On Map-Centric Programming Environments (Vision Paper)*

Walid G. Aref Sunil Prabhakar Jaewoo Shin Ruby Y. Tahboub Aya Abdelsalam
Jalaleldeen W. Aref
Purdue University, West Lafayette, Indiana
{aref,sunil,shin152,rtahboub,jaref}@cs.purdue.edu, aya.abdelsalam.91@gmail.com

ABSTRACT

2D maps and 3D globes can be used as programming toys to help students learn programming in contrast to using robots, visual, or multimedia components in Computer Science I introductory programming courses. This paper studies research challenges related to supporting this concept, and presents one instance of a 2D and 3D map-based programming environment to motivate these challenges.

Categories and Subject Descriptors

K.3.2 [**Computer and Information Science Education**]: Computer Science Education; D.2.6 [**Programming Environment**]: Map-integrated Environment.

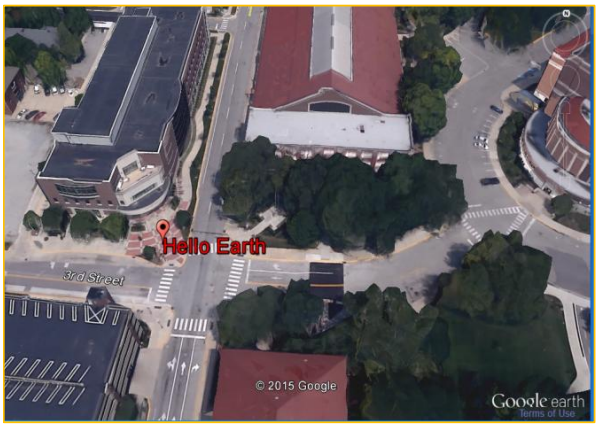
General Terms

Design.

1. INTRODUCTION

Academic institutions have been challenged to develop innovative methods to increase the interest of high-school students in learning Computer Science as well as to improve retention among Computer Science undergraduate students. Nowadays, computer literacy among the incoming student population has increased significantly due to the ubiquity of smartphones and tablets as well as the ubiquity of computer gaming and social-networking applications. Computer programming education needs to go beyond the boring nature of the freshman Programming I courses to address this change in the demographics of the incoming student body. A first program that prints "Hello World!" on the screen is not thrilling anymore to students playing computer games with high-quality graphics since kindergarten. To add excitement, several colleges have redesigned their freshman Programming I courses to use toy devices, e.g., Robots [8], or multimedia [7]. The target would be to program the robot to perform certain tasks while learning various programming concepts in the process. Although interesting to some students, using robots has several disadvantages. High setup cost in obtaining and maintaining the robots is one set back. Another disadvantage is the frustration that many students have when dealing with hardware as well as software debugging issues simultaneously.

In this vision paper, we propose to use 2D and 3D navigational spaces, e.g., Maps, Globes, and the Solar System, as the "toy" through which freshman CS students learn programming concepts. Besides the low setup cost, learning using 2D and 3D maps adds the excitement needed by introducing high-quality graphics and high interactivity by relating the programming tasks to maps or photos of locations that the students can relate to. Moreover, learning programming using maps and globes can be very suitable to students and researchers in the area of Spatial Computing [6] and Geographic Information Science and Systems (GIS) [9]. Also, learning to program using maps and globes is more appealing when developing curricula that promote Computational Thinking [10, 1].




```
1. address = read_address("305 N University St",
   "West Lafayette", "IN", "47907")
2. display_message("Hello Earth", address)
```

Figure 1: The visualization of a "Hello Earth" program in LIMO using Google Earth.

*Walid G. Aref's research is partially supported by the National Science Foundation under Grants IIS 1117766 and IIS 0964639.

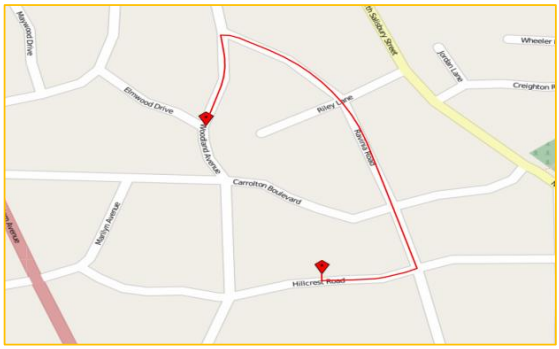
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
SIGSPATIAL '15, November 03-06, 2015, Bellevue, WA, USA
Copyright 2015 ACM 978-1-4503-3967-4/15/11
http://dx.doi.org/10.1145/2820783.2820886 ...$15.00.

