

Curly-Squeegee Process Book

Brian Kimmig & Jimmy Moore

December 3, 2015

Abstract

Curly-Squeegee is a web-based tool for visualizing and exploring actor filmographies in an interactive way. Using different views users can select and filter for a variety of film traits including: rating, box office earnings, genre, and release date. We hope it enables users to explore their favorite actors and find interesting or surprising trends.

Contents

1	Editing notes	4
2	Project Background	5
2.1	Motivation	5
2.2	Intended Audience	5
3	Data Collection	6
3.1	Data Processing	6
3.2	Data Formatting	7
3.2.1	Only Visualizing Films	7
3.2.2	Formatting numbers	7
4	Project Interface	8
4.1	Landing Page	8
4.1.1	Site Redesign	9
4.2	Loading Page	9
4.3	Results Page	10
5	Visualizations	12
5.1	Actor Fact Box – Deprecated	12
5.2	Filmography Timeline	13
5.3	Parallel Axis Coordinate Visualization	15
5.4	Treemap Visualization of co-workers	16
5.5	Genre Visualization	17
6	Connected Views	18
7	Project Feedback	19
8	Team Evaluation	20
9	Future Work	21

To ourselves . . . for always being there.

1 Editing notes

Depending on how thorough we want to be (and how much time we have for thoroughness) we could incorporate any of the talking points from the “example” documents

- Real time audio visualization project - Soundscapes
- Oh Shit, earthquakes!

Also:

1. Explicit analysis on color choices, modes of comparison (length, hue, etc)
2. in-depth explanation of our choice of design principles or encodings \visual variables
3. Compile a bunch of good funny gifs we could use as loading images. pick from them with a randomizer function.

2 Project Background

2.1 Motivation

The idea for this project came from the fact that both developers watch a good amount of movies and enjoyed sites like IMDB and RottenTomatoes, but wanted something that focused on specific actors rather than being movie-centric. CS is their solution to needing to sift through lists and text to appreciate a given actors filmography. In three views, one can see their entire body of work as a timeline, with length and color encodings for film-output and film-quality, respectively, as well as career visualizations showing the breakdown of movie genre over the course of their career and a multi-axis interactive plot to explore an actors output as a function of date ranges, ratings, box office earnings, and directors.

CS is meant to provide a new way of viewing actor data, and seeks to facilitate a fun and interactive web-based solution to questions like:

- How many movies has an actor acted in?
- Has an actor been type-cast to a specific genre?
- What is the best movie they have made? The worst?
- Do they consistently star in well-reviewed films?
- Has their career had a golden period in which they were particularly busy, or appeared in well-reviewed films?
- Do they have any co-stars with which they regularly appear?

2.2 Intended Audience

Curly Squeegee is geared towards the general “internet public”, and offers something of interest to the casual movie-goer and film buff, alike.

The interface is clean and self-explanatory. Users are presented with a short text prompt describing the purpose of the site, a text query box, and a list of trending actors. This presentation allows them to start using the tool immediately with little explanation.

We hope that the site design and visualization aesthetics make it a natural and easy tool to use.

3 Data Collection

Our original design goals always envisioned this as an interactive web-based tool. We did not want to rely on a static data, and instead wanted to tap into existing cinema datasets to generate filmography visualizations. We wanted our tool to be applicable to any actor, from any time for the most flexible and fun experience for the user.

In order to build such an actor database we explored several film-related API's. After some searching, we discovered the majority were poorly populated, did not have the data we wanted, or charged for use. Luckily, we were able to identify two options which allowed us to proceed: My API Films and Open Movie Database.

Using Node.js and Meteor\MongoDV, we implement s a RESTful architecture to gather API calls via GET requests. These returns are formatted in JSON and stored in a MongoDB database. This dataset is fairly dynamic since we rely on user queries to pull the necessary information from the APIs.

Do we want to have a breakdown or description of the API call, data retrieval, and database storage?

