Napkin Sketch Visualizations – Sketch-Based Authoring of Improvisational Visualizations

William O. Chao
wochao@gmail.com

Description of Domain

Can you think of a time when you wished you could quickly sketch a visualization to illustrate a point? Was it while teaching an graphics class? Perhaps it was while collaborating with a group on a project? Or possibly it was to look at something quickly while brainstorming a project, and not interrupting your flow of work in the process? This project is aimed at exploring quick creation of visualizations for improvisational purposes, whether it is quickly demonstrating data live in a presentation, visualizing to move a conversation in a collaborative process, informally toying around with visualization ideas to see how they'd look, or even teaching information visualization principles in a classroom setting.

Description of Task

The task of this project is to enable the creation of on-the-fly visualizations for interfaces geared toward natural sketch-like input such as touch and pen interfaces. The hope is that this will become useful as interfaces shift from traditional keyboard+mouse interactions, to more free-form, widely applicable touch, gesture, and drawing kinds of interactions. To simplify the scope of the project, the initial focus will be on creating area and wedge based visualizations, and then expanding to other types should time permit.

Description of Dataset being Targeted

To simplify things, this project will aim at creating visualizations for tabular, structured data, most likely from CSV or similar file-based sources (as opposed to stored in a database). The data will be of a magnitude that can comfortably be loaded and manipulated in Javascript without causing a slowdown when using linear complexity algorithms to work with the data. This data should contain no unknown points.

Personal Expertise

In order to help me with this project, I will draw on some past learning experiences. I've worked with and informally evaluated various visualization programs such as Inspire, GeoTime, Tableau, and more. This experience will help with brainstorming of ideas for authoring visualizations, as well as provide some background knowledge of what interactions have already been well established. In addition, I have programmed some preliminary gesture-based prototype interfaces over the second half of the summer, and I hope to use this experience to help with programming the gesture parser and vocabulary for this project. I have also programmed visualizations using Prefuse in the past. Because protovis appears to follow many of the paradigms of Prefuse (but at a much higher level), I hope to draw on my experience of programming with Prefuse in order to understand and determine the most important visual encodings and parameter bindings to map to a sketch-based interface.

Proposed Infovis Solution

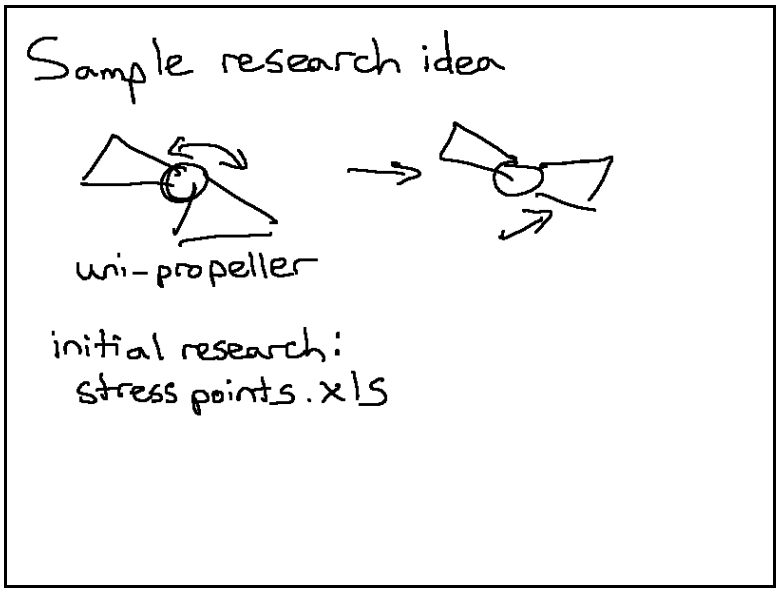
This project will aim to create a sketch-based interface to protovis, an declarative visualization toolkit. The goal will be to try and transfer some of the functionality of protovis to this sketchy interface in order to give the user the feel of a sketch-based visualization toolkit, allowing finer control of visualizations compared to Excel or Tableau, but be significantly faster than coding alone.