Following this vision, we have prototyped *LIMO*, a web-based map-centric visual environment that is designed around the activities of moving objects on 2D and 3D versions of a map. LIMO's current version supports interfaces to Google Maps (2D) [3] and Google Earth (3D) [2]. In LIMO, a set of simple programming constructs are made available to the student programmer to build programs of various degrees of complexities. LIMO offers script and visual interfaces for programming so that as the LIMO programs execute, their animations are displayed simultaneously on the visual interface. In addition, LIMO has simple programming interfaces that provide access to the underlying road network as well as a points-of-interest (POI) dataset extracted from OpenStreetMap [4]. A backend PostgreSQL [5] database is used to store the road network and POI databases.

1.1 Example Map-based Programs

Program 1. **Hello Earth**. Figure 1 demonstrates a "Hello Earth" program that displays "Hello Earth" on address '305 N University St, West Lafayette, IN 47907". The program uses a function termed *read_address* that takes as input a textual address where the *"Hello Earth"* message needs to be placed. Then, the program displays the message on the map using a *display_message* function.

Program 2. **Following Directions**. Figure 2 gives a simple program that starts from one address and follows directions until it reaches a certain destination. The program informally illustrates the use of several LIMO programming constructs, mainly start_at address, move_distance, turn, move_to_next_intersection, and move_until_intersection.




```
1. address = read_address("700 Hillcrest Rd",
"West Lafayette", "IN", "47906")
2. display_marker(address)
3. start_at("com1", address,"EAST")
4. move_to_next_intersection("com1")
5. turn_to("com1", "Ravinia Rd", "NORTH")
6. move_until("com1", "Woodland Ave")
7. turn_to("com1", "Woodland Ave", "left")
8. move_distance ("com1", 0.09)
9. last_location = get_current_point("com1")
10. display_marker(last_location)
11. show_on_map("com1")
```

Figure 2: A simple LIMO program for following directions to reach from one address location to another using OpenStreetMap.

Program 3. **Locating Nearby Airports**. Figure 3 gives a more elaborate program that uses the underlying Point-Of-Interest (POI) database to retrieve the nearby airports to a given focal point (a k-nearest-neighbor operation). The program illustrates the use of a *getall* function to retrieve the POIs from the underlying database. The k-nearest-neighors are identified by simply looping over all airports in a given area (in this case, the State of Indiana) and computing and displaying the distance between the focal point and each airport. The top five airports are reported and are marked on the map (Refer to Figure 3).

The examples above demonstrate the use of several LIMO programming constructs. Although not the focus of this paper, to give the reader a more holistic view, other LIMO programming constructs are briefly listed in Table 1.

2. CHALLENGES

LIMO is one example of a map-centric programming language. It is a proof-of-concept to demonstrate the feasibility of having 2D and 3D maps as the underlying toy for learn-

Table 1: Sample LIMO Programming Library

Category	Function Name	Description/ Options
Map (Basics)	start_at(name, address, direction)	Sets Commuter's start location to address with heading direction
	move_distance(name, distance), move_until(name, street), move_to_next_intersection (name)	Move Commuter for certain distance or until a clear street/intersection
	turn_to(name, streetName, direction)	Re-orient the commuter towards a new street and direction, e.g., 123 University St. and South
	display_message (message, address|location)	Place a text message, e.g., at a given address or at a geo-location
	display_marker (address| location)	Place a marker on the map at a given address or a geo-location
	display_[distance| time](commuter)	Display total distance/time commuted so far
	compute_distance(add1, add2)	Return the distance between two addresses or geo-locations
Spatial	get_location(address)	Return a point that represents the geo-coordinate of the given address
	get(name, description, geometric shape)	Return the location (as geometric shape) of the place that matches place-name and description
	get_all(description, geometric shape)	Return a list of locations (as geometric shape) for places that match the input description
	overlaps| touches| intersects| contains (shape1, shape2)	Boolean operators that test whether two shapes: overlap, touch, intersect, or contain one another
	display_shape (geometric shape)	Display geometric shape (e.g., lake boundary) on map

