Sam Strasser
May 4, 2008

# Visualize History

Visualize History uses temporal and geographical relationships to present history graphically. The website, visualizehistory.com, displays arbitrary historical data on top of a map, so allow for the correlation of history in space and time. By putting history in a graphic setting, implicit relationships show up to even the most basic user that would have been hard to spot otherwise. The user can navigate through history, choosing which topics and what time period to study at any time, and user interface ideally is able to adapt to both broad and narrow queries. To achieve that goal, the project comprised of three distinct steps: defining historical data in the abstract, researching and implementing the back end and designing and implementing the front end or user interface. Since no applications approach the problem of history in exactly this way, each piece required research and design before implementation, and each presented unique and complex challenges.

## Historical Data in the Abstract

Before implementing anything specific to Visualize History, history has to be formalized. By looking at some specific examples that cover common topics in history, a fairly clear picture of history can be seen. In graph theory terms, history can be considered a graph with vertices and edges. Those vertices, which I call events in this project, can be broken into four subsets that encompass history while distinguishing between different entities sufficiently. All three subsets have a start and an end date, which is what makes them events. Each subset then encompasses different situations found. The first is what most people would consider an historical event, for example, a battle. It has a well-defined geographic location that determines where it sites in both time and space. That might include a region and not a specific point as its geographical piece, as long as such a piece exists. For example, the founding of Hawaii would have a complex set of polygons describing the exact shape of the islands when the state became a state. This category encompasses all events with geographic parts, even entities like a presidency, which may not be considered to be an event by many people. Contrast that with the second subset of the vertices, people. A person does not necessarily have a geographic location, although some are obvious, such as birth place and death place. The vertex, i.e. the person, fits in history, but moves around geographically, and has to be represented that way. Abraham Lincoln would be shown, for example, in Illinois during the Lincoln-Douglas debates, but in Washington D.C. for his assassination. The third and final subset is nodes which need to be part of the graph so that relationships can be shown, but which do not themselves get displayed on the graph. For example, a node called "American Wars" could be displayed on the map, but would provide an interesting topic for study. That last subset is called topics, as the nodes are usually abstract topics that people study but that did not take place per se.

The edges in the graph represent links between those nodes. There are many potential types of edges, although three examples will provide a clear picture of the use for an edge. The first type is parent-to-child. It specifies when a node is a child of another node. So, for example, the Civil War and the War of American Independence would both be children of the node American Wars. That way, those two wars are relatable by the node that they have in common, that is, their parent node. The other types of edges can easily be considered the verbs of history; an army that marches across Asia should be shown as moving over time, and a family that moves to Israel in 1949 should be shown as moving during their 2 week journey. The edges in this case would be of type "geographic movement" and would tell the user interface what happened, letting the UI display that information in a suitable way. Another example of an edge type would be of causation, which would show events that explicitly led to other events.[1]

## Data Backend

[1] One interesting implication of Visualize History is that these explicit connections are not entirely necessary. When scrolling through time and space, the users observe connections that become obvious when overlaid on a map but which might otherwise escape notice. Nonetheless, by describing the connections, the site allows for the maximum flexibility for future displays of history which may not illuminate implicit connections so obviously.

Given the abstract model, the project's next step is to implement the storage for that data. The basics for the model can be implemented on the backend, using the basic functionality of a relational database system. Two related design goals govern the implementation of the backend part of the site. The first goal separates backend from frontend entirely, which allows for different user interfaces to be plugged in and for development on the two pieces to happen simultaneously, with minimal interaction required. The backend outputs its data in JSON format,[2] which can be read by nearly all standard languages and which provides the data only, without any information describing how it should be displayed. As of right now, the backend only implements a subset of the data model described above. The topics and the events subsets of the vertices are implemented, and the parent-to-child edges are implemented as well. The current set allows for an easy to manage set of data while providing enough flexibility to demonstrate the core ideas in Visualize History.

Having established the data schema, data needs to be added to the site. The project explores two different ways of storing data. Ideally, Visualize History would not be responsible for storing the entire history of the world. Many people already study history, writing about it and synthesizing it in interesting and creative ways, and it would be impractical for a website to be in charge of that process. Several sites jump to mind, the most promising one being the Library of Congress site, which hosts vast digital collections, including an API. [3] The API is difficult to use and does not present information in a way that would allow for the kind of historical manipulation required by Visualize History. Sites like Wikipedia also provide an API, but problems both with parsing the human-created HTML and with trusting a community-edited website arise.[4] No single source currently exists that can be trusted and consistently parsed into historical data.

