NYC Transit Interactive Visualization

New York University Shanghai

CSCI-SHU 370-001 Object Oriented Programming

Jack B. Du (Jiadong Du) - May 15, 2015

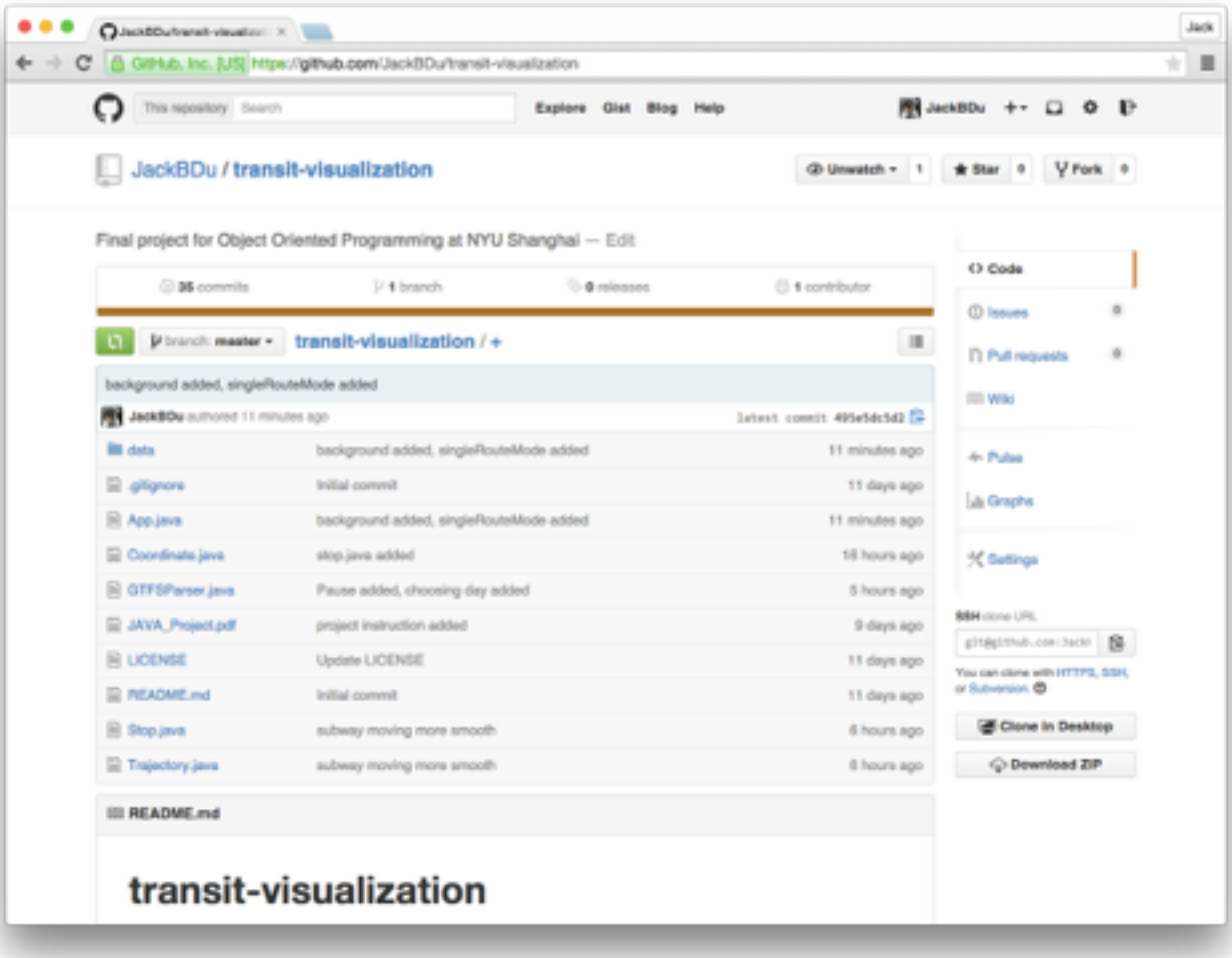
Introduction

This project is an NYC transit interactive visualization built in Java. MTA New York City Transit is the largest public transportation agency in North America and one of the largest in the world. The subway has a daily ridership of 5.6 million and an annual ridership in 2014 of roughly 1.751 billion. Our fleet of more than 6,300 subway cars traveled approximately 361.1 million miles in 2014, along 660 miles of track, 24 hours a day, seven days a week, although not all routes operate around the clock. There are 468 subway stations, including 109 made accessible to customers with disabilities, via elevators and ramps. NYC Transit Subway serves Brooklyn, the Bronx, Manhattan, and Queens, making 7,883 weekday trips. MTA Staten Island Railway (SIR) serves Staten Island and has four ADA-accessible stations. (MTA)



The purpose of this project is to provide to the audience a visually pleasing way to view the NYC transit, both for study and for arts. The application simulates the positions of all the public vehicles and lets the user to view the simulation with the user's preferences. The project aims for those who are experts that need to study the transit system as a whole or those who are interested in seeing the NYC transit system and playing around with it, or even those who are just interested in arts and would like to watch the stunning visualization of the NYC transit.

Code Architecture



All the source code of the project is hosted on GitHub. The repository can be found here: https://github.com/JackBDu/transit-visualization.

I used GitHub not only because its ability to revert to any history version as git does but also because it is an open source platform so that the project could be used by someone else and others can even contribute to it, which makes this project more promising, because it may be developed to be something much more powerful. Below is the high level explanation of all the source code.

Coordinate.java

Coordinate is the most basic class of the project. It simply stores the latitude and longitude of a coordinate with the ability to transform the class to String, which is handy for debugging, and to return the latitude/longitude values so that those values could be easily accessed as needed.

Stop.java

Stop is also a very basic class, and it basically stores all the information that you need for a stop. It contains the stop ID, stop name, stop coordinate, location type and parent station. Some of the data is not used in my application, but since these data are in the file, I keep them so that I may use it as I improve this project. Again, it provides the ability to transform all the data into String and also return all the values that you may need.

