Curly-Squeegee Process Book

Brian Kimmig & Jimmy Moore

December 4, 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 intersting or surprising trends.

Contents

To ourselves . . . for always being there.

1 Project Background

1.1 Motivation

The idea for this project came from the fact that we both watch a good amount of movies and enjoy sites like IMDB and RottenTomatoes, but preferred something that focused on actors rather than being movie-centric. Curly-Squeegee is our solution to sifting through lists and text to appreciate a given actors filmography. In four views, users can see a queiried actor's entire body of work on a timeline, with area and position encodings for release date and film-quality, and popularity. Additional career visualizations include a genre view, which aggregates the total number of movies which occur in individual film genreas over the course of their career and a parallel-axis plot to explore an actors output as a function of date ranges, ratings, and box office earnings.

Curly-Squeegee is meant to provide a new way of viewing actor data, and seeks to facilitiate 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?
- How many cos-stars collaborate with them on multiple movies?

1.2 Intended Audience

Curly Squeege 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.

2 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.

We used Node.js and the Meteor web-framework to set up our website. We use the RESTful framework to make requests to the APIs that have the data (OMDB, My API films). These requests are then stored in a MongoDB database. We created two collections, the first being 'Actors' and the second, 'Movies'. The actors are the related to the movies by the imdbID. Each actor has a list of imdbIDs associated with him/her. We use the IDs to gather their movies. By checking for the imdbID in our DB before making a request to the API we can avoid storing multiples of the same movie.

2.1 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 uncessary 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 agregate parameter counts of our actor data to quantify and display the values we wanted to show.

2.2 Data Formatting

The data came in JSON format but there were not values for every field we wanted to use. For this we implemented the function 'filterData' in utils.js to remove and format the data. For instance, we needed to convert the box office data to a numerical value. We therefore had to parse a string that looked like the following.

```
$1.1M or $1.1k
```

We also chose to set fields that were not available to 0, making each data set have the same type, and allowing us to plot them all using the same scale. Below are the major types of data formatting we performed.

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

Movies that were not yet released were removed from the visualizations because they have no data, besides title/cast, to display.

2.2.1 Only Visualizing Films

There is a plethora of data returned to us via the API, for this project we chose to ignore items like television shows, soundtracks and directing. These features could one day be added to Curly Squeegee but for now we found that it was a major task. Therefore we limited the scope of the project to movies.

3 Project Interface

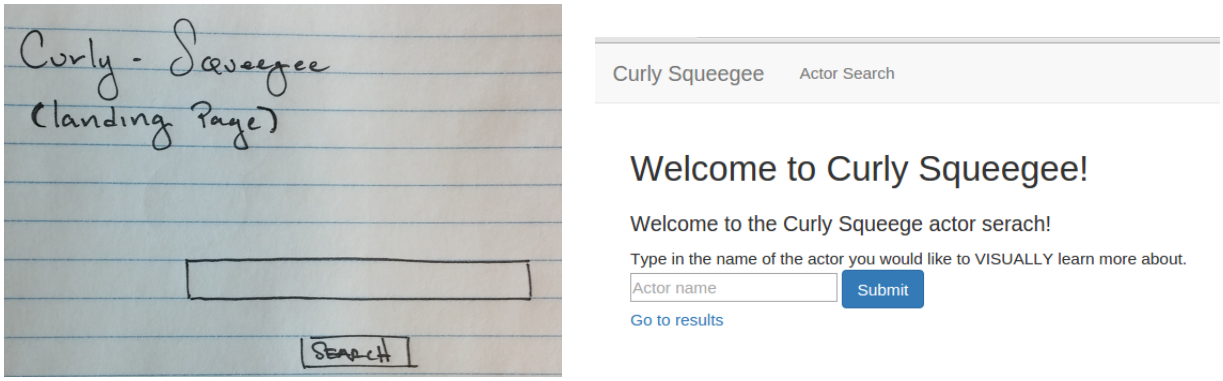
In building out the project interface, we were able to move from sketch book to implementation very quickly due to our accelerated development schedule. We brainstormed multiple views to answer the questions raised in section 1.1 and sought to encourage user interactivity and re-use by making these views applicable to any actor.

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 aggregte genre view, and tree map of common co-stars.

We discuss the implementation and styling of each interface in the following sections.

3.1 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 to inform users of what they are presented with, and include a navigation bar for ease of moving around the pages. In this first iteration, the user must input the actor they want to search for. In the redesign (figure 2) we added an option to select from a list of popular actors.

