Schematizing Maps: Architectural Organization for an Initial Version
Addendum for Distributed Development

Michal Young
University of Oregon

DRAFT version of 2010.03.11

Abstract

In this addendum to the SCORE project description, we describe a particular application domain (specializing the general class of schematized maps) and an organization of the system for development as an incremental series of releases. The organization is elaborated for the first, minimal increment, which is divided into modules suitable for development by a team that is divided into two widely separated sub-teams.

Introduction

The “schematizing maps” project for SCORE 2011 asks for an aid in constructing a map in a non-uniform scale so that more detail is provided for some areas and less space is devoted to other portions of the map. A schematized map may also be simplified in other ways, such as simplifying lines and limiting their orientations. This document considers one possible way schematized maps might be produced (by tracing over the image of a standard map). It briefly describes a sequence of incremental steps toward such a system, and an architectural organization for delivering those incremental steps, and then considers in more detail the modular organization of an initial, minimal version that can be produced in a few weeks by a distributed team of students.

Application domain: Trace maps

The “schematizing maps” project for SCORE 2011 must be specialized to a particular audience, and suited to ways in which that audience will use a schematizing tool. We
also need to define what the system will actually do (that is, the solution rather than the problem it addresses) to define an architectural structure for the system. While doing so, we would like to identify core functions that can be used to build a family of related systems.

This document describes one such specialized system. The initial audience addressed by this system is organizers of long-distance bicycle events.\(^1\) The route of such an event typically includes some relatively long stretches with no turns, and also some areas (e.g., in towns along the route) in which detailed directions are needed. Schematizing a map can help make more room for the complicated, detailed parts by shrinking some of the parts that are straightforward.

Except in a few very large bicycle clubs and very large events (RAGBRAI in Iowa, STP in the U.S. Pacific northwest), people who prepare bicycle route maps are not professional cartographers and do not have access to professional cartographic tools like ESRI ArcGIS. Thus, the map schematization system described here is designed to be used in conjunction with free or inexpensive GIS tools (e.g., Google Maps) and common drawing tools (e.g., Adobe Illustrator) or open source drawing tools (e.g., Inkscape).

A typical workflow might be:

- Capture a route with a GPS unit
- Use a tool or service like GPSVisualizer to display the route in Google Maps.
- Capture a screen-shot of the route and map display, and save it as an image.
- Place the image into a background layer of a drawing tool like Inkscape, Adobe Illustrator, CorelDraw, or OmniGraffle.
- Trace the route (manually) over the image of the map.

The system described here adds an image manipulation step between capturing a screen shot of the route, and tracing over it in a drawing tool. The idea is that, after the image has been manipulated, the user will draw over the manipulated version to create a schematized map. The manipulations will be shrinking of portions (stripes) of an image, in either the X or the Y dimension but not both, as suggested in the Schematizing Maps project description. Figure 1 sketches this workflow, with the initial system to be developed labeled as “Distort with image slicer.”

\(^1\) I believe the requirements will be very similar for organizers of long distance foot-races (marathons), triathlons, bicycle tours, and other organized events, but I choose bicycle events here because I am most familiar with them.
Initially the map designer produces an image of a geographic map, with or without a trace of the intended route. The initial version of the map schematizer support, described in this document, is used to slice and modify the map image. When the image has been modified by snipping out slices, it is then used as a base layer in another drawing program, such as Inkscape, Adobe Illustrator, or Omnigraffle.
Conceptual Model

In this section we introduce terms and an overall conceptual model of the trace map schematizing system, so that in subsequent sections we can more clearly and succinctly describe the organization of modules and the development effort.

A map is a representation of a rectangular region of space on the surface of the earth. In addition to its extent on the surface of the earth (delimited by four \((latitude, longitude)\) pairs, a map has an extent in a plane geometry, delimited by a coordinate pair \((x_0, y_0)\) and a coordinate pair \((x_1, y_1)\). The relation between the extent on the surface of the earth and the extent in the plane is called the projection of the map. Because every projection from the surface of a sphere onto a plane distorts some important geometric properties (relative sizes, orientation or straightness of lines, etc), there is no single “best” projection, and this in turn means that many different projections are in common use.

