptors.

3.2. Conversion

After receiving sensor inputs, the real-world position and orientation have to be converted into the 3D model's coordinate system and then a camera (or projection) matrix in order to convert from 3D to our 2D view. The camera matrix is made up of an intrinsics matrix, a rotation matrix, and a translation vector. We first convert the Euler angles we receive from sensor data to radians and then into a common rotation matrix. We can convert latitudes, longitude, and altitude into a translation vector $T = (x, y, z)$ in the virtual world using references points and a homography. Finally we combine the rotation matrix and translation vector as well as smartphone camera intrinsic information to create the camera matrix.

3.3. Collision

One of the most challenging parts of our method is determining what building are within our field of view from the location and orientation data. To solve this problem, we designed a function collision.py. In this file, we begin by converting the Euler angles into a theta and alpha value. The theta angle measures the cardinal direction we are facing in radians with 0 being east and $pi/2$ being due north. The alpha angle, on the other hand, measures the angle in relation to the cardinal direction plane. From this information we then visualize a line of increasing length that senses when it is has intersected with a building. If the line leaves the bounds of the model, then the function outputs false. Obviously, unless the building is perfectly centered within the image, the correct building will not be identified. For this reason, we include an error parameter such that multiple lines are drawn in slightly different directions. Once we have identified the building within the field of view, we pass the building identification number to the projection function.

3.4. Projection

In order to highlight the building on the users screen, we imitate our position and direction within the 3D model using sensor data from the smartphone and then project each point in the building from its 3D coordinates to our current 2D view. In order to convert from 3D to 2D, we calculate a camera (or projection) matrix we calculated earlier. Converting from latitude and longitude to model coordinates gives us a position within the model, and the Euler angles from sensor data can be used to find the direction we are looking in. Once we have a camera matrix, the projection into our 2D view comes simply from multiplying the matrix by each building point, and then normalizing over the new z coordinate[3].

As every building in the GML file is organized in terms of its surfaces, we can project the points from each individual surface into their 2D coordinates, and then use OpenCV's 'fillPoly function to fill each surface in our new 2D view. Once all surfaces are filled, we get a binary mask of the building in the same orientation and position as the input photo.

3.5. Alignment

In a perfect world, our method for projecting points from 3D to 2D would line up exactly with our input photo. However, due to slight errors in sensor data, this is often not the case. In order to ensure a good fit between the projection and phone photo, we need to align our projected mask with the actual building in the photo. Our original thought was to use SIFT to calculate similar points between the projection and the photo and then use these similar points to compute a homography between in the images. Unfortunately, the

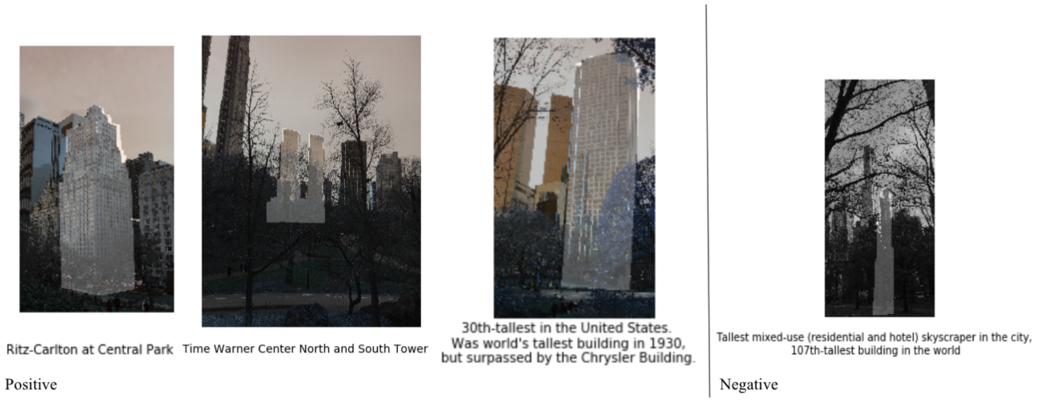
Ritz-Carlton at Central Park
Time Warner Center North and South Tower
30th-tallest in the United States. Was world's tallest building in 1930, but surpassed by the Chrysler Building.
Tallest mixed-use (residential and hotel) skyscraper in the city, 107th-tallest building in the world
Positive
Negative

Figure 1. Sample Results

GML file often does not contain enough detail for SIFT to work with confidence. We chose a simple SSD (sum of squared distances) function, which works well since we can assume that our projection mask very similarly matches the orientation and size of the building in the input photo.