One serendipitous benefit of using our API calls is that this approach is fairly robust against alternate names and typographical errors. Searching for actors based on their stage names, nick-names, (or thier misspellings) will return the most-similar actor. 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.

3.1.1 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

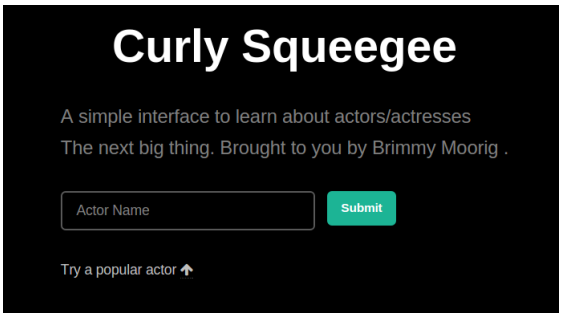


Figure 2: Final version of landing page.

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 2) , 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.

3.2 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. Fulilling these requests can take some time, so we display a loading gif to indicate that the process is ongoing.

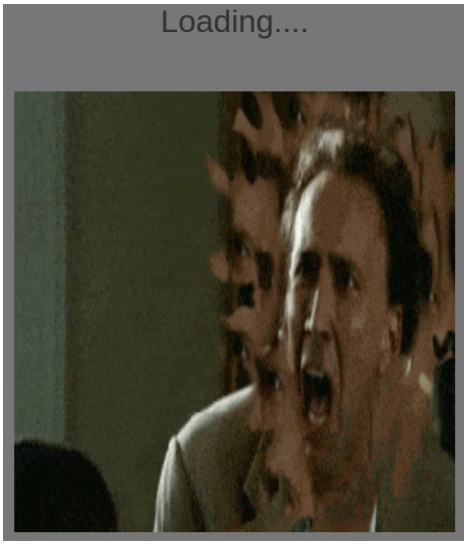


Figure 3: After the user submits a search query, the application waits for the API calls to complete and fetch the actor's filmography. An animated gif can help pass the time.

The wait times for each search request depend on the extent of the actor's film catalog as well as the user's internet connection. In our tests, we have experienced wait times as

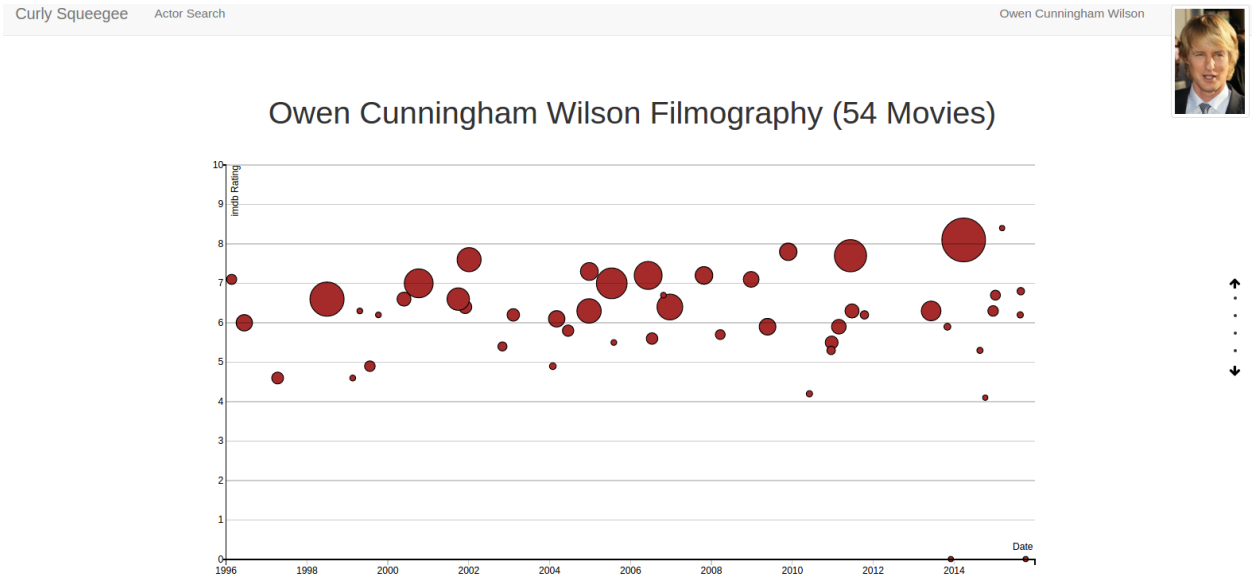


Figure 4: Results page layout

small as a few seconds to upwards of 5 minutes. Typical queiries return within 1-2 minutes.