Trajectory.java

Trajectory is a bit more complicated compared with the previous two classes. This class basically stores the whole route for a single vehicle throughout the day. Besides that, it has the service ID, route ID, trip ID. `public boolean isActive(long time)` checks whether or not the vehicle is active at given time. `public Coordinate getPosition(long time)` returns the coordinate of the vehicle at a specific time. Since only arrival and departure time for each stop is provided, it is impossible to query for any given time. So once there is a time passed it, it will first check whether or not the vehicle is active. If it is not active, the coordinate must be the coordinate of the stop that the vehicle leaves at every day. If it is active, it checks through all the stops that it passes by, if the vehicle is at a specific stop at the given time, the method just returns the coordinate of that stop, if not, it will find between which two stops it is at the given time. Then, according to the distance between two stops and the time it has spent after leaving the stop, we can calculate how far it has gone. Note that this is not accurate

because the route between two stops are not necessarily straight lines. However, it turns out to be effective because approximately, most route between two stops are straight. That is the key point why my application runs much more smoother than other students. Since I am not making an app for users to navigate and travel around NYC, instead, it is an application that demonstrate the general flow of all the transits in NYC so that it does not have to be 100% accurate and it makes my application run smoothly so that we can appreciate and study the flow of the transits.

GTFSParser.java

GTFSParser plays one of the most important roles and it matters a lot to the startup speed. At the very beginning, it took 10 to 20 minutes for the GTFSParser to parse all the data, which is insanely ridiculous. However, after some optimization, now it takes less than 20 seconds to parse all the needed data. Generally speaking, that is because the previous version loop through three files and try to match the three while the newer version only loop through two files and one of them is very small so that it does not take too long. The reason why there is one less file is that the information that I need is actually contained in the biggest file "stop_times.csv" so that I do not need to match this file with the other big file.

`public static ArrayList<Map<String, String>> readCSV(String filePath)` is the method that read the CSV files with the given path. It reads the file and convert it to a ArrayList of Map<String, String> so that it can be manipulated.

`public static String getServiceIdFromTripId(String tripId)` slices the service ID from trip ID and `public static String getRouteIdFromTripId(String tripId)` slices the route ID from trip ID.