[2] JSON, like XML, provides a standard definition for common programming ideas like arrays, objects and literal values. It provides a less verbose and therefore faster syntax than XML and has become the industry standard for data transfer across websites. See http://www.json.org/ for more.
[3] See: http://www.loc.gov/library/libarch-digital.html and http://www.loc.gov/standards/sru/
[4] See: http://en.wikipedia.org/w/api.php

In place of a single source, the current implementation uses several different sources to demonstrate the capabilities of the site. The four sources each display data in a different way, but each had to be parsed by a custom script, translated into a format understandable by the site's backend, and then uploaded. The cumbersome process serves the goals of visualizehistory.com as a prototype but would not suffice for future work. The four sites cover different topics in different levels of depth, but focus on the United States and especially the period around the Civil War. No piece of the site, on either the frontend or backend, relies on the time or place, but, for purpose of demonstration, that time period and geographical area was chosen.

Several data formats present themselves as potential alternatives to a single source or to manual upload. In the closely related field of GIS, several data formats exist that describe data, and some of those formats include syntactic elements for dates and times. Mostly, however, GIS applications require vastly more precise and accurate data that relates directly to the geography, as opposed to historical data, which is linked to the geography but encompasses more than just geography, as discussed. A more helpful data format, GEDCOM, is used to display information on family trees.[5] Correlating a family's history with global events shows a lot of promise as a point of interest for users of the site, who want to mesh family traditions and stories with history and social studies class. The most promising format, though, aims at representing very similar ideas to the history described here. Called KML, the format originated as a part of Google Earth[6] and has now been adopted as an international standard.[7] The format, which is based off of XML and therefore can be used with many standard parsing libraries, describes "placemarks" that correspond directly to the "events" described above. They include a date and time, a name and a description, and, most importantly, an optional date and time. KML only recently began to support time in their syntax, and so very few publicly available KML files include times.

[5]See: http://www.familysearch.org/GEDCOM/GedXML60.pdf for the specification and description
[6] See: http://earth.google.com
[7] "OGC Approves KML as Open Standard". http://www.opengeospatial.org/pressroom/pressreleases/857

As the KML specification becomes more standard, more files will include the TimeSpan element. Currently, Visualize History supports generic KML files, which can be loaded, parsed, and translated on the fly so that the site's backend does not store any of the information in the external file.

Combining data from multiple places brings several issues to light. All sources use distinct terms for the same events, and no general purpose algorithm can be used to match an event from one site to another site. The correlation of data would benefit greatly if an application could link the names and places on one site to those found on another, but discrepancies and loose definitions make that process difficult. Consider a case where one site references "FDR", another talks about "Franklin Roosevelt" and a third discuss "President Roosevelt." Although all three websites are discussing the same person, it would be difficult for an application to determine that. One interesting thought, alluded to above, stems from this inability to correlate different topics; it may be that implicit relationships are sufficient for the level of study aimed at by the site. That is, as events from different topics appear on the user interface at the same times or around the same regions, maybe that implied relationship will enlighten as much as explicit ones determined by an algorithm. In any event, the issues that combining data causes prevent manual combination from being a practical solution.

Future work on the backend of the site will necessarily revolve around finding sources for data and managing the translation from those sources into the Visualize History model. The internet provides an infinite resource for historical data, and storing its content on the site's database quickly becomes impractical. Any future model will have to account for the vast size of the datasets being considered. Presumably the solution will involve a decentralized approach, which does require local storage but can read external sites and merely transform their data. Visualize History would need a way of establishing trust with those sites, or at least of conveying to the user exactly where the information comes from, so that the user can understand the trustworthiness of the data being presented. Even having solved the problem on the backend, the problem of transmitting the large datasets to the frontend still remains. Some concept of chunking would be required, so that some subset of the data could be passed to the user, granting him the decision whether to continue with the topic or to move on to something else.

## User Interface

Before delving into the choices made in the user interface currently implemented, it will be helpful to consider exactly how the site should look. Several broad goals direct the design of the user interface of the site. The main goal, to display historical topics at the same time, comes above all others. A few secondary and less obvious goals follow that main goal. One goal is for the site to tell the story itself, and although that may seem simple, it differs greatly from the traditional presentation of history. There is no narrator, no text to provide context and organize thoughts, only the implied and stored relationships between events. The user interface should act as a sort of guide that tells the narrative without the explicit textual telling.