3.3 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 4:

- **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.
- **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.

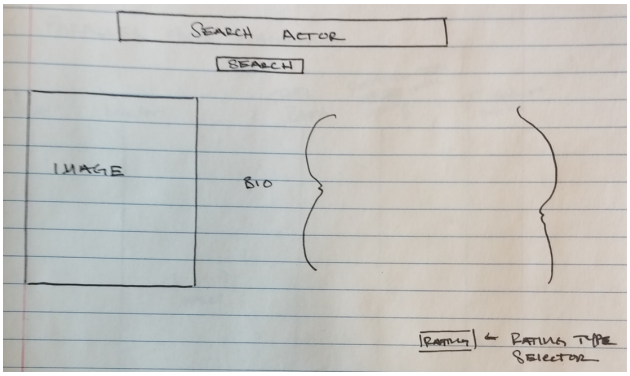
4 Visualizations

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

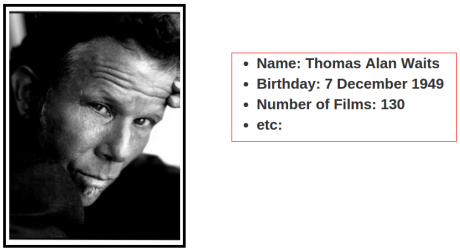
4.1 Actor Fact Box – Depricated

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.

(Depreicated) 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



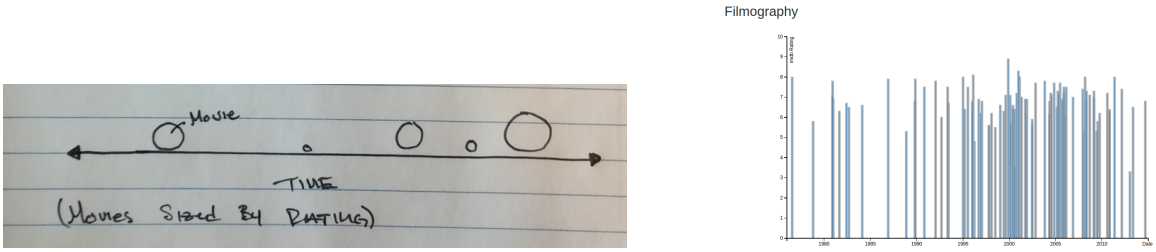
(b) Original implementation

Figure 5: Actor 'Fact Box' development

4.2 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 chroniologically 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 6), 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.



(a) Original design sketch **(b)** First Implementation

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 thus most effectively plotted on a cartesian grid. Figure 4.2 is a marked improvement over the earlier efforts to illustrate an actor's filmography. In addition to spatial placement, we encoded the film popularity by its dot size. Here, popularity is determined by the number of votes it received on its IMDB page.

We feel this view is especially effective as XY plots naturally lend themselves to ordered rankings in each dimension. This is quickly used to identify information such as : high and low ranked films, popular films, and any trend in the actor's career over time. For example, figure 8 verifies the steady decline of Robert DeNiro's film quality throughout the 90's.

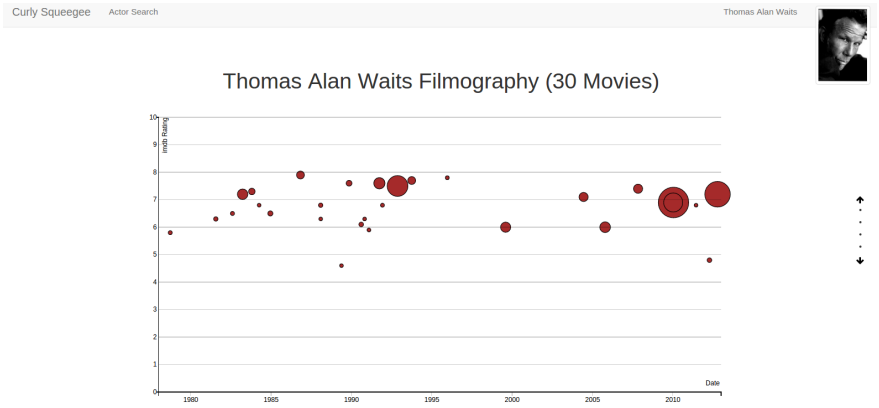


Figure 7: XY Plot of Tom Waits filmography. circles are sized by number of IMDB votes.