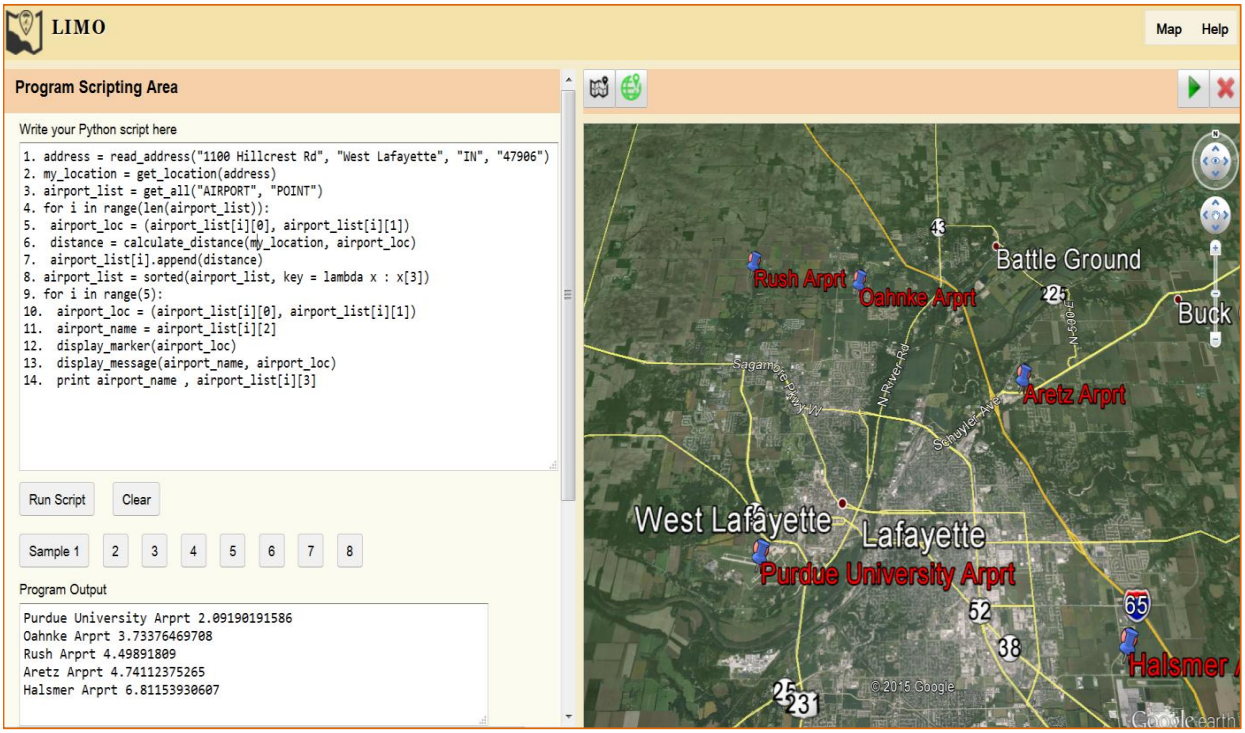


Figure 3: Example LIMO program to find the nearest five airports to a given textual address. Also, the figure illustrates the various components of the LIMO programming environment, mainly a *program scripting area*, *a program output area*, and an *interactive map*.

ing computer programming. However, as can be obvious already, there are many research challenges that need to be addressed. There are expressiveness challenges, interfacing challenges, and performance challenges. We highlight these challenges below.

2.1 The Expressiveness Challenge

2.1.1 *Proper Design of the Programming Constructs*

The question is: What are the proper programming constructs to have in a map-centric programming language? The programming constructs should be: (a) Ease to use, and (b) Complete. These programming constructs should fully address the map navigation functions as well as seamlessly integrate and weave the POI database into the programming language.

2.1.2 *Defining a Minimal Set of Programming Constructs*

The challenge here is to define a minimal set of map navigation programming constructs that are both sound and complete. It is important to balance that with the ease of use of the programming constructs. An important issue is to draw a line between what makes a primitive programming construct and what is left for the programmer to realize. For example, is shortest path a primitive? or should LIMO provide the constructs to be able to realize one shortest-path algorithm? Similarly, is the k-nearest-neighbor (kNN) operation a primitive or should the programmer realize a KNN algorithm from a set of provided primitives? The cutoff or boundary between what a primitive programming construct is and what is not needs to be studied.

2.1.3 *Data Types*

Identifying the types of the spatial objects that the map-centric programming language will handle is important. In addition to the regular geometric types, e.g., points, line segments, poly-lines, etc., how will the programming language deal with the temporal aspects of these types, e.g., trajectories of moving objects, and the programming constructs that operate on the trajectories? While being comprehensive is good, this may make the programming language hard to learn. Also, how text data interplays with map and location data is important to address. LIMO tries to address this issue with the programming interfaces that it provides, but this needs to be studied more thoroughly.

2.2 The Interfacing Challenge

2.2.1 *The Connection Between the Textual and the Map Interfaces*

Designing the visual interface of the programming language and the degree of interaction between the textual programming interface and the map interface are important, e.g., when the user clicks on the map and how this affects the programming interface. Also, as the programs execute, when do the effects of the execution of the programming primitives get reflected on the map? In other words, it is

important to utilize the map as both an input and output devices to the map-centric programs.