3.1 Data Processing

Minimal data processing was required beyond defining and populating our data structures. The returned API requests are returned in JSON format, so there is little clean-up beyond string formatting (for names and numerals), and logic for handling missing data.

The raw API calls were fairly detailed, so we selectively culled unnecessary fields and aggregated filmography data based on what we want to visualize. These fields included:

· Movie Title	· Release Date	· Movie Poster
· Director	· Genre	· Co-Stars
· IMDb Rating	· Rotten Tomato Rating	· Box Office Earnings

In order to interact with this data, we stored our actor query results in two separate data structures

- **Actor Table:** This stores all the actor information of a selected actor, including the movies they've acted in.

- **Movie Table:** This contains all of the information for each movie we wish to plot or visualize.

The data collection and filtering was likely the most sophisticated portion of this project. However, once complete, everything was easily accessible for views.

The nature and scope of our visualizations did not require much computation or analysis. We used built-in javascript math functions and aggregate parameter counts of our actor data to quantify and display the values we wanted to show.

3.2 Data Formatting

talk about filter and utils functions?

3.2.1 Only Visualizing Films

- Mention Tom waits case where his most popular work was soundtracks
- Talk about formatting for films which are currently released (and therefore have data/votes).

3.2.2 Formatting numbers

- Remove NA from votes
- removing commas from votes
- formatting years

4 Project Interface

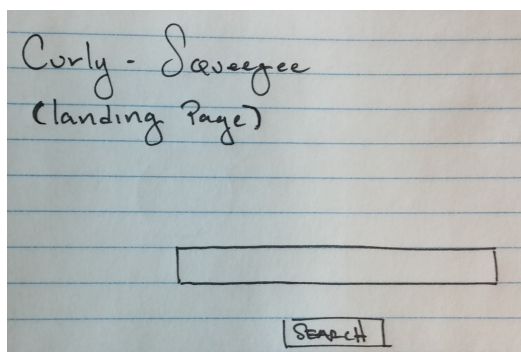
Due to our accelerated development schedule, we were able to move from sketch book to implementation very quickly. We brainstormed multiple views which could help answer the questions raised in section 2.1 for a variety of actors. We also sought to encourage user interactivity and re-use.

In the interest of time, we decided on four views: two main visualizations (Actor Filmography and a Parallel Axis Chart) and two more easily implemented graphics – an aggregate genre view, and tree map of common co-stars.

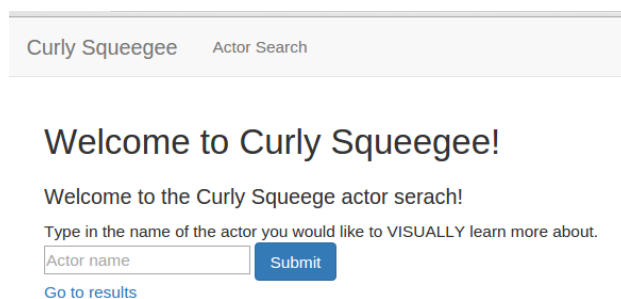
The following subsections discuss the implementation and styling each interface, and how or why it evolved during the life of the project.

4.1 Landing Page

Taking a cue from Google’s clean and austere design, we wanted a plain front page with minimal clutter.



(a) Original design sketch



(b) First implementation

Figure 1: Landing page development

We added some explanatory text, realizing a new user would have no idea what they were presented with, and added some navigation bars for ease of moving around the pages. In this first iteration, the user has the choice to enter any actor.

One feature of our use of API calls is that this approach is fairly robust against using alternate names. Entries such as stage names, nick-names,

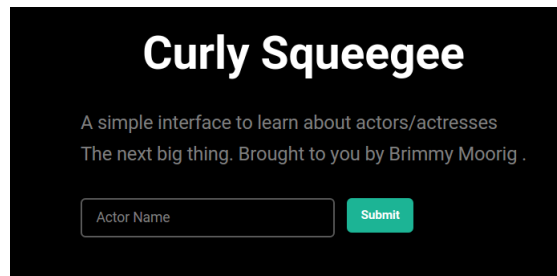


Figure 2: Final version of landing page.

and misspellings will return the most-similar actor, which is often times the desired result.