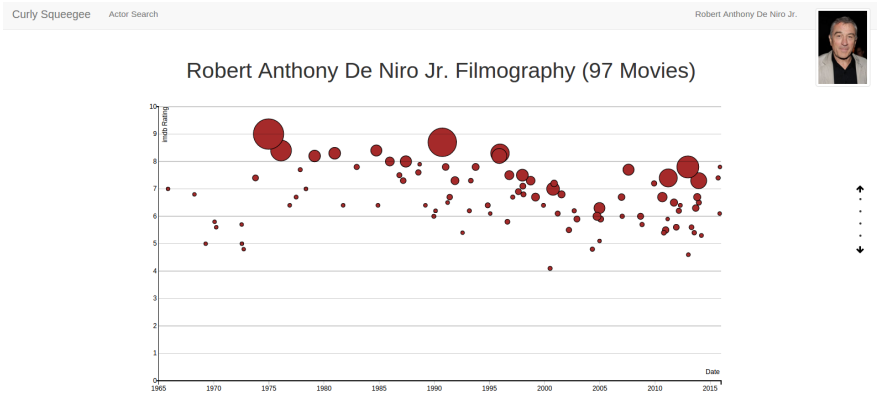


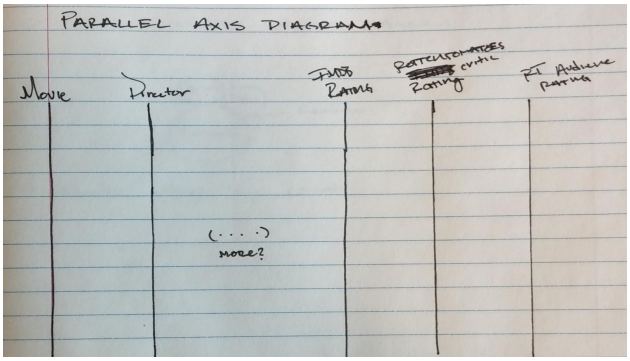
Figure 8: Slump in the 90's.

4.3 Parallel Axis Coordinate Visualization

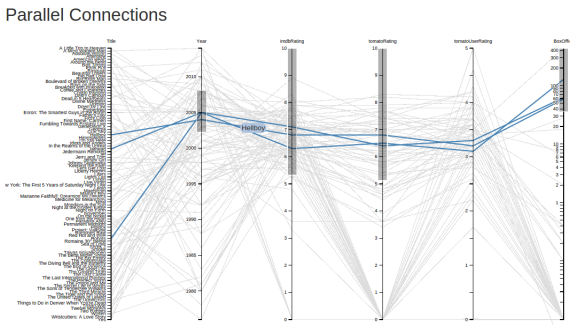
From the start, the parallel axis coordinate view was what we were most interested in implementing for this dataset. With this view, we can brush over multiple axes to highlight movies fitting arbitrary critea the user can impose. The films which satisfy these filters are highlighted in blue in the parallel 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

4.4 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.

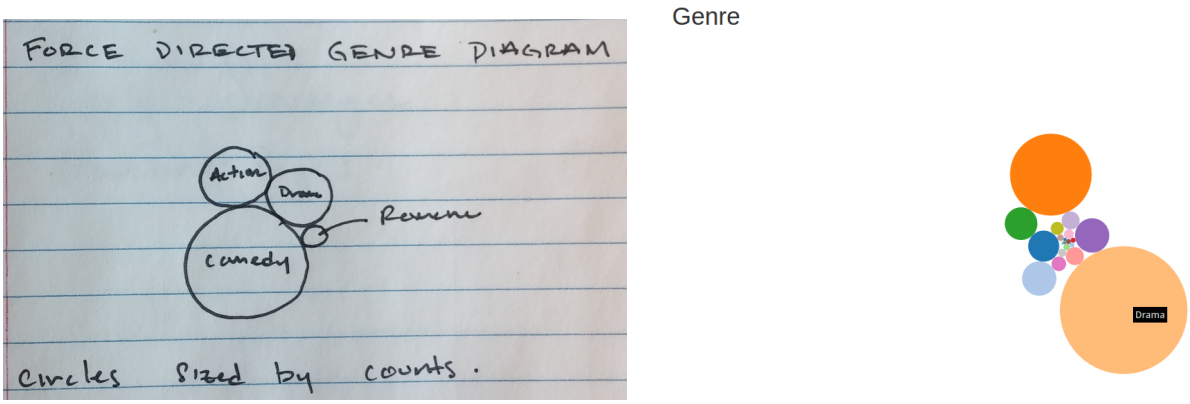
(a) Collection of co-stars for Owen Wilson.

