Java Map Mouse Control

CIS422 Project 1 (prototype), Fall 2007

Michal Young

Preliminary version 1.0 of 24 September 2007

Abstract

A haptic soundscape map is a way of making a map accessible to blind people using sound and some form of touch (force feedback, vibration, etc.) in place of visual data. CIS 423/510 students, working with geography students, created a haptic soundscape map of the University of Oregon campus in spring 2007. While largely successful, one of the important outcomes of that project was the realization that using Flash as the presentation medium was not compatible with some proposed interaction techniques. This raises the question: Could the map be presented using Java graphics libraries? What are the consequences for performance and for application size and complexity? In particular, can a reasonably simple implementation in Java track the mouse cursor and trigger sound and haptic effects in real-time as the cursor sweeps across a complex map? The purpose of this project is to resolve that question with a prototype implementation.

Note: This document is incomplete. Revisions will provide more information about the earlier map implementation (from which you may scavenge code as well as data) and more detail about what you must actually turn in.

Contents

1 Introduction

1.1 Haptic soundscape maps

Blind people need maps as much as sighted people do, or perhaps more: They are less able than sighted people to use street signs and other visible indications to find their way around in an unfamiliar location. Conventional maps, however, are not accessible to blind people.

Most maps designed to be accessible to blind people use one of two approaches, *tactile* mapping or *haptic soundscape* mapping. *Tactile* mapping produces raised impressions ("bumps") on a physical medium. Although there are still many open research questions regarding tactile mapping, it is well enough understood that standard tactile symbols are being developed, and tactile map production software is under development (by the research group of Professors Lobben and Fickas at University of Oregon).

Sighted people increasingly make use of interactive maps, but current tactile output (with the exception of small Braille displays) are static. *Haptic soundscape mapping* is an attempt to provide interactive maps that use the senses of hearing and touch in place of the sense of sight. The haptic (touch) element could be force feedback or texture conveyed through vibration, as some gaming devices. Sound can include speech sounds (for example, pronouncing the name of a street or building under a map) or non-speech sounds, which could be symbolic (e.g., traffic noises to indicate a street) or non-symbolic (e.g., middle C played on a piano to indicate a doorway).

The research supporting haptic soundscape maps is much less well developed than that supporting tactile mapping. Several experimental maps have been created at University of California, Santa Barbara, and a few have been constructed by research groups elsewhere, including University of Oregon. We cannot say that haptic soundscape maps are a good way or a practical way of providing interactive, computerized maps to blind people. What we can say is that more research is needed to understand whether and how haptic soundscape maps can be useful, and that research necessarily involves constructing maps for evaluation.

The overall purpose of this project is to support research in haptic soundscape mapping, which may in turn enable production of really useful maps that use haptics and/or sound.

1.2 The UO Campus Soundscape Map

In Spring 2006, CIS 423/510 students, in collaboration with graduate students in Geography, created a haptic soundscape map of the University of Oregon campus. I believe it is the first such map to largely automate the map production process, taking map data from a GIS database and applying sound and haptic effects specified in a style file to produce Flash files. Automation and flexibility are important in a research environment, where it is necessary to produce and evaluate variant maps to test different ways of using sound and touch.

The software produced in Spring 2007 is a "proof of concept" prototype. I consider it very successful, because we demonstrated (to ourselves and to our collaborators) that we could entirely bypass the usual steps of editing the map in a drawing program like Adobe Illustrator. We also discovered some problems in the approach we took, which we would not have discovered without actually building it.

One problem is that, while we were able to automatically generate all the ActionScript[1] code to produce a campus map in Flash, we did not eleminate the (trivial, but necessary) step of compiling the ActionScript code into a runnable Flash object file. This made us dependent on an (expensive) piece of commercial software to produce maps. Sharing a single floating license (and only for the Windows platform) was a frustrating bottleneck in development. Recently Adobe has distributed a free compiler for the most recent version of ActionScript, so this problem may resolve itself.

The second problem with using Flash as the presentation medium seems to be more fundamental. One of the interface techniques devised by members of the CIS423 team involved a kind of panning in which the mouse cursor remains still while a (potentially very large) canvas moves under it. This was intended to avoid potentially frustrating problems when the mouse left the main map canvas. Unfortunately, it turned out to be impossible to implement the auto-pan technique in Flash displayed in a browser window, because the Flash executes in a sandbox environment and cannot modify the mouse cursor position.

Both of these problems, but particularly the second, led the student team to consider whether it might be advantageous to use a Java application, rather than a

[1] ActionScript is the scripting language embedded in Flash. It is a dialect of ECMAscript, better known for its JavaScript dialect. JavaScript and ActionScript were once nearly identical except that JavaScript included Document Object Model (DOM) libraries for HTML and ActionScript included DOM libraries for Flash graphics. Now they have evolved somewhat divergently, but the core ActionScript language still looks familiar to JavaScript programmers.