One drawback is that there is currently no disambiguation to select between multiple actors with the same name. The only choice is to search for a more exact version of the actor’s name in the database.

4.1.1 Site Redesign

Once the views were working and the settings more or less finalized, we decided to work on changing the look of the site. We wanted to keep the same simplistic look, but benefit from some additional polishing from some pre-packaged CSS and JS styling elements.

Not every page was affected, but we did alter the landing page (Figure ??) , and modify the scrolling behavior of the results page. These changes were purely cosmetic and were employed to give the site a more cohesive and smooth look.

We chose a design template from HTML5 Up for its stripped-down appearance, and then JS package (fullPage.js) for jump-to-view functionality. Since all of our content lives on one page, we wanted a clean way to transition between views.

4.2 Loading Page

Once the user has made their selection, the node.js framework takes that search query and polls the MyAPIFilms’ and OMDb’s databases to collect the necessary actor information. Fulfilling these requests can take some time, so we display a loading gif to indicate that the process is ongoing.

The wait times for each search request depend on the extent of the actor’s

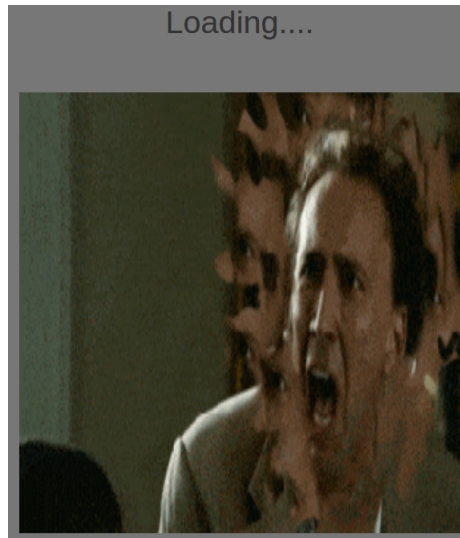


Figure 3: After the user submits a search query, the application waits for the API calls to complete and fetch the actor’s filmography. An animated gif can help pass the time.

film catalog as well as the user’s internet connection. In our tests, we have experienced wait times as small as a few seconds to upwards of 5 minutes. Typical queiries return within 1-2 minutes.

4.3 Results Page

Once the API calls are complete and the data is saved in our database, we load a “results page”, showcasing our multiple views and actor information.

The style of the page is intentionally minimalistic, including a top navigation bar, a small inset photo of the selected actor in the top right, and a Navigation hotlink scroll box on the middle right.

Anatomy of Figure ??:

- **Navigation Bar:** The top of the page displays links back to the search page, the actor name, and their photo.
- **Main Views:** Each visualization is centered and displayed in its own screen space/ The user can scroll down to view each one, or use the view selector on the right side of the screen.