For a site aimed at non-technical and technical users alike, a user interface that feels intuitive at first use serves the user base best. It is the job of the user interface to provide a familiar experience to the user. A user expects to be able to drag, zoom and explore and interactive map, so the user interface should include those well-established features. Making many seemingly obvious features act the way a user expects turns out to cause complex problems. Dates provide a clear example of the phenomenon. As a brief aside, nearly all computer languages store and treat dates as milliseconds since January 1, 1969, which allows for fast and easy calculations using dates. By using milliseconds, JavaScript considers a Date to be a specific moment in time, say, for example, 3:45 PM on today's date. According to JavaScript, 3:46PM today is not 3:45PM today, which might seem ideal but does not represent the way humans use dates. If an event happened on July 4, 1776, for example, JavaScript stores that date as midnight on the 4th. When shown on the map, then, the signing of the Declaration of Independence would appear on midnight before the 4th and then disappear, leaving users particularly confused. The issues stems from the human usage, which implies a precision along with the date. Saying the Declaration was signed on July 4th means to a human sometime on the 4th, and so the event should be displayed at all hours of the day. There are some fairly obvious solutions to the Date problem, and it is not meant to be an example of an unsolvable hurdle. It does demonstrate the complexity required when designing the user interface to fit human ideas and not programming languages.

One way to provide an intuitive user interface is to mimic existing user interfaces. Like the zoom on a map, many of the common tasks Visualize History hopes to address have developed common user interfaces in other applications. For the map, applications like Google Maps, Mapquest or Google Earth have established common practices for interacting with maps.[8] Other concepts can be easily applied. Scroll bars have become the de facto standard for scrolling up and down on pages, and that concept is easily transferrable to scrolling backward and forward through history. Even more than the UI pieces, people expect topics to be presented in similar ways. If the particular topic being studied is politics, then a user will likely expect red and blue states to be red and blue, and outlines to reflect the states or counties being examined. The general model should therefore accept customizations based on the topic being considered. More generally, by using metaphors that people recognize, the user interface becomes usable even to those unfamiliar with the site.

With all those goals and principles in mind, the interface for history naturally emerges. The user should be able to search or scroll through historical topics. When she finds an interesting one, she should be able to explore it more, and, if she wants, add it to the map. The events in that topic will then be displayed on the map. She can add as many topics as she wants, and those topics should be distinguishable. When she is ready, she will begin exploring the map, and will be able to scroll forward and backward in time. She will be able to adjust the time period she is looking at, and to "play" the map like a slideshow of pictures, where events pop up and then disappear as time passes by.

[8] See: http://maps.google.com, http://www.mapquest.com/, http://earth.google.com/

The programming environments available on the web constrain this picture severely. The specific evolution of web browsers has created a complex and generally incompatible framework for web development. Each browser implements its own version of HTML, the Document Object Model, Cascading StyleSheets, and, despite many efforts to standardize the browsers, many incompatibilities remain.[9] Recently, the sudden surge in AJAX applications has caused further inconsistencies. AJAX, which stands for Asynchronous JavaScript and Xml, is not a language but a technique which enables a site to make many requests to and from a web server without reloading the surrounding page. That way the minimum amount of data can be used, speeding up the process and also making the web feel more like the desktop environment that users have grown accustomed to. AJAX development relies on a native object implemented by the browser called XMLHttpRequest, but the object is implemented differently if at all on each browser. For all those reasons, UI programming on the web involves complexities in coding and in testing, since all changes must be checked on each browser and each operating system, not to mention with different security and cookie settings. Several programming frameworks have emerged to deal with this, and Visualize History uses two, namely the Yahoo User Interface (YUI) library and the Google Maps API.[10] YUI tries to standardize many of the user interface issues but external dependencies and a cumbersome initial process to set up the framework. Its intention is power entire sites, and so using YUI for small pieces sometimes can be overkill. The Google Maps API deals specifically with their mapping application but also exposes a model for Events in JavaScript.[11] Each API provides value but add complexity and often fail to address the bigger issues of compatibility. Flash and Java applets are the common alternatives to AJAX programming. Flash requires expensive proprietary software for development and Java applets have a bad history of compatibility.

[9] See: http://www.webstandards.org/about/mission/ and http://www.w3.org/ for more on web standards
[10] See: http://developer.yahoo.com/yui/ and http://code.google.com/apis/maps/
[11] JavaScript is an event-driven language. For example, user clicks or key presses trigger the events on a page

Both alternatives require the user to install a plug-in, and a primary goal of Visualize History is to minimize the effort a new user needs to put in before starting.

Website design, even ignoring programming constraints, limits the design decisions for the user interface. Since the application runs entirely on the user's machine, it can be difficult to save information between sessions. To solve this problem, many sites add a user login module, but that would go against the ease of use design goal. As a result, the default settings should be sufficient for as many users as possible as they are unlikely to be able to change them. Also, the asynchronicity provided by the AJAX framework means that the URL a user sees never changes. Users often look at the URL to remind themselves what they are doing, and search engines use the URL to determine the meaning of different pages on the site. Those concerns are secondary to the implementation of the UI framework that people can begin to use.

