ECE297 Milestone 2
Visualizing an Interactive Map

"The real voyage of discovery consists not
in seeking new landscapes, but in having new eyes."

–Marcel Proust

Assigned on Friday, Feb. 11 In-lab demo = 13/15

Due on Friday, March 11 at 11:59 pm Code style and project management = 2/15

Autotester = -1 penalty if errors

Total marks = 15/100

1 Objectives

In the previous milestone you learned how to query libstreetsdatabase to get useful information about a map, and you started building your project in libstreetmap by implementing a number of functions. In this milestone you will use a graphics library called EZGL to visualize the map, and you will add functions to find elements in the map and display detailed information about them. While there are many graphics libraries, almost all use similar conventions, so learning EZGL will also give you experience that is very useful in working with any graphics library in the future.

After completing this milestone you should be able to:

1	Visualize data, in this case a graph of a city map, using computer graphics.
2	Develop an user interface to respond to user queries and display data.
3	Learn how to use the EZGL graphics library.
4	*(Optional)*: EZGL is built on top of a more full-featured graphics library called gtk, and you *may* choose to directly use some of its user-interface functions.

2 Problem Statement

In this milestone you will extend your project to implement the function in "m2.h" by adding one or more files to implement the graphics functions for your application. You will try to

implement all the features that you proposed in your **WD1: Graphics Proposal** document, starting with the most important ones in case you run out of time. You can adjust the plan you laid out in your graphics proposal without penalty, however, as in this course we recommend iterative development which frequently results in plan changes.

3 Specification

There are several **required features** that your program must support.

- Your program must be able to load and visualize any map *without recompilation* (i.e. you must get the map name from user input in some way).
- You must be able to visualize streets, points of interest and features (lakes, buildings, etc.) – essentially all the items provided by the layer 2 `StreetsDatabaseAPI.h`. Use the equirectangular map projection described in milestone 1.
- You must be able to show street names and point of interest names.
- The user must be able to tell major roads from minor roads, and tell which roads are one-way and in which direction.
- The user must be able to click on an intersection; the intersection should be highlighted and information about it should be either displayed in the graphics or printed to the command (terminal) window.
- The user must be able to enter the names of two streets (for example by clicking on `Find` button and entering the street names) and have all intersections between those street names be highlighted in the graphics and information about the intersections be shown in the graphics or printed to the terminal window.
- The find feature above should also work with partial street names in some way. For example if the text typed uniquely identifies the street, a partial street name is sufficient.
- The map should be fast and interactive to use, and while it displays all the information above it should do so in ways that do not overwhelm the user with unnecessary detail and clutter.

To receive full marks, you must not only implement the required features above well, but also devise and implement some interesting and useful **extra feature** functionality. Some ideas to get you started are listed below, but you are certainly not limited to these ideas – it is very much encouraged for you to come up with other ideas on your own. You can implement multiple extra features if you wish; demonstrate all your extra features to your TA so they are all considered during grading. *Possible* extra features include:

- Extract and display additional information from the more detailed "layer 1" API we have provided to OSM, which is described in the **OSM Quick Start Guide**.
- Implement additional highlighting functions when a user clicks the mouse, such as highlighting a street, building, park, etc.

- Upgrade the `Find` feature so that it can auto-complete or suggest street names, or find an intersection even if there are minor spelling mistakes in the street names. This feature would also be very useful in milestone 3 to improve the usability of your program for travel directions.
- Implement additional `Find` features that allow the user to find streets, points of interest, buildings, etc. by name.
- Make your user interface more usable by incorporating elements like dialog boxes to enter data for find or other commands, rather than requiring users to enter text at the command line.
- Extract useful live data such as traffic accidents from a web site and display this data on your map. The **libcurl Quick Start Guide** has helpful information on how to extract data from the web using C/C++.
- Make your program speak instructions using the festival text-to-speech library, which is installed on the UG computers. To learn more about festival, see http://www.festvox.org/docs/manual-2.4.0/festival_toc.html.
- There are many more features you could implement. Be creative and come up with your own, focusing on features that are truly useful!

4 Walkthrough


~40mins

Refer to "ECE297 Quickstart Guide: Graphics with EZGL" and the associated example graphics project to get started with EZGL.

After learning how to use the EZGL graphics library, you will update your NetBeans project to add the EZGL .h/.cpp files. Only one team-member needs to execute the command below (others will need to perform a subsequent `git pull -r`):

```
> ece297init 2
```

Next, implement the function in **m2.h** (Listing 1). The function `draw_map` should draw the map and allow the user to interact with it. Your main program should (and our unit tests will) call `load_map` on the appropriate map before calling `draw_map`. See the **"WD1: Graphics Proposal"** communication deliverable on the course website for a discussion of important aspects of making your graphics interactive, user-friendly, and capable of displaying information in a way that enables understanding by the user.

```
/*
 * Copyright 2022 University of Toronto
 *
 * Permission is hereby granted, to use this software and associated
```

```
 * documentation files (the "Software") in course work at the University
 * of Toronto, or for personal use. Other uses are prohibited, in
 * particular the distribution of the Software either publicly or to third
 * parties.
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
 * SOFTWARE.
 */
#pragma once

// Draws the currently loaded map.
// drawMap () assumes the map has already been loaded with loadMap ()
// before it is called. The ece297exercise unit tests always call
// loadMap (string map_name) before calling drawMap. Your main () program
// should follow the same convention.
void drawMap();
```

Listing 1: m2.h.