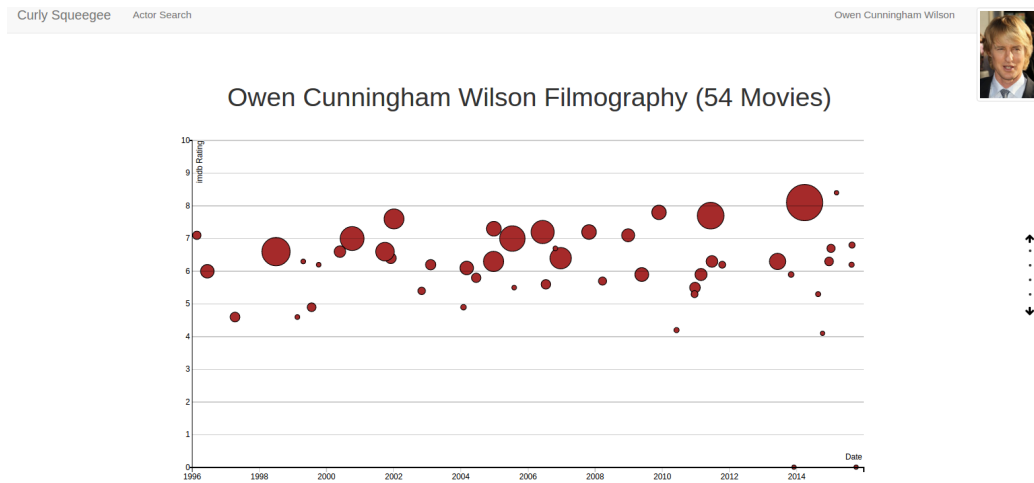


Figure 4: Results page layout

- **View Selector:** On the right hand side of the screen, the user can click the stacked dots to jump to any of visualizations. They may also scroll, and when in the vicinity of the visualization, the browser will auto-center the visualization after a short dwell time.

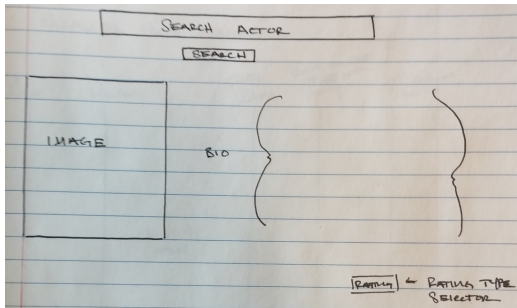
5 Visualizations

In this section we go deeper into the visualization choices and design details for each of our chosen views.

5.1 Actor Fact Box – Deprecated

Note: This section was removed from the final release. We decided it did not contribute to the overall presentation and was at odds with the interface style we were hoping to create. The text remains for historical analysis.

(Deprecated) This data serves as an orientation tool, indicating to the user whether the correct actor was returned. This is particularly important in cases when searching for actors with common names. Using the OMDb API data, we are able to display a photograph of the actor, their full name, date of birth, and other derived data from the API call. We currently display their total number of films. other useful information would be when they started acting \first credited role, years active, and highest or worst rated movie.



(a) Original design sketch



(b) Original implementation

Figure 5: Actor ‘Fact Box’ development

5.2 Filmography Timeline

The most natural way to visualize an actor’s career is over a timeline. We considered using circles on a timeline as a suitable encoding for showing film distribution Early in our design phase. This decision was inspired by the philosophers, poets, and musicians visualization shown in class.

In our original design, we wanted to chronologically visualize the distribution (and density) of an artist’s career and encode information such as film ratings or movie earnings as the circle radius or fill color. We refined this decision after our brainstorming session (Section 7), where concerns were raised that actors with frequent and/or successful work would create a cluttered timeline visual. We also recalled the ineffectiveness of using areas to encode quantitative values for comparison. When it came time to code our visualization, we decided to switch to a barchart visual. This way we could cleanly show movie release dates along an x axis and use clearer comparison of quantitative data by mapping it to bar height.

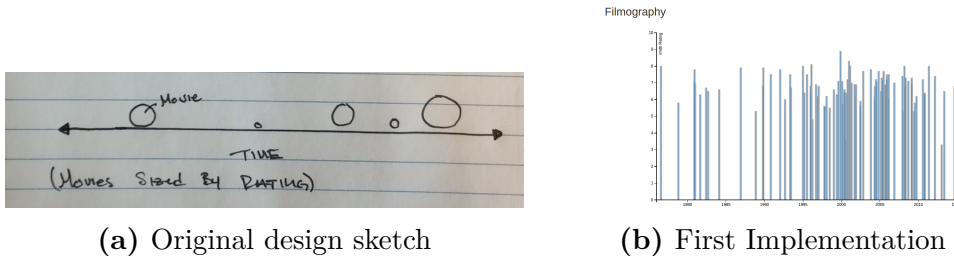


Figure 6: Timeline development

Despite this revision, we encountered several issues relating to bar sizing, spacing, and overlap. When designing this visualization, we did not initially consider cases of long acting careers, high film output, or areas of dense activity (Figure 6b). In each of these regimes, bar sizing and spacing becomes an issue for readability and visual appeal. Particularly frustrating, were positions where an actor would have multiple films released within a small time frame. We tried distinguishing these views by using thinner rectangles, adding stroke width to the bars, or introducing opacity. In the end, we were not happy with any of these solutions and sought a better visualization method.