The current implementation, factoring in all the goals and constraints, shows a working version of the site described thus far. When a user reaches the site, the four manually-uploaded topics are shown. The user can explore the topic by reading a short description or by following the citation url to read more about it. The user will then add a topic or several topics to the map, and each topic will get its own color scheme so that it can be distinguished from the others. Any topic can be temporarily hidden or removed, and then added again. After the first topic is added to the map, the time slider is added. The time slider provides the core functionality, namely the ability to scroll forward and backward through time.

The time slider's current implementation demonstrates the idea of Visualize History, but also displays serious flaws. The slider displays one day at a time by default, and can be stretched to explore more dates. When a user drags it, the events for that date range appear and disappear on the map, showing an animation of history. When sticking to the basics, the slider suffices as a device to navigate history. In particular, there are two pairs of datasets which each have similar date ranges and levels of detail (US Presidents and US States is one pair, and Valley of the Shadow and Major Events of the Civil War is the other). When each pair is examined, the slider presents a clear correlation, for example, showing the birthplaces of US Presidents move westward, following the expansion of the United States. When a detailed topic is mixed with a broad one, though, the slider fails in its task. It is difficult to examine the detailed topic sufficiently since moving the slider one pixel causes the dates viewed to changed by years. Even the most casual user observes the problem right away, as seen by the user feedback submitted in the survey sent out:

> "A small point, but the explanations for dates often overlapped in my browser. I wanted to look at everything happening on a certain date simultaneously, but had to jump back and forth."

> "i think it is tricky to permit a variable length of time to be displayed, and the slider allows the viewer to do that"

The other obvious problem is in locating an event by date, since it requires fine navigation of the slider. That problem also appeared in the user feedback:

> "I was clicking randomly on the slider because it didn't have date guidelines (as in, an '1864' marker). The random clicking detracts from the connection between time and space. "

> "When the slider is expanded so that you're looking at multiple events, there's no way to tell exactly when an event happened. So to figure out the exact date of event you have to make the slider really small, which is annoying because then you have to move it really slowly to find events. So maybe it would help if it said the date in the event description."

It is clear from the user feedback and from basic use of the site that the slider does not fulfill its original intentions.

A redesign of the slider would need to address the current issues addressed above. The two mentioned relate to each in other in that fixing each would require giving the user more control over the range looked at. The solution would seem to be to make all the date ranges editable. By displaying the minimum and maximum dates at the endpoints of the slider, the user would see the large range. Then, when the slider is dragged, the dates of the events being displayed would also be editable, so that to examine the events of a certain date, the user would just type that date in. Adding smaller dates to the tick marks on the slider might also help users focus and understand their queries better. To implement such a slider would require an entirely different approach. Currently, the slider users three instances of a horizontal slider provided by YUI, which allowed for a relatively simple model but which requires digging deep into the YUI source code to make any core changes. To add editable labels, for instance, would require reverse-engineering of the minified[12] JavaScript code, not to mention that it would require manual animation of all parts of the slider. The goal in using YUI was to avoid manual animation, so that would be defeated. Given different programming environments or a more sophisticated API, though, the implementation might become easier.[13] A 10 minute survey brought 2 or 3 core issues to light, and shaped a design that had not come up in the previous months of development.

Three other features, dropped because of the limited time available for the project, should also be mentioned when discussing a potential redesign for the project. The first, alluded to in the original description of history in the abstract, would be to represent edges by cartoons. For example, a march from Atlanta to the sea might come up as an arrow, which would show in a memorable way the same story showed already. The second feature would be to make a node "drillable". One issue mentioned in the feedback was the clutter caused by zooming out on the map and displaying many events all at once. One way to fix that problem would be to show a small number of markers, and, if the user double-clicked an event, to show the children of that node. In that way, the user could zoom in on the detail and not just on the map itself. The other, closely related, would be for the map to auto-zoom based on its current boundaries. For example, if the entire United States was showing, then maybe only markers separated by 50 miles would be shown, but as the user zoomed in, more and more markers would become visible. All three of these features seemed difficult to implement, and got prioritized lower, leaving the implicit connections and manual zooming to tell the story.

[12] See: http://www.crockford.com/javascript/jsmin.html for an explanation of minification.
[13] For example, between the time the slider was originally coded and now, YUI has released a multi-slider that looks similar to the time slider and implements a piece of the functionality hand coded into the slider.

Future work would first address the issues with the slider. Before continuing, I would reevaluate the programming environments, using the year's lessons to determine which would work best. Assuming AJAX did work best, I would fix a few bugs and then redo the slider, probably without the help and constraints of YUI. It is clear through the experience that, now that a working prototype exists, user feedback should determine the major design decisions.