(b) Common films between Owen Wilson and Ben Stiller

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.

4.5 Genre Visualization



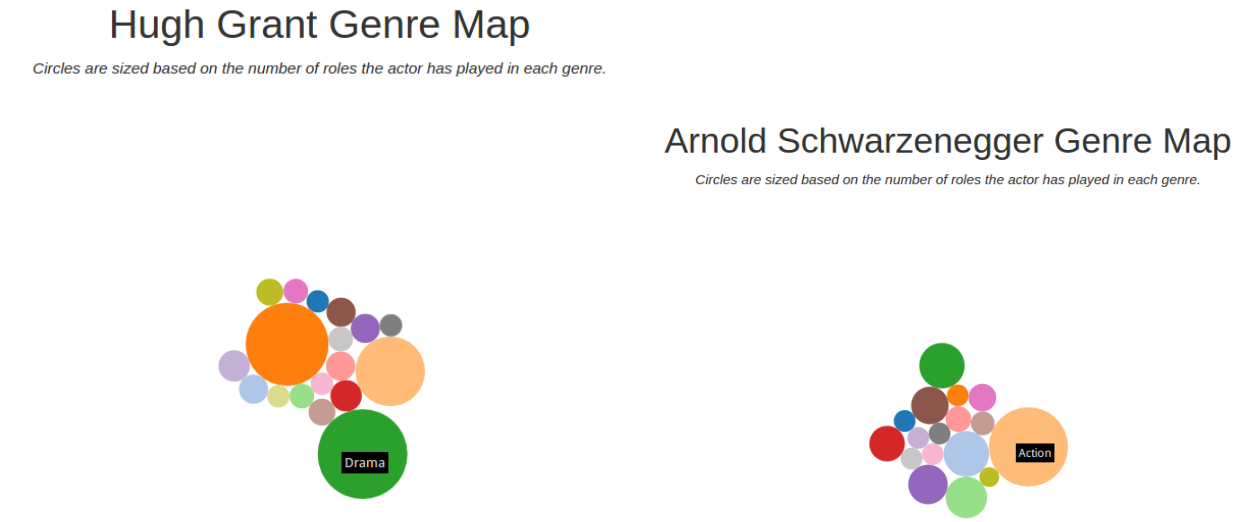
(a) Original design sketch

(b) Current implementation

Figure 11: Genre visualization development

The GenreVis view (figure 11) was created to visualize an actor's breadth of work or whether they are disproportionately cast one type of genre.

In this view, 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 lable 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.



(a) Hugh Grant. The Other large orange dot is 'Romance'.

(b) Ahnold. No surprises here.

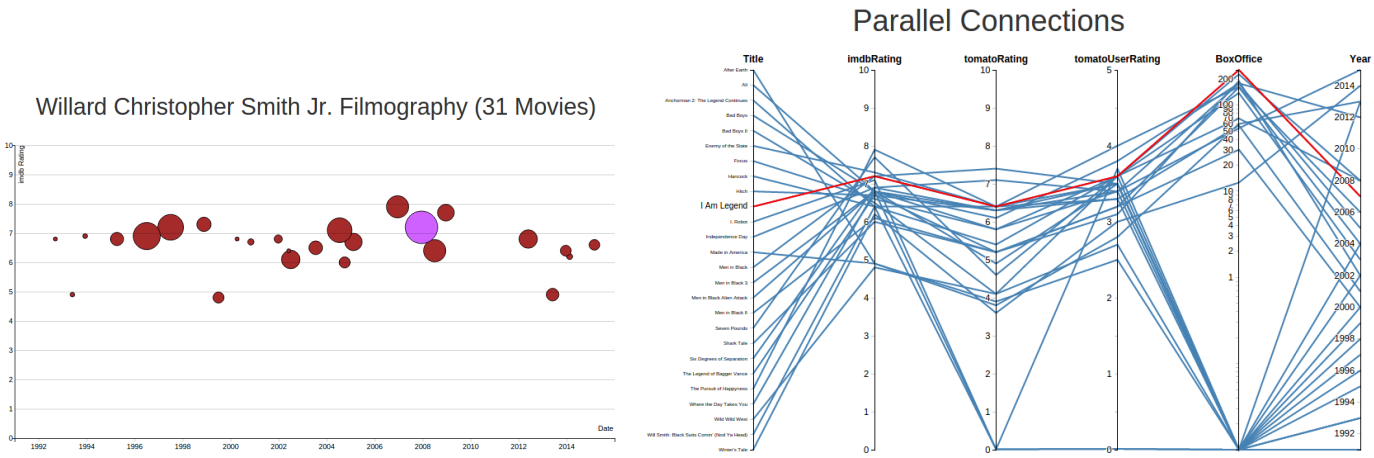
Figure 12: specific Actor Genre visualizations