After realizing we were most interested in showing two values – release date and rating, we saw this could be arranged as an ordered pair, and

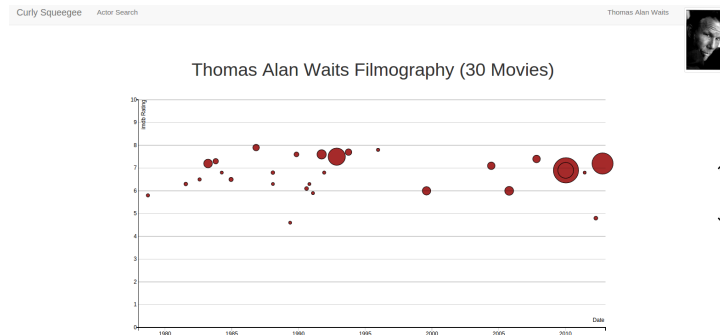


Figure 7: XY Plot of Tom Waits filmography. circles are sized by number of IMDB votes.

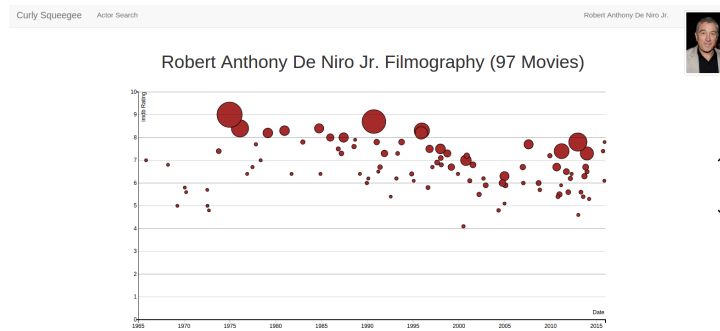


Figure 8: Slump in the 90's.

thus most effectively plotted on a cartesian grid. Figure 5.2 is a marked improvement over the earlier efforts to illustrate an actor's filmography. in addition to spatial placement, we encoded the the popularity of a film as the size of its dot. Here, popularity is determined by the number of votes it received on it's IMDB page.

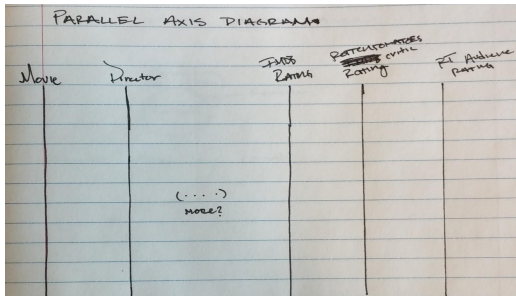
The XY plot naturally lends itself to an ordered ranking in each dimension, which can be quickly exploited by users to identify such information as : high and low ranked films, popular films, and any trend in the actor's career over time. For example, figure ?? verifies the slight decline of Robert DeNiro's film quality.

5.3 Parallel Axis Coordinate Visualization

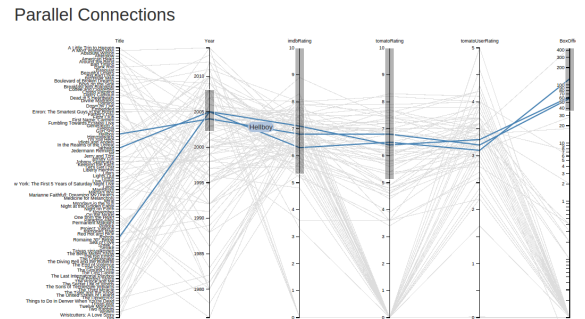
From the start, the parallel axis coordinate view was what we were most interested in implementing for this dataset. With this view, we can brush over multiple axes to highlight movies fitting arbitrary criteria the user can impose. The films which satisfy these filters are highlighted in blue in the parallel axis chart and also in a contrasting color above in the Filmography timeline.