A map may be composed of multiple layers of information. We say a map is registered if a single projection applies to all layers, i.e., if the planar coordinate pair \((x, y)\) in each layer refers to the same \((latitude, longitude)\) position on the surface of the earth. Unless otherwise specified, we will assume that maps are registered.

Each layer of a map may be classified as a vector layer or a raster layer. While there are many possible representations of a vector layer, or of a raster layer, representations within each of these two broad classes are generally interconvertible. Vector representations can be converted to raster representations, but raster representations typically cannot be converted to vector representations. Vector representations can generally be scaled precisely, while scaling a raster representation typically degrades its accuracy. Scalable Vector Graphics (SVG) is a typical vector graphic format; ESRI Shapefiles (SHP) is a common vector format for geospatial information. Common raster representations include Jpeg, GIF, and Portable Network Graphics (PNG), as well as internal forms maintained by graphics systems like Java2D.

A map display is an image suitable for display on a graphical screen or on paper. A map display is not the same as a map. For example, a map display may include only portions of the extent of a map, and it may include all or only some of the layers of the map, combined in some way and further transformed for display purposes. The coordinate system of the map display may not be the same as the plane coordinate system of the map.

Converting a map into a map image is called rendering. Combining multiple vector and/or raster layers of a map into a single map display image is called rendering.\(^\text{2}\)

\(^2\) “Rasterizing” a vector representation often requires some additional control information, such as the width of lines.
Release 0 and Release 1

A map schematizer for bicycle event maps can be developed in a series of incremental releases, adding new features with each release. The modular structure of the system is intended to facilitate building one increment, then the next, and so on, without radical redesign or discarding large amounts of working code or documentation. We describe a design for Release 0 in detail so that teams can get started and organize the work quickly. We make a few hints about possible directions for Release 1, but leave choices of what to implement as well as how to implement it to students.

Release 0

The initial release should be implementable in a few weeks. It is intended to be the simplest useful system that can also serve as a building block for further development. The initial release is limited in several ways:

- It supports only one map layer, which is an image layer (for example, a screen-shot taken from a Google map).
- Since there is only one layer, there is no registration. The map projection and the coordinate system of the map are essentially arbitrary.
- The user interface is very simple.

Modules

Here is one way a trace map schematizing tool can be broken into smaller pieces that can be specified, implemented, and tested somewhat independently. Each of the modules can serve as a work assignment for one sub-team, and they can be incrementally assembled into larger and larger pieces of the overall product. The organization of these modules is illustrated in Figure 2, and further detail of the relations among some key modules is illustrated in Figure 3.

Map

The map module maintains an ordered set of map layers. Individual layers may be raster layers or vector layers. In the first version it has just one, raster layer (e.g., a screen-shot of a Google map). It has a projection (e.g., Mercator), but initially that
Figure 2: Overview of the “uses” relation among modules in the map schematizer (release 0). The dashed blue line illustrates how the organization corresponds roughly to a conventional model-view-controller pattern (the map is the model, the tile and display are a view of the model, and the user interface is primarily a controller).
A map is composed of an ordered set of layers. Each layer may be a vector layer or a raster layer.

The relation between planar coordinate systems with uneven transformations is separate from actual manipulation of graphics.

A tile is a raster representation produced by the "render" operation on a map, with an interpretation of its coordinates.

A portion of the "uses" relation among components of the map schematizer. A "tile" presents an interface suitable for the user interface to manipulate (which could include panning, scrolling, etc). The map itself manages a set of layers in a right-hand coordinate system, and composites them to create a tile. The coordinate transforms (both shrinkage of map rectangles, and mapping of left-hand integer tile coordinates to right-hand floating point map coordinates) is handled by the coordinate mapping module, which does not actually manipulate graphics at all. The Map and Tile modules depend on the graphics system (e.g. Java2D).