Scenario of use

At this stage, the details of interaction will still need to be worked out. However, from preliminary thought experiments it seems like the following may be a good estimation of the general actions the user will go through in performing a task with this system:

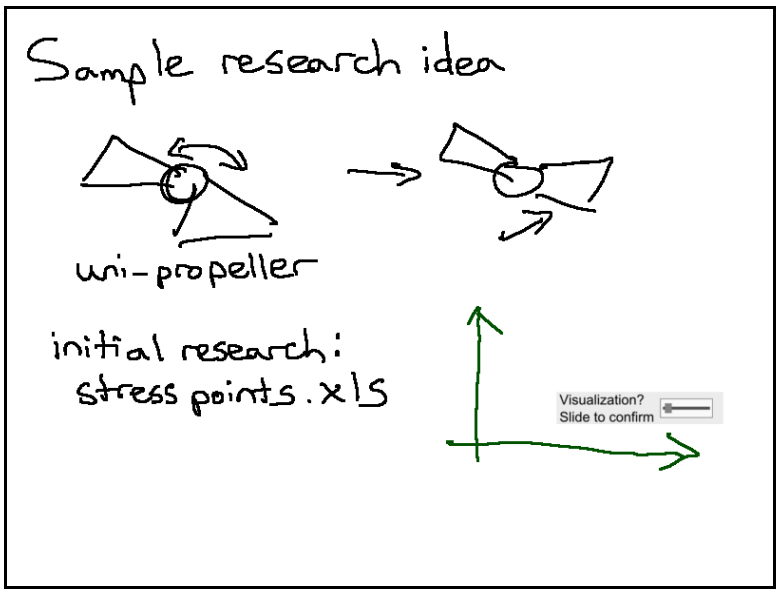
Before the magic happens

The default interface will be a paper-napkin-like free-form canvas where people can write, take notes, or perform doodles. Typical WIMP components will be kept at a minimum. In this example, a student is contemplating building a new propeller design with her group, but she remembers that she wants to look up some important information about the physical stresses involved in this kind of system to make a point to her group. She remembers that her professor provided a spreadsheet containing relevant information.



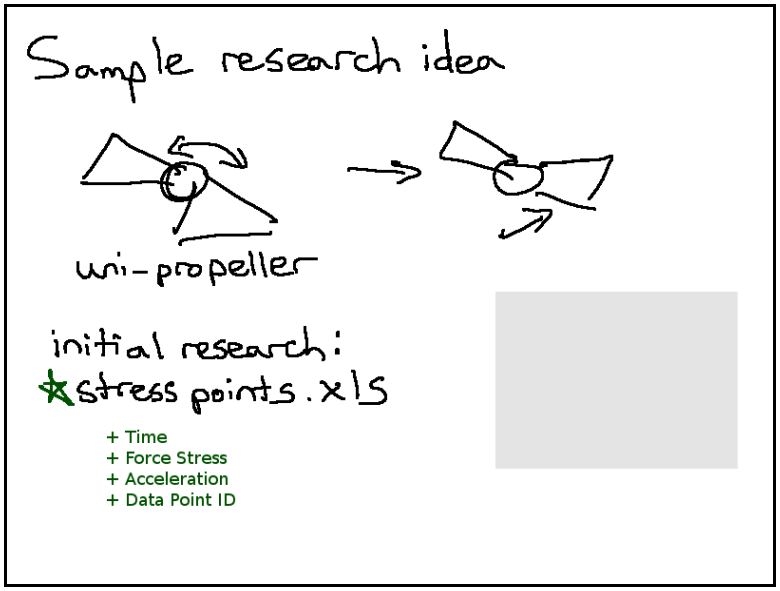
Trigger some sort of visualization authoring mode

The system needs to know that the ink being placed on the canvas is supposed to create a visualization, and not just produce drawings. This can probably be done by drawing an axis or a box indicating that a visualization is desired. In this example, the student quickly draws an axis using 'active' ink. The system responds by identifying the potentially desired active drawing (which will be limited to visualizations in this project), then asks for confirmation.