4. Results

Various components of the application had varying degrees of success. As certain components did not meet expectations, we increased assumptions in order to give the various other functions valid inputs.

4.1. Converting Sensor Data

Conversion from the real world to the 3D model is the base of our application, so accuracy of sensor values is extremely important. The sensor data, however, was not sufficiently accurate to give meaningful results in the virtual environment. In particular, the altitude could not be determined with a high degree of precision due to the conversion from barometric pressure to altitude requiring more inputs than are available. Additionally, without calibration, the orientation data was inconsistent with itself. The rotation vector could give readings up to 0.5 (180 degrees) different despite facing the same exact direction. This caused almost completely unusable orientation data, so we were forced to gather rotation information from MeshLab manually.

4.2. Collision

After getting a correct rotation matrix and translation vector, we used collision detection to understand which building was pointed at by the user. This function was successful at ID'ing buildings in simple cases, but in cases in which the orientation was complicated, the building very off-center, or other buildings in the way, the building ID could not be automatically found. In these cases, we are forced to hard-code the building ID.

4.3. Projection

The projection of 3D faces of an object into a 2D filled outline was consistently successful and matched the shape of the target image. All inaccuracies can be confidently attributed to inaccurate sensor data.

4.4. Matching

Our initial results from SIFT were not promising – many of the key points were matched with noise or arbitrary features. RANSAC, a function that returns a homography between images, resulted often in an incorrect homography. With SSD we were able to successfully match the more distinct building shapes, but often could not properly match the more rectangular or simple buildings. Additionally, occlusions from objects such as branches greatly reduced accuracy of matching.

5. Discussion

5.1. Sensor Inaccuracies

One of the largest sources of error was the unreliability of smartphone sensor data. In particular, the altitude cannot be precisely calculated and the orientation is almost completely useless without calibration. Altitude can perhaps be locally approximated by using weather data to understand how altitude depends on pressure in New York City, but this approach is not scalable. We can calibrate the phone to reduce orientation inaccuracies by pointing it to a known landmark or due North, or by executing the "Figure-8" motion to ground it (as in Google Maps).

5.2. Collision Detection

We had significant troubles with the collision function that we wrote for this project. For our first iteration of this function, we would only visualize one line that represented the center of the image. This had the limitation that only buildings in the center of the image would be identified.

In order to expand upon this, we incorporate an error rate to calculate the angle so that now twenty lines are being projected in different directions. We then output all of the buildings that are detected at a specific iteration along the line. This method however scarcely outputted the building we wanted to identify. We believe that the main issue is inconsistency with phone sensor data and difficulties converting Euler angles into theta and alpha values. One way we could expand upon our project and render these issues irrelevant would be to project lines in stochastic manner at different angles within the frame of view. This would allow to identify multiple building in one image and is more likely to identify the desired building. This issue is one of several ways we can expand upon this project.

5.3. Matching

It may be possible to improve our alignment algorithm while still using SSD by improving how we process the projection image and the original input photo. Because our projection is a binary mask, SSD currently rewards blocks of high pixel values which may not be the building we are looking for, such as a tree or other obstruction in our image. Running our projection mask through a Sobel filter or other edge detector may solve this issue by only rewarding objects in the input photo which have the same outline as our projection; hopefully only the building we are interested in. In addition, we can assume that our projection will have a similar orientation and position to the building in the input photo, so cropping the input photo around an area we expect the building to be in may improve alignment accuracy.

References

[1] Android sensors. `https://source.android.com/devices/sensors/sensor-types`. Accessed: 2018-12-10.

[2] List of tallest buildings in new york city. `https://en.wikipedia.org/wiki/List_of_tallest_buildings_in_New_York_City`. Accessed: 2018-12-10.

[3] P. C. Carl Vondrick, Deva Ramanan. Lecture 9, computer vision, October 2018.

[4] O. G. Consortium. Citygml, 2018.

[5] D. of Information Technology and Telecommunications. Nyc 3-d building model, 2014.

[6] M. C. M. D. F. G. G. R. P. Cignoni, M. Callieri. Meshlab: an open-source mesh processing tool. Sixth Eurographics Italian Chapter Conference, page 129-136, 2008.