The code for this visualization was borrowed from the Mike Bostock Block, and we modified it to include the following:

- Highlight-on-hover for each line
- Movie title tool tip on hover for each line.
- Increase movie title font size on mouse-over for films in the ‘Title’ axis.
- Each axes is scrubable and will highlight any films in the given selection, and those in the filmography.



(a) Original design sketch



(b) Current implementation

Figure 9: Parallel Axis Coordinate visualization development

5.4 Treemap Visualization of co-workers

The treemap visualization was a last-minute addition to our project, inspired by the “Visualizing Maps and Trees” lecture.

In this view we catalog the other costars the queried actor stars alongside and sizes each box according to the number of shared films.

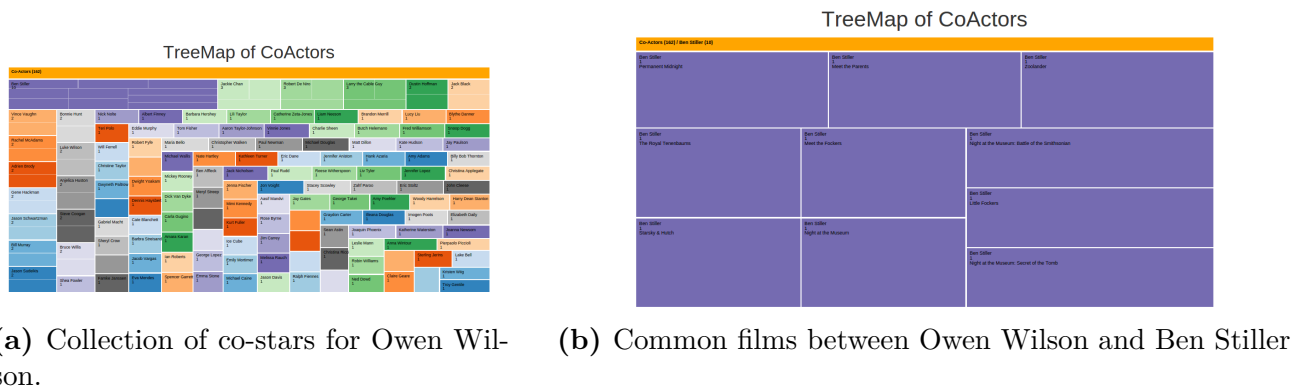


Figure 10: Tree Map visualization for Owen Wilson's career

The tree map supports one zoomable level, to drill down and see a list of the movies the two actors share.