Flash movie, as the medium for presenting the map.

1.3 Prototype goals

The purpose of this prototyping exercise is to learn whether using Java graphics libraries is really a practical alternative to presenting a map with a presentation engine like Flash, an SVG viewer, Silverlight, etc.

The Flash player (or an SVG viewer plug-in, or presumably Microsoft's new Silverlight players) contains high quality, optimized graphics facilities. There is a danger that using Java graphics libraries instead might cause noticeable performance lags. There are at least three ways performance could be noticeably worse.

- Drawing a complex map could be slower using the Java graphics library than using a Flash movie. An additional lag of more than a second would be noticeable. A lag of more than 5 seconds (on a current but modestly provisioned desktop or laptop computer) would be unacceptable.
- A Java program to display a map might be considerably larger than a corresponding Flash movie, causing a noticeable communication delay. This is a less urgent problem than slow drawing, because it might be acceptable to separate downloading or otherwise acquiring a Java-based map from displaying and interacting with it.
- A Java program might be too slow at hit detection, that is, triggering the appropriate effects (sounds, haptic effects) as the mouse enters and leaves shapes on the screen. (Hit detection is closely related to collision detection, and can be considered collision detection between an object representing the mouse and other objects on the canvas.)

In addition, the auto-pan feature might be difficult to implement or might not perform acceptably.

1.4 Priorities

1.4.1 Hit detection performance

The first priority for the Fall 2007 prototype is to evaluate the feasibility of hit detection in Java for interactive maps.

- For a soundscape map of the same complexity as the UO campus haptic soundscape map, can a Java application track the mouse cursor and determine which objects it is moving over quickly enough to play the appropriate sounds even when the mouse is moved quickly across the screen? How quickly?
- If performance is acceptable with the UO Campus soundscape map, how does it scale with larger numbers of shapes or more complex shapes? What is the practical limit on number, density, or complexity of shapes, before hit detection becomes too slow or its support data structures become too large? A working definition of "too slow" is inability to determine which shape the mouse cursor is over at least 10 times per second on a typical laptop computer.

1.4.2 Auto-pan

The second priority is to demonstrate the feasibility of auto-pan, in which mouse cursor movement is limited (it may either be fixed, or move only within a limited sub-area of the display window) and the map canvas moves under it in a direction contrary to mouse movement (so that the net effect, with respect to mouse cursor position on the canvas, is the same as it would be without panning, if the window were as large as the canvas).

- Can auto-pan be implemented in Java?
- If so, does it have any unanticipated negative effects (e.g., does it interfere with hit detection?)

2 Background: What's a prototype for?

The purpose of a prototype is to reduce risk by obtaining information. Often a prototype is an early version of an actual application (an "alpha" or "beta" release), but not always. It is crucial to first consider what information a prototype is designed to obtain, and only then settle on the approach and content of a prototype.

2.1 Requirements risks

One common and important kind of risk in application development is requirements risk. If I build for this client exactly what she described in the meeting, will she hate it anyway? Requirements risk is highest when building some really novel piece of software, and lowest when building software that is very routine and commonplace. If the project is to build an Excel work-alike for Linux, requirements risk is low because our client and we have a good understanding of what is needed to be similar to Excel. (There might still be requirements risk, because the user audience on Linux is not the same as the user audience on Windows, but it's probably an acceptable risk.) If, on the other hand, we are building some entirely new class of application that no one has used before, then requirements risks are very high. It is very likely that what one imagines will be useful and what is actually useful are not quite the same.

Prototypes to control requirements risks are designed to help users and designers discover the actual requirements for an application. They provide some, limited functionality, sufficient to discover flaws and incompleteness in the original conception of the application. They may or may not be early versions of the actual application — often they are, but other times the most efficient approach is to build a simple throw-away prototype to play with, perhaps using a different programming language and libraries than are required by the eventual production version of the application.

2.2 Interaction risks

Even if we have a firm grasp on the essential functionality of an application, there is a risk of not getting all the details of interaction right. In fact, for any piece of software used directly by people, user testing and iterative refinement are essential for usability. (Take CIS 443/543 to learn more about this.)

Interaction risks are related to requirements risks, but they are not quite the same, and techniques for risk control are not quite the same either. Often a good "prototype" for early usability testing is something completely fake — even rough sketches of screens on paper, or mocked up with a drawing program. As development continues, it is important to also perform user testing with actual running versions of the software, but that is seldom needed to get started.

2.3 Development risks

A third important class of risk is in the development process itself, especially with respect to schedule. You can consider any milestone in the development process at which running software is demonstrated to be a kind of risk reduction prototype. A demo of running parts of the project (if planned correctly) reduces the likelihood that our schedule is wildly optimistic. We'll talk a lot about this in CIS 422/522.