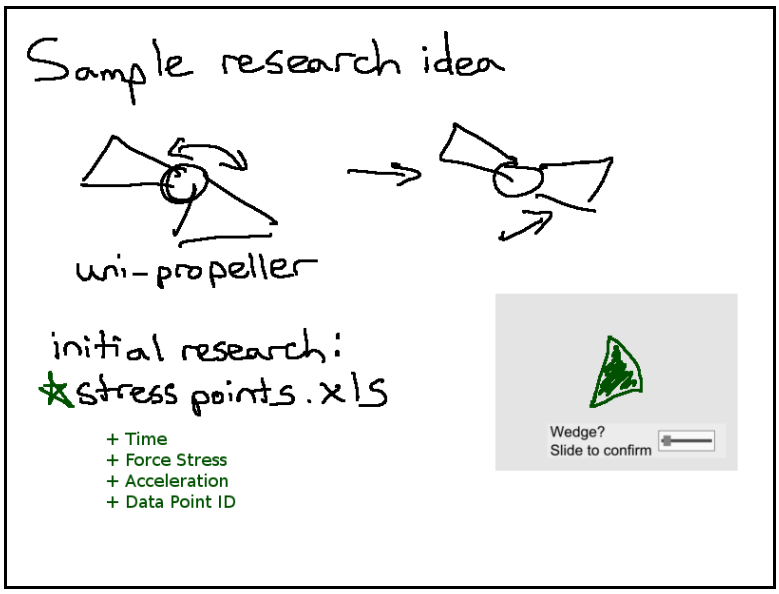
Specify data to use

The visualizations will somehow need to access data. In order of increasing difficulty, this can probably be done by either a file selector, typing in the filename, or writing the filename, then drawing some confirmation mark or doing some confirmation interaction. In this mock-up, the student draws a star beside the spreadsheet her professor provided using active ink, and the system parses the columns of the file and provides active representations that can later be used.



Specify type of mark to use

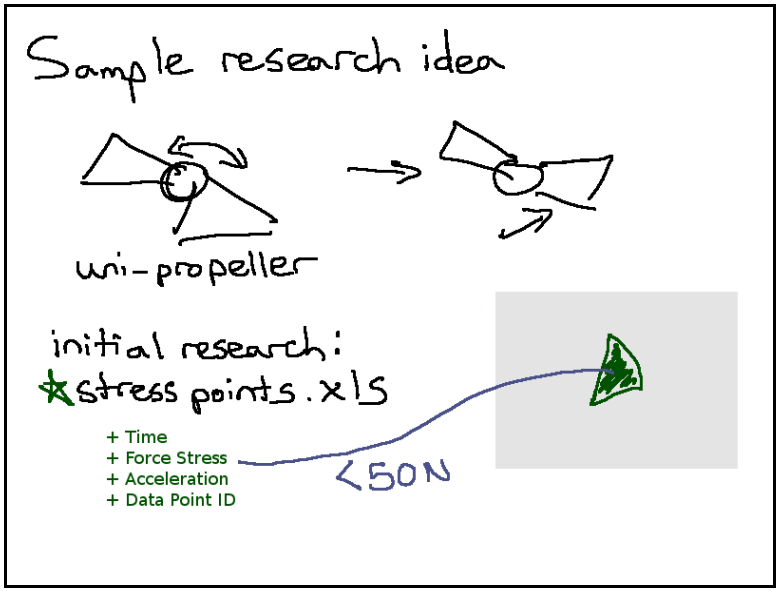
In protovis, visualizations are composed of cleverly formatted and arranged marks. This system will be adopting that design paradigm. In order to reduce the scope of the project, the marks will be limited to wedges or areas. In this picture we see the student draw a wedge to indicate a visualization based on wedges.