Figure 3: More detail on uses relations between services of selected modules.
is just an uninterpreted constant. It has an extent \((x_{\text{min}}, y_{\text{min}}, x_{\text{max}}, y_{\text{max}})\), and its coordinates are floating point numbers in a right-handed coordinate system (that is, \(y\) increases to the north, \(x\) increases to the east). In addition to the obvious "width" and "height" \((x_{\text{max}} - x_{\text{min}}\) and \(y_{\text{max}} - y_{\text{min}}\)), it can be queried for its effective height and width, after shrinkage. It has operations \(x\text{Shrink}(x_1, x_2, sfx)\) which applies scale factor \(sfx\) to the region from \(x_1\) to \(x_2\), where \(x_1\) and \(x_2\) are within the extent of the map and \(sfx\) is a floating point number between 0.0 and 1.0, inclusive, and there is an analogous operation for shrinkage in the \(y\) direction. There are operations for creating new layers, but initially there is just an operation for creating a single raster layer (perhaps from an image file). There is a rendering operation that creates a tile (described below).

The data structure(s) used for the map representation are a secret of the module. The number of layers in the map, their order, and their extent (in a right-hand, floating point coordinate system) is information available through the module interface. Other secrets of the map module include: How layers are managed, and most importantly, whether shrinkage operations are applied eagerly (to the layers themselves) or lazily (during rendering only). It also provides the algorithmic service of compositing layers into a tile.

Coordinate Transform

There is a module that manages plane coordinate transforms\(^3\) It is likely implemented as two distinct Java classes, but they are a single module because the secrets of the module are shared between them.

If a series of horizontal and vertical shrink operations (as provided by the map component) are recorded by the coordinate transform module, the coordinate transform module can provide an iterator over rectangular regions that have been uniformly transformed. It can also determine the effective width and height of a transformed region, or (inverting that relation) determine the original region that, after shrinking, now fits in a rectangle with a given origin and extent.

The coordinate transform module can also record a transformation between the real-valued right-hand coordinate plane of a map and the left-hand integer plane of a tile. When a tile is produced by rendering all or part of a map, this relation is recorded.\(^4\) The coordinate transform module provides a mapping from integer

---

\(^3\)Do not confuse these planar coordinate transformations with the projection from spherical coordinates on the earth (latitude and longitude) to the planar coordinate system of the map. We will try to consistently use the term "map projection" for the relation between spherical earth coordinates and planar map coordinates, and "coordinate transforms" for relations between planar coordinate systems.

\(^4\)Here we are not specifying the Java API for these operations in detail, so it may sound like the
coordinates in the tile to floating point coordinates in the map. The result is always coordinates in the original map coordinate space (i.e., it accounts for shrinkage as well as the scaling and translation factors applied in rendering).

The primary design secrets of the coordinate transform module are the data structures used for maintaining a record of transformations, and the algorithmic details of how they are used to provide the needed services.

Tile

A tile is a raster representation of all or part of a map. It has integer coordinates in a left-hand coordinate system (the origin is in the upper left corner), following conventions for 2D graphic systems like Java2D. It can render all or part of a tile into a canvas area managed by the user interface module. Given an integer coordinate pair within the tile, it can provide the corresponding floating point coordinate pair in the map (because it includes a transform object managed by the plane coordinate transform object, and can delegate the translation to that module).

The tile module is essentially a thin veneer on an underlying graphics system (likely Java2D). It is not very abstract because it does not really hide the binding to that graphics system very effectively. However, to the extent possible, code that depends heavily on which graphics system is used should live in the tile module rather than other modules, so that a different graphics binding would mostly require replacing the tile module. Other dependence on the graphics system may not be totally avoidable, but they should be minimized.

User interface

The user interface is responsible for display of maps and providing a way for the user to control map manipulation. The choice of user interface toolkit (e.g., Java Swing) is a secret of the user interface module. Many details of the interaction (e.g., how the user specifies the degree of shrinkage desired, and how the user selects protected regions in the map tile) are also isolated in the user interface module.