2.4 Technical risks

Finally, there are risks that are purely technical in nature. Can this database package really handle the quantity of data we'll encounter? Will distributing the application in this way across servers lead to unacceptable performance, or open us to security holes? Does the printing library work the way we think it does?

Every software project involves hundreds or thousands of small technical risks. Most are small enough that we simply resolve them as we go, without much worry. A few are more significant, and it's important to anticipate them and reduce the risk early. It is not a happy even when we discover late in a project that the approach we have chosen is simply not going to work.

As with requirements risks, technical risks are smaller or greater in proportion to the degree of novelty in a project. However, technical novelty and novelty in the application as seen by the user are not the same thing. I might use libraries I know forward and backward to produce an entirely new kind of application, or I might produce a very conventional application using a technique I have never tried before.

It is often useful to build prototypes to assess technical risks. A prototype for assessing a technical risk needs to be just realistic enough for that assessment. It need not be useful. It need not have all or even many of the features of the real product. On the other hand, it often needs to be scalable and instrumented in ways that the final product is not, to answer questions about the range of conditions under which the approach taken in the final product will work.

Our current prototyping exercise is of this kind. There are specific technical risks that need to be assessed. It is not important that the prototype be a useful map, but it is important that it tell us whether it will be feasible to implement useful haptic soundscape maps in Java. Since there is no reason to think that the answer depends on the nature of the feedback triggered as the mouse moves over objects on the canvas, we can answer it with sound feedback alone, leaving aside haptics for the moment. On the other hand, in addition to the UO campus map,

we should produce (possibly random) collections of shapes that are much larger or have many more shapes, and collections that we think might be "bad" in some significant way. For example, unless we can be certain that the distribution of shapes on the canvas doesn't matter, we should consider the case where many, many complex shapes are stacked in one part of the map.

2.5 How to win by losing

Answers to the questions posed in this exercise might be negative. Maybe the answer to "Can auto-pan be implemented in Java?" is "no". Maybe the answer to "can a Java application track the mouse cursor and determine which objects it is moving over quickly enough . . . " is "no". What then?

Of course we prefer a positive answer, perhaps of the form "Not quite like that, but if . . . then we can . . . ". But we may not get even that. What then?

The value of a negative answer comes from being clear, convincing, and loaded with information that can be used in finding alternative approaches. "I didn't find a way to make it fast enough" has some value, but not a lot. "The cost rises quadratically with the number of sides in each shape, because the floomflazzle library compares all pairs of sides" is much more valuable. If you appear headed for a negative result, make it worthwhile by finding a definitive explanation of *why* something doesn't work. Insight into possible workarounds is also good.

3 What is Required

For this prototype project, what I require is an assessment of at least one of the two prioritized questions above. This assessment should include running code, but it need not be a usable map. I will not require a user manual. I will, however, require a write-up that provides a good analysis of your design.

If the answers to the questions are positive, then I sugggest (but do not require) that your prototype also contain code and design documentation useful to developers producing a full haptic soundscape map. I will suggest (but not require) that teams use the second half of the academic term to produce such maps, or more precisely, tools for creating such maps. Teams that produce reusable code in addition to a good assessment of the performance and capability questions will have a big head start on the final project.

4 Technical notes

Here are some random notes on how you might approach this. These are *not* requirements.

Beware non-simple shapes. Look at Bean housing complex on a campus map — it looks like two interlocking rings. There are different ways to represent such complex shapes. The way used by ESRI software (which is the dominant software used in professional mapping) seems to use the direction of rotation (clockwise or counterclockwise) to indicate whether a shape indicates donut or hole. I do not know if this can be easily reconciled with the winding rules supported by Java graphics libraries.

Consider z-order. If there are two or more objects under the mouse, the one upper-most is the one that matters. You don't need three dimensions for this. It's sometimes called "2.5d graphics", i.e., 2 real dimensions plus order of stacking.

Divide and conquer. java.awt.geom.Area has collision detection methods ("contains", "intersects", etc), and ultimately you probably want to use these, but you can't possibly afford to ask every shape on the screen whether it contains the cursor. A fairly standard approach to such problems is quadtree data structures, but if you use quadtrees, you need to be particularly careful of the case where many objects are stacked in exactly the same place. If you google around, you'll find that some people advocate a fixed grid arrangement (sometimes called a "spatial hash") instead, but it seems to me that a spatial grid will have the same problem with large and irregular shapes.

Here's your worst nightmare for a simple-minded quadtree structure: In the campus map of UO, there is really only one street object — it's one big street-thingy representing all the streets on and around campus. Therefore, if you tried to use its bounding box to place it just in the elements of the grid that it could lie in, it would be in *every* grid element. So you'll have to do better.

Jumping the mouse. The robot class seems to be able to do this, but it will also generate an input even that appears as if the user had made that mouse move. I'm not sure if there is a better way, or a good way to cope with these bogus input events.