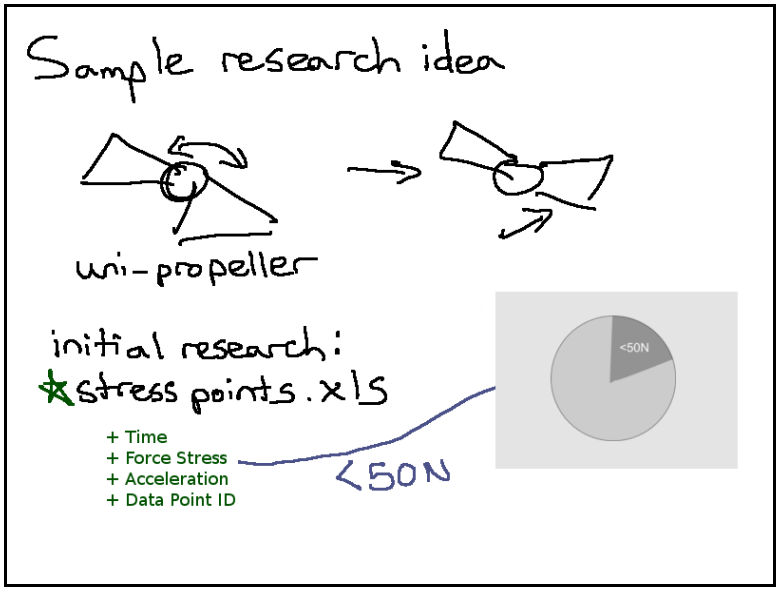
Specify visual encoding:
The data that is imported can be linked to many visual encodings. Protovis provides a general way to edit these encodings. For this project, a limited useful subset will need to be identified, which can still provide functionality powerful drag-and-drop software such as Tableau cannot provide, but can still be easy to code and reducible to 5 or less gestures to remember. Example visual encodings are, but not limited to:

- Position – relative and absolute
- Anchors
- Angle
- Size
- etc.

In this example, the student encodes angle of a wedge to the dimension “force stress” on the propeller joint, and specifies a filter to “less than” a threshold value she is interested in.



Enable visualization re-use
The visualization should then be available as an active widget that other data can be piped into. In this case, if the student should receive other data to look at from another student who has found a different source of data, she can then re-use the visualization to look at the other student's new found data.



Caveat
Please keep in mind that the drawn mock-up is probably still ambitious for the scope of the project, and things may be simplified further where the working area is split into a visualization region and a drawing region, or the entire area may simply be for the visualization authoring while other simpler means will be used to access the data in a file. The best interactions that are feasible within the scope of this project must be taken into consideration and implemented. And finally, data filtering may not even be a viable option without the basics programmed.

Proposed implementation approach
The language this will be programmed in will be Java, as my preliminary thesis work is done in this language, or Javascript, as protovis is implemented in Javascript. Thus due to the nature of the languages being considered, this program will be platform independent. This project will be built on top of the protovis toolkit, either directly (protovis will be added to in order to permit pen interaction), or indirectly (pen interaction from an external program will generate code to be executed with protovis). There are several milestones that I can think of at this point. These will be further refined and added to by the end of the coming weekend.

Milestones
There are several important milestones for this project

- Determine most important set of 5-10 visualization parameters to specify
- Determine if these parameters cover visualizations seen in class and in papers
- Determine robust but easy visual metaphor for how to link data to visual encodings
- Determine easy-to-remember sketch language for encoding
- Learn how to port gesture work in thesis to Javascript, or learn how to link

protovis to Java programs/applets
- Make Javascript console + sketch interface for debugging and fine tuning
- Produce first milestone visualization – reproduce the example area and pie charts, this will use a very basic gesture parser (smoke and mirrors)
- Create a parser for gestures, active ink objects, or both
- Program toy template scripts for different gestures recognized
- Program actual visual language
- Provide rudimentary visual feedback
- Allow visualization to be embedded in a free-form sketchpad (so it's a visualization within a sketch, rather than a visualization alone)
- Satisfy minimum outcomes