`private static long toSeconds(String time)` converts the time in HH:MM:SS to seconds so that the simulated time would be in second. Since the schedule is the same

on Saturdays, Sundays and Workdays so that what we need to know is only the time in a day and what day it is.

`public static ArrayList<Trajectory> parseTrips(String folderPath)` parses all the files that we need to create the ArrayList of Trajectory. Basically, the "stop_times.csv" and the "stops.csv". It loops through each time in the former and find the coordinate of the corresponding stop and stores these matched information in the Trajectory.

App.java

App is a class that extends JPanel, where all the visualization is being painted. This class stores most of the data that needs to be displayed or transformed and also handles most of the events that are required for the interaction.

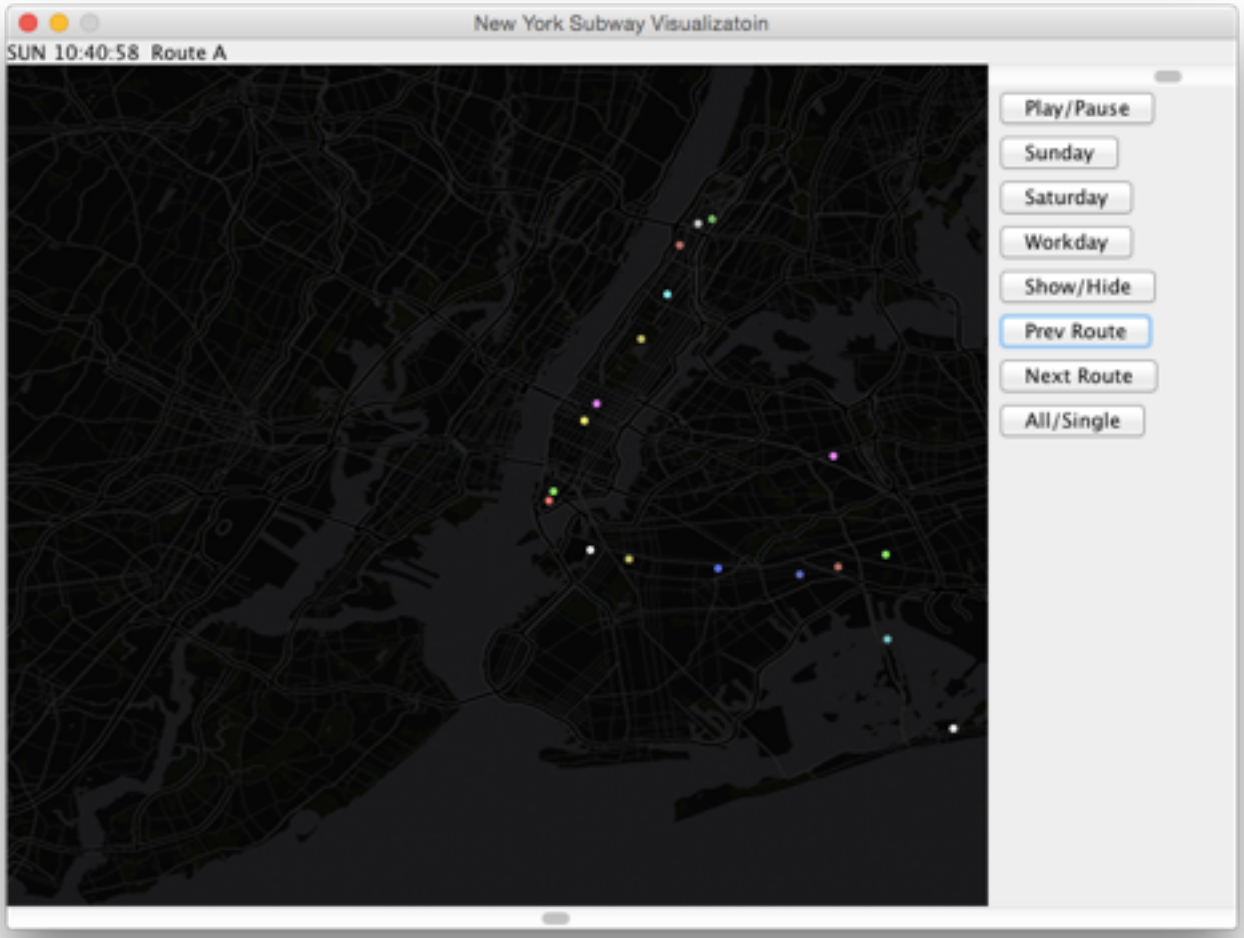
In `public void paint(Graphics g)` method, it loads an external image and displays it at the bottom layer of the graphics. Then as long as the routes are not hidden, it will go through the ArrayList of the shapes and paint all the lines. Since there are so many lines, which take up a lot of CPU to paint so that it slows down the whole application, I disabled it for default so that the animations are much smoother and it looks more pleasing. After painting the lines, it paints the vehicles. Basically, it goes through all the trajectories for all the vehicles and finds the corresponding position for the current simulated time. It chooses a random color from a predefined color array and assign it to each vehicles so that different vehicles are distinctive. Also, the background image is very dark so that the vehicles, which are bright dots, would really stand out from the background and look like little lightbulbs at night so that they are beautiful.

