

# Software Engineering Project

Cornella, Garcia, Kozlovskis, Pak

University of Burgundy

7 of January of 2014

# Outline

# Tools and resources

## IDE

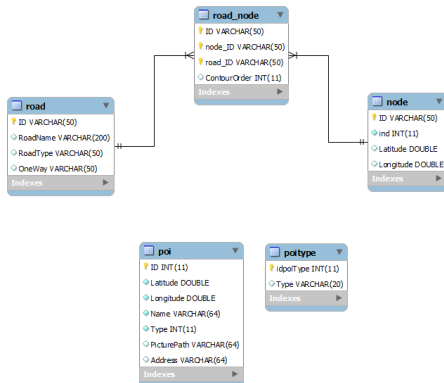
- Qt 5.1 with OpenGL
- Matlab 2013Ra

## Group meetings and coordination

- Trello
- Git and Bitbucket

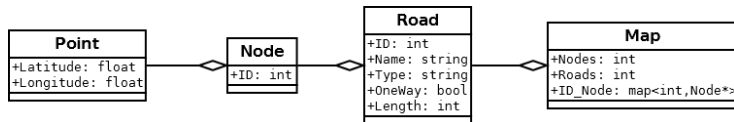
## Database

- MySQL Server 5.6
- PostgreSQL



# C++ Implementation

- Main Structure:



- Loaded in memory through QMYSQL Driver
- Displayed using GL\_STRIPES of OpenGL
- Creation of a map of ID\_Node\* because of the complexity
  - Avoiding Queries
  - Avoiding double loops

# C++: Dijkstra's algorithm

- Other kind of algorithms:
  - Bellman-Ford and SPFA
  - Johnson's algorithm
  - Floyd-Warshall algorithm
  - Dijkstra's algorithm
- Why? Performs better with non negative values
- How it works?
  - 1 Initialize D to 0 in the diagonal and  $\infty$  in non connected nodes
  - 2 Suppose that  $a = x$  (current node)
  - 3 Check all adjacent nodes of  $a$  except the ones that are marked
  - 4 If  $(D_i > D_a + d(a, v_i))$  then,  $D_i = D_a + d(a, v_i)$
  - 5 Marked as completed the node  $a$
  - 6 We take as next current node the smaller in D
  - 7 Go back to step 3 until there are nodes not marked

# C++: Adjacency Matrix

- Sparse Matrix of the Euclidean distance between nodes (Eigen library)

0	3	0	0	0
22	0	0	0	17
7	5	0	1	0
0	0	0	0	0
0	0	14	0	8

Results measured  
under Debug mode  
and dependent on the  
performance of each  
processor

Values:	22	7	3	5	14	1	17	8
InnerIndices:	1	2	0	2	4	2	1	4

Table 1: Sum up of the benefits of Sparse matrices

	Matrix	Sparse
Num. nodes	$\simeq 84 \cdot 10^6$	$\simeq 16 \cdot 10^3$
Memory	$\simeq 336$ MB	$\simeq 64$ KB + Zero vector
Time	>30 sec	<1 sec
Dijkstra	>30 sec	<3 sec

# Route path

- Functions:

- Path calculation
- Distance and time
- Export

- General case:

Same road  $\rightarrow$  NO  $\rightarrow$   $Angle < 180 \rightarrow$  Right

Same road  $\rightarrow$  NO  $\rightarrow$   $Angle > 180 \rightarrow$  Left

Same road  $\rightarrow$  YES  $\rightarrow$  Distance

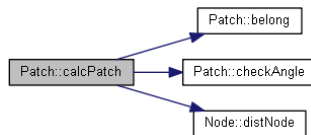


Figure : Functions involved in the route path calculation

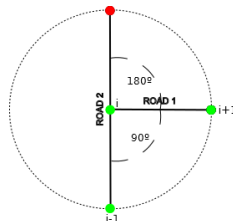


Figure : Example of road intersection

# Heuristics

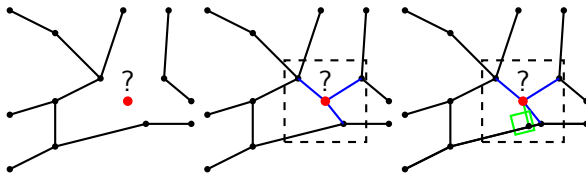


# Normalization and displaying

# Selecting a route

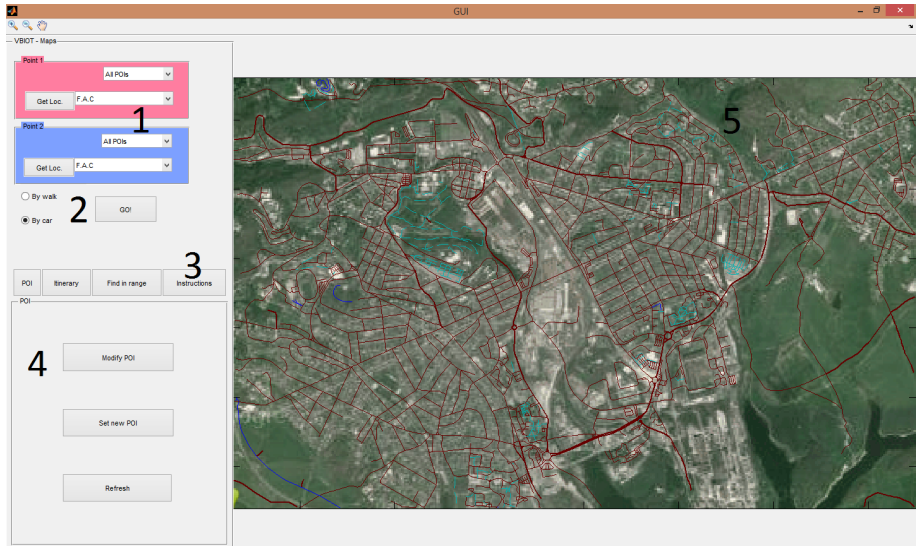
# Matlab implementation

- Data Structure
- Shortest Path



**Figure :** Finding the shortest point/projection for the random point on window domain

# Graphical User Interface (GUI)

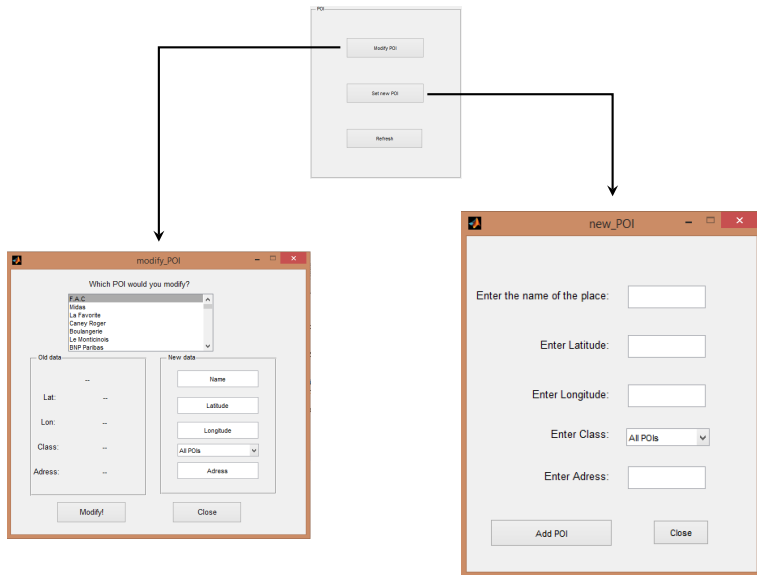


# Insertion of points

- Point from a list
- Filter by class
- Get Location from the map
- Walk / Car
- GO!

The screenshot shows a web application interface for inserting points. It consists of two main panels, 'Point 1' (pink) and 'Point 2' (blue), each containing a 'Get Loc.' button and a dropdown menu showing 'F.A.C.'. Below these panels are radio buttons for 'By walk' and 'By car' (selected), and a 'GO!' button.

# Manipulating Points of interest



# Itinerary

- Adding a third point
- Getting and storing the distance
- How it works
- Cost of time

The screenshot shows a window titled "Itinerary". Inside, there is a section labeled "Point 3" which is highlighted with a green background. This section contains a dropdown menu set to "All POIs" and a "Get Loc." button next to a text field containing "F.A.C.". Below this section, the text "Maximum distance: (in meters)" is followed by a text field containing "--". At the bottom of the window is a "GO!" button.

# Find in a Range

- Using Point 1 data
- Selection of classes
- Distance
- Slow process

Find in a range

Select your current location by using POINT 1 in the upper left corner

What do you want to find?

Maximum distance: (in meters)



# Generation of the instructions

- Generating instructions from shortest path
- Setting a scroll bar
- Exporting this data
- Default text when missing data