Image display  The user interface maintains an on-screen representation of the map (which may be all or part of the tile). It may zoom or or pan the image display. Detailed design of the operation interfaces is left to the developers, who should design it to fit the conventions of other display modules in the selected interactive coordinate transform is "remembered" somehow by the module. The likely implementation of this is that an object is constructed, and the operation for mapping from tile coordinates to map coordinates is a method on that object.
environment. For example, it might implemented as a subclass of the Java canvas class.

**Mark regions** The user interface allows a user to interactively mark rectangular regions of the image canvas. These are the regions that are *not* to be squished (e.g., the user might draw a protective region around the area of an important intersection or group of intersections of streets or roads). The user interface records a list of such regions, translated into map coordinates. See the slice computation module below for how these regions are used.

**Squish** At the user’s request, the image squish control module is responsible for performing a horizontal or vertical image manipulation using the image manipulation modules. This comprises three interactions with other modules: Obtain a list of slices from the slice computation module (below); invoke the map shrinkage operations corresponding to the slice computation; and obtain a new map tile with the shrinkage commands applied.

**Slice Computation**

Computation of image slices depends neither on the image format nor on the on-screen display; it simply provides a geometric calculation to obtain a set of horizontal extents and a set of vertical extents from a list of rectangular regions. (Its design secret is simply the algorithm it applies.)

Compute horizontal slices takes a list of rectangular regions (in any order) represented as \((x_{emin}, y_{emin})\), \((x_{emax}, y_{emax})\), where each \(x_{emin} < x_{emax}\) and each \(y_{emin} < y_{emax}\). It returns a vector of 2\(i\) floating point numbers in ascending order. The first element of the vector indicates the beginning of a region that *must not* shrink, the second is the end of that region, the third is the beginning of the next region that cannot shrink, etc. (Note that if the list of rectangular regions was not empty, there should be at least two \(x\) values in the returned vector.) Compute vertical slices is analogous but computes slices on the \(y\) values.

**Release 1**

The functionality described for release 0 is intended to be minimal but at least somewhat useful. There is a wide range of options for producing a better map schematizer, starting with release 0 as a building block. Students are encouraged to be creative but disciplined, thinking carefully about how the map schematizer will be used, but also about how it can continue to evolve through many more cycles of de-
sign and implementation, including further work by distributed teams with limited opportunity for face-to-face discussion.

Some possible enhancements include:

Add a vector layer

There are potentially two hard parts to this, and students might want to tackle one or the other.

**Import a vector layer from geo-referenced data.** Import from a geographic data format (e.g., ESRI Shapefiles), export to (and possibly import from) drawing tools such as Adobe Illustrator, Inkscape, or Gimp.

The hard part of this is rectifying (lining up) the vector and the raster layer. It means the raster layer needs a “real” projection, not just an arbitrary interpretation of its pixels. There are excellent open source libraries for manipulating geographic data, including application of map projections, but learning to use one of those libraries (and experimenting to make sure it works as one expects) takes some time.

**Draw a layer.** This might be trivially “rectified” by drawing in the same coordinate system as a background raster layer, but it requires a lot more work to add drawing functionality to the user interface, and it would be a lot more useful if the raster and vector layers could be exported to a multi-layer drawing tool as separate layers, rather than composited into one raster layer. This may be a practical undertaking in a few weeks only if one has access to good libraries or components for both the drawing part and for converting two and from the format used by an external tool.

Either approach to adding a vector layer is a significant piece of development; doing them together is probably too much for the second half of the term, unless students know of libraries to take care of most of the details.

More sophisticated shrinkage

One can imagine a variety of alternatives to vertical and horizontal stripes through the whole map. Some of them (e.g., angled stripes) are probably too difficult to provide, but others, such as a patchwork of equal-width or equal-height rectangles with the same net shrinkage as a single stripe, may be quite feasible.

**Vector simplification**

The project description for schematic maps mentions rules for simplifying as well as shrinking a map. Of course this only makes sense if the map schematizer already supports one or more vector layer, so it cannot be the first enhancement attempted.
Undo

A good user interface should allow the user to experiment. The design described above makes no provision for trying some shrinkage, undoing that change, and then trying again with a different way of shrinking the map. In addition to the user interface module, this probably requires a small change to the map module and a more significant change to the coordinate transformations module.