5.5 Genre Visualization

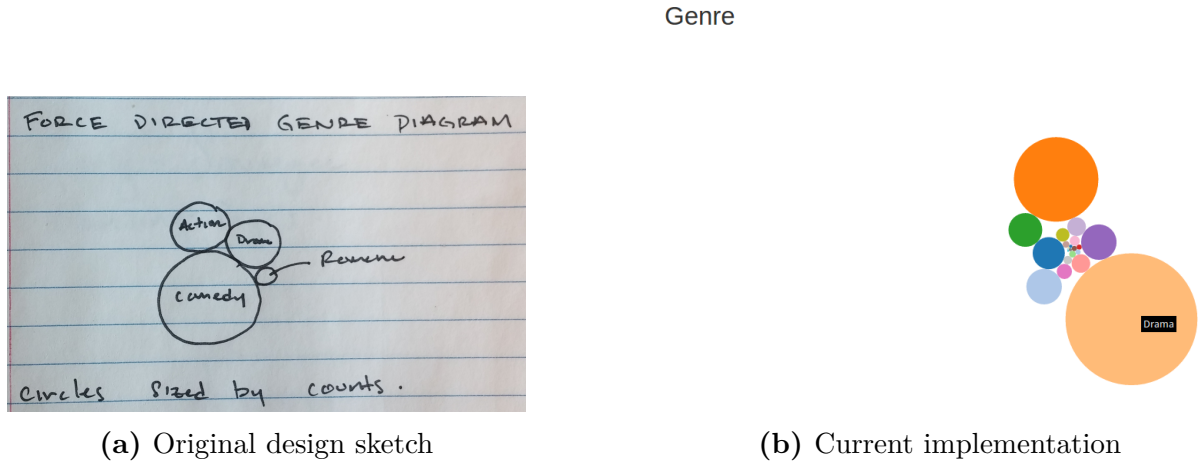


Figure 11: Genre visualization development

The GenreVis view (figure 11) was created to answer whether an actor gets has been typecast, or to visualize the proportion of film genres they appear in.

Each circle represents a unique film genre (Drama, family, comedy, etc.) which is viewable via mouse-hover text. To create this view, we parsed every film genre label in the actors filmography and added unique labels to an array. We counted the number of times each label appeared, and sized the circles accordingly. Films which had multiple or compound genres ('action thriller' or 'romantic comedy') would count each descriptor towards the actors genre aggregate.

TODO: link views.

6 Connected Views

To make this a fully interconnected visualization, we chose to connect views across the four visualizations. Selecting films in the Filmography or Parallel axes views will highlight the corresponding entry in the other view. Brushing in parallel axis view will highlight multiple films in the timeline.

This is especially useful when applying multiple filters in the parallel axis coordinate view and seeing the highlighted distribution of films in the timeline graphic.

Films brushed from the parallel axis plot. We thought this functionality would help uncover any time trends with given film earnings, directors, etc. Of course, any trends will be dependent on the selected actors.

7 Project Feedback

We had the opportunity to present a Curly-Squeegee to another project team after we settled on our project design. We pitched our Idea to Phil Cutler, Ariel Herbert-Voss, and Ian Sohl of the "Legion Profiling Visualization" team. They provided the following feedback:

- **Phil Cutler** (u0764757@utah.edu)

Provided good feedback and constructive criticism. Raised concerns about Parallel axis plot readability in the limit of a long, active acting career as well as a lack of information on new actors. Liked the idea of the filmography visualization, but recommended using bar charts as opposed to circles anchored to a timeline. He thought it was a neat idea, but did not see it's utility. He admitted he does not like watching movies.

- **Ariel Herbert-Voss** (u0591949@utah.edu)

Overall very positive and excited reaction. she loved the parallel axis plot idea and also agreed with Phil that a bar chart for the filmography timeline would be more effective. She suggested scaling bars either by film rating or number of films in a given period (for a drill-down style barchart), as well as shading a given bar to convey additional information. Ariel is a film buff and saw a great deal of utility in this visualization

- **Ian Sohl** (u0445696@utah.edu) No additional feedback beyond what Phil and Ariel had to suggest.

8 Team Evaluation

Brian Kimmig: Brian was responsible for the managing the preliminary API calls, data collection, and database wrangling. His experience as a web developer was very helpful for making this portion of the project proceed smoothly. Brian also created the genreVis, treep map, and parallel coordinate view.

Jimmy Moore: Jimmy contributed code for the Database storage of API calls, the actor filmography visualization, and with site formatting. He also was the project scribe and responsible for proposal and final report documentation.

9 Future Work

We both are excited at the direction this project could take with some more time. There are several areas which work well enough, but which we could see being more user-friendly or thouroughly fleshed out:

1. Timeline View:
 - extend search to other roles (Director, etc.) and add a selector to change search query from visualizing filmography to other directorial roles, soundtrack, or other creative capacity.
 - Allow support for searching and overlaying multiple actors. color the points and overlay them to compare the activity and success of multiple people.
2. Parallel Axis View:
3. : Treemap view
 - General cleanup and exploring other shapes
4. genre view
 -
5. Additional views
 - Create a set view to show which movies any n actors have acted in. Or show a network visualization of the separation of a number of actors. Use the individual movies as the graph nodes. Force directed visualization might be best.