2.2.2 Identifying the Various Types of Visual Map Interfaces and their Effect on the Programming Constructs

In contrast to limiting the interface to 2D and 3D maps, other possible choices of visual map interfaces are a galaxy-level visual interface, an inter-galactic visual interface, or a 2D or 3D indoors visual interface. Because the way the navigation is performed in each visual interface may differ, new programming constructs may be needed for each visual interface.

2.2.3 Concurrent Visual Map Interfaces

Also, it is feasible that the programming environment permits a combination of concurrent multiple 2D and 3D map interfaces. Defining how the programming interface will interact and will make use of these concurrent interfaces is challenging. However, allowing multiple concurrent map interfaces will also enable concurrent programming exercises.

2.2.4 Database Interfacing

The types of the underlying databases that the map-centric programming language can have access to may vary. For example, one map-centric programming language may allow access to only a 2D road-network map database. Another may add a Points-of-Interest (POI) database, e.g., shops, malls, airports, clubs, etc. Other possibilities are 3D glode databases (includes road networks, countries, continents, oceans, etc.) or a galactic and intergalactic sky database. The challenge is to provide a uniform and simple interface that would allow each of these datasets to be integrated seamlessly into the map-centric programming language, and explore whether specific programming constructs will need to be tailored for each underlying database or not.

2.3 Performance and Productivity Challenges

2.3.1 Modeling location and geocoding text addresses

Location information can be either presented using a textual address or a geo-coordinate. The mapping between the two in both directions is a core operation in a map-centric programming language and should be supported in a seamless and very efficient way.

2.3.2 Response Time and Interactivity

Having real-time response to all operations is important. The interaction between the visual interface and the underlying databases that are accessed by the programming language constructs need to be very efficient. Approximate vs. exact execution of these constructs needs to studied in light of performance and real-timeliness needs. Also, if approximate techniques are to be used, how will this affect the programming language interface? For example, should the programmer specify the level of approximation or confidence-level guarantees while coding with the map-centric programming language?

2.3.3 Novel Debugging Tools

Designing a proper debugger for debugging the map-centric programs with proper interactions and interrogations between the textual and visual map interfaces is an interesting subject. New single-step modes and map-based breakpoints and stopping criteria at landmarks may need to be defined. For example, the program should stop if a variable in the program reaches a certain geographic location, and similar map-based debugging criteria and constructs need to be developed for map-centric programming languages.

2.3.4 Programming Paradigms

It is important to study how map-centric programming languages can be used to teach the various programming paradigms, e.g., procedural vs. object-oriented programming languages.

3. CONCLUSIONS

Based on practical experience with the LIMO system, we realize that map-centric programming languages are interesting and important. They can help Computer Science students as well as the spatial computing community. It would be important to identify other applications of the map-centric programming languages beyond teaching how to program, e.g., as a simulation tool, for verification of directions, for the visually impaired, and traffic and dynamic constraints.

4. ACKNOWLEDGEMENTS

The authors would like to thank the anonymous reviewers for their constructive comments. The support by the National Science Foundation under Grants IIS 1117766 and IIS 0964639 is greatly appreciated.

5. REFERENCES

[1] Computer science curricula 2013. `https://www.acm.org/education/CS2013-final-report.pdf`.

[2] Google earth. `https://www.google.com/earth`.

[3] Google maps. `https://www.google.com/maps`.

[4] Openstreetmap. `http://www.openstreetmap.org`.

[5] Postgresql. `http://www.postgresql.org/`.

[6] P. Agouris, W. G. Aref, M. F. Goodchild, E. Hoel, J. Jensen, C. A. Knoblock, R. Langley, E. Mikhail, S. Shashi, O. Wolfson, and M. Yuan. From gps and virtual globes to spatial computing -2020. In *The next transformative technology, computing community consortium (CCC) workshop proposal.* *http://archive2.cra.org/ccc/files/docs/spatialComputing_2.pdf*, 2012.

[7] M. Guzdial. A media computation course for non-majors. In *ACM SIGCSE Bulletin*, volume 35, pages 104–108, 2003.

[8] T. Lauwers and I. Nourbakhsh. Designing the finch: Creating a robot aligned to computer science concepts. 2010.

[9] P. A. Longley, M. F. Goodchild, D. J. Maguire, and D. W. Rhind. *Geographic information science and systems*. John Wiley & Sons, 2015.

[10] J. M. Wing. Computational thinking. *CACM*, 49(3):33–35, 2006.