Note that unlike milestone 1, you must make a `main()` function for this milestone, and typically you will write it in `mapper/main/src/main.cpp`. You will demo your mapper program, which starts execution in your `main()` function, to your TA when you are marked. You can also use the `ece297exercise` command to test your `draw_map` function on various cities and run it through valgrind, but your demo will be of your mapper program, not these unit tests.

Your graphics should support any map in our ".bin" format. You will find several maps to test out under the **`/cad2/ece297s/public/maps`**; the smallest of these maps is "saint_helena" which is useful for testing for memory errors or leaks with Valgrind or the debug_check build.

Use a small map, and run your program with Valgrind or debug_check to make sure that your code has no memory errors or leaks.

4.1 Features

You already know how to use most of the functions of *StreetsDatabaseAPI.h* from milestone 1. However, there are two functions, which get the name and a type of a **feature** (see Listing 2), that you haven't used yet.

```
const std::string& getFeatureName(FeatureIndex featureIdx);
FeatureType        getFeatureType(FeatureIndex featureIdx);
```

Listing 2: Features functions from StreetsDatabaseAPI.h.

Features on a map include parks, ponds, rivers, lakes and beaches; the various types are given in Listing 3.

```
enum FeatureType {
    UNKNOWN = 0,
    PARK,
    BEACH,
    LAKE,
    RIVER,
    ISLAND,
    BUILDING,
    GREENSPACE,
    GOLFCOURSE,
    STREAM
};
```

Listing 3: Types of features available in the StreetsDatabaseAPI, from Feature.h.

As you learned in milestone 1, each feature is a closed polygon or multi-segment line defined by a number of Lat/Lon points, and you an ask the StreetsDatabaseAPI for these points. An example of a closed polygon is a `Lake` or `Building`, while an example of a *line* feature is a `River` as shown in Figure 1. To test whether a feature is open or closed, check the first and last Lat/Lon points – if they are the same position, then it's a closed feature, while if they are not the same, they are an open feature (multi-segment line).

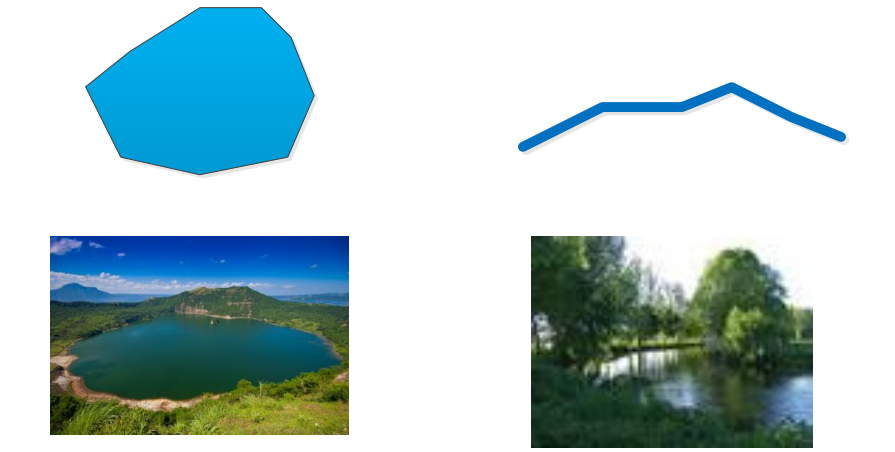

Figure 1: Some features are closed polygons like lakes, while others are a series of points like a river.

You can also ask for the name of a feature using `getFeatureName(id)`. Some features have names like "High Park" while others do not have a name and will return a string like "<noname>".

4.2 More Data: OSM layer 1 API

You can complete all the required features of this milestone using the higher-level "StreetsDatabaseAPI.h" interface which you used extensively in Milestone 1. In this milestone you can **optionally** also use the lower-level "OSMDatabaseAPI.h" interface to access additional OSM data. This data is generally less structured so it will require more investigation to determine what data you want and how to use it, but it can allow you to add extra features to your map that will make it stand out. One such feature would be showing the subways in a city. To learn more about how to use the layer 1 OSM data, including code examples of how to extract subway data for a city, see the ECE 297 Quick Start Guide on OSM.

Read to the "ECE297 Quickstart Guide: OSM" for information on how to get more data on a city.

5 Additional Resources

An extra feature you could implement is accessing data from the web and displaying it on your map.

See the libcurl Quick Start guide for information on how to access data on the web from a C++ program.

6 Grading

This milestone is open-ended so you are encouraged to be creative in your implementation. You will demo your map to your TA in the week after this lab is due, and your grade will be determined by your TA based on the quality of your implementation.

One part of your mark will be determined by **basic functionality**. This mark is mostly based on how well you implented the required features of Section 3. This mark will also include automatic testing of your programs ability to load, draw and close maps cleanly through `valgrind` via `ece297exercise`.

Another large part of your grade will be based on **usability and aesthetics** – how user-friendly your map is and how good it looks. Is it is easy to find information using your map, and does it respond quickly to user input? Can it show all the relevant data about map items to users who want detailed data, without overwhelming users looking for the big picture when they are examining large parts of the map? Is the most important and appropriate information easily visible at every zoom level?

Other marks will reflect the quality and ambition of your **extra features**.

Finally, there are marks for **project management and code style**. Project management will include how well you divide up tasks and track them on the wiki and how well you use git for continuous integration.

- **5 marks:** Required functionality marks.
- **4 marks:** Usability, responsiveness and aesthetics marks.
- **4 marks:** Extra features marks. Bonus marks possible for outstanding implementations.
- **2 mark:** Project management and code style

See the rubric on quercus for more detailed grading information.