The outcomes for this project are as follows

- ***Minimum outcomes***
  - Given imported data, produce a visualization in under 10 seconds
  - Be able to quickly create the following
    - Area graph
    - Stacked area graph
    - Layered area graph
    - Pie chart
    - Doughnut chart
- ***Target outcomes***
  - All minimum outcomes
  - Allow basic interaction with visualizations if protovis allows them
    - Zooming
    - Rotation
    - Panning
    - Selection
  - Be able to quickly create the following in addition to the minimum outcomes
    - Exploded pie chart
    - Nightingale's Rose
    - Burtin's Antibiotics
    - Horizon graph
    - Theme River
- ***Extreme outcomes***
  - Discovered a common gesture/sketch language I can use for visualizations (similar to how iLoveSketch had a common and very simple interaction method to let the user draw a myriad of 3D drawings
    - Preferably all the user will need to remember is 5 things, which can be used for all visualizations
  - Automatic placement of labels and rules, with adjustments using 'ticks'
  - “More like this” or alternatives shown for ambiguous interpretations of drawn commands
  - Can input functions or dynamically create random data with desired characteristics to test visualizations
  - “Infinite canvas” layout, with appropriate slide transitions

Previous Work

There are several works that this project will be using for inspiration. On the animated presentations front, [1] looks at how to create active animated objects controlled by a small number of parameters (often just one) mapped to sliders. This paper shows a scripted, parameter based language for creating visualizations, and maps this to physical interactions appropriate for presentations. In addition, good animation guidelines for presentations are discussed in this paper. There are several other important papers which will be included in the final write up, however to save on time and because these have had less of an impact to the creation of this proposal, I will instead skip to the most relevant tools that have an impact on the design of this project, and will include these references in the final write up.

There are several existing tools which are aimed at simplifying the production of visualizations. Excel and similar spreadsheet programs take a wizard-based approach to automatically generate visualizations based on predefined templates. Tableau [4] takes this a step further and allows for quick visual encodings of data dimensions to visualizations by drag and drop to predefined areas on the screen. At a more general pen-based active animations level, programs designed for kids such as eToys [2] or Phun [3] allow the creation of robots and simulations, which can often act as interesting visualizations if constructed cleverly. The former acts as a visual programming language while the latter is a pen-enabled physics simulation, both allowing for triggering of events, enabling user directed timing if used in a presentation setting. At an even more general level, several toolkits exist to allow for the prototyping of visualizations. Protovis [5] provides a declarative Javascript toolkit which allows for a very rich set of visualizations to be specified and programmed in minutes. Prefuse [6] and Flare [7] are equivalent toolkits which allows users to specify data bindings, visual encodings, rendering, and control of visualizations to enable production of a very wide variety of visualizations. Unfortunately, the learning curve of Prefuse and Flare is very steep. Other toolkits with predefined visualizations also exist, however some argue about the ease of extending these visualizations [8, 9, 10]. Another flavour of toolkit is also available to the designer. Processing [11] is another toolkit which lets designers focus on programming visual marks without worrying about any complicated underlying programming structures (such as initializing many abstract classes). Unlike protovis, this doesn't provide shortcut marks with well defined anchors which can be bound to data or other marks, which leaves the logic of certain visualizations such as stacked area graphs too difficult for the everyday user. From this, tools become more general and more difficult to effectively design visualizations, such as ActionScript, Flash, OpenGL, etc.

This project hopes to create something with a flexibility in between Tableau and Protovis.

References

[1] Zonker, D.E., and Salesin, D.H. (2003) On Creating Animated Presentations. Eurographics/SIGGRAPH 2003

[2] eToys, www.squeakland.org (accessed on October 28, 2009)

[3] phun, www.phunland.com (accessed on October 28, 2009)

[4] Tableau, www.tableausoftware.com (accessed on October 28, 2009)

[5] Protovis, vis.stanford.edu/protovis/ (accessed on October 28, 2009)

[6] Prefuse, www.prefuse.org (accessed on October 28, 2009)

[7] Flare, flare.prefuse.org (accessed on October 28, 2009)

[8] Infovis toolkit, ivtk.sourceforge.net (accessed on October 28, 2009)

[9] Visualization toolkit, www.vtk.org (accessed on October 28, 2009)

[10] JavaScript Infovis Toolkit, thejit.org (accessed on October 28, 2009)

[11] Processing, processing.org (accessed on October 28, 2009)