5 Connected Views

To make the main views more interactive, we used the broadcast model of event handling from homework 4 to connect our filmography and Parallel Axis views. Selecting films in one of the views will highlight the corresponding entry in the other view. Brushing in parallel axis view will highlight multiple films in the timeline (figures 13, 14).

These connected views are especially useful when applying multiple filters in the parallel axis coordinate view and seeing the highlighted distribution of films in the timeline graphic.

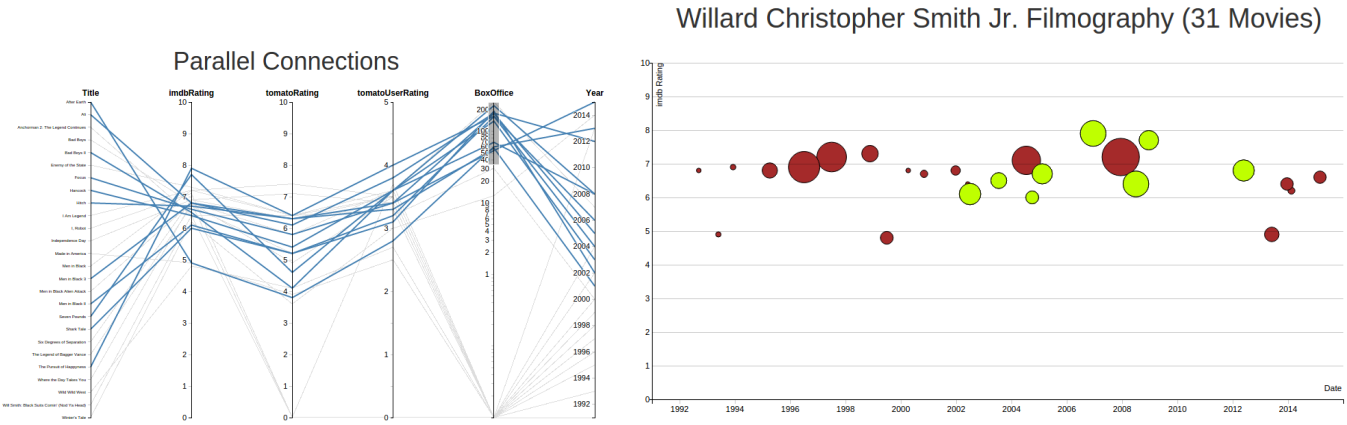
We thought this functionality would help uncover actor trends. For instance, the bulk of Will Smith's highest grossing movies all occur in 2000-2010 (figure 14b).



(a) User-selected movie from Will Smith's Filmography

(b) Also highlighted in the Parallel Connections View

Figure 13: Event-driven highlighting. Selections from the Filmography are highlighted in the Parallel Connection view.



(a) User-brushed selection from Parallel Connections view.

(b) Also highlighted in the Parallel Connections View

Figure 14: Event-driven highlighting. Selections from the Filmography are highlighted in the Parallel Connection view.

6 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.

7 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, tree map, and parallel coordinate view.

Jimmy Moore: Jimmy was the project scribe and responsible for proposal and final report documentation. He also created the filmography timeline visualization and contributed code for the API Database storage and site formatting.

8 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
    - Add different axes to filter against
    - Collect additional film data (perhaps from a separate source) to fill in Box office earings numbers which appear to be missing from a majority of the API calls.
3. : Treemap view
    - General cleanup and exploring other shapes
    - Link view to highlight corresponding films in the Filmography Timeline
4. Genre view
    - Create a legend for the various colors
    - Explore other methods of displaying this information (Bar charts?)
    - Label the dots directly
    - Link view to highlight corresponding films in the Filmography Timeline
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.