`public void parseRoutes()` method parses all the routes from the file and store the route names in an array so that the user can loop through each route in single route mode. `public void prevRoute()` navigates to the previous route while `public void nextRoute()` navigates to the next route. There are also methods that handles "play/

pause" and "show/hide" and `public void setDay()` that switch between Sunday, Saturday and Workday. `public void setSpeed()` sets the speed, which actually changes the time that the thread sleeps. `public void setTime()` sets the time of a day from 00:00:00 to 23:59:59 while the `public void update()` updates the simulated time.

Finally, the `public static void main(String[] args)` is where the most high level stuff is going on. This whole application starts from this method. It first parses all the trips that are in the data file and converts it to an ArrayList of Trajectories so that it could be later looped though and painted. It creates the JFrame, where the Panel and all other components are placed and it also creates all the buttons and scroll bars. It handles all the events that are related to those buttons and it controls the main flow of the visualization, or in other words, those animations.

Features



Basically, what this application does is to display all the NYC transit in the window at simulated time. But it is demonstrated in a quite artist way. As the background is quite dark, the vehicles that are simulated are bright and look like stars in the dark sky so that the way vehicles move really forms a kind of beautiful routes that are visually pleasing.

On the top-left corner of the window, it displays the day it is simulating and the time it is simulating and the current route if it is in single route mode. So this corder basically displays all the useful information that you need to know about the simulation.

On the right hand side, there is a side bar, which is basically a control panel. On the top of the side bar, there is a scroll bar that controls the speed of the simulation. The next button is “Play/Pause” that can pause or resume the simulation. Below that, there are three buttons that are used to switch between Sunday, Saturday and Workday. Following that is the button that shows or hides the routes. Then the “Prev Route” and “Next Route” is used to switch to previous or next route if it is in single route mode. The last button down there is to toggle between single route mode and all routes mode.

At the bottom of the window, there is a scroll bar that controls the simulated time. The user can scroll to change over the simulated time.

Conclusion

The NYC transit interactive visualization not just visualizes the NYC transit data for academic or research use, but also a piece of interactive art itself. Interactive Media Arts is still relatively a new major and new field, but since I am majoring in both Computer Science and Interactive Media Arts, I will try, in my future career, to integrate those two subjects seamlessly. I like technology because technology is changing our lives and technology helps me to think reasonably and scientifically. I like arts because arts help me to relax and arts help me to achieve immortality. Thus, I believe that the integration of technology and arts, or in other words, the combination of Computer Science and Interactive Media Arts will really help me to make the world a big difference. Just like this project, it not only shows my ability to make use of my own technical skills but also conveys my own way to express artistic concepts through technical project.

One thing I can do to further explore transit visualization is to add the ability to zoom in and zoom out as well as pan around the map. Thus the visualization would be more interactive. Another thing is that there might be information bubble that shows up for each stop and each line so that the user can know more about the transit. In general, there could be more features that allow users to gain more information to improve its usability. On the other hand, as an artistic work, there could be more effects and animation that makes it even more visually pleasing.