Curly-Squeegee Process Book

Brian Kimmig & Jimmy Moore

December 3, 2015

Abstract

Curly-Squeegee is a web-bsaed 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 Editing notes

Depending on how thourough 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. Analysis of data
4. 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 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?

2.2 Intended Audience

Curlee 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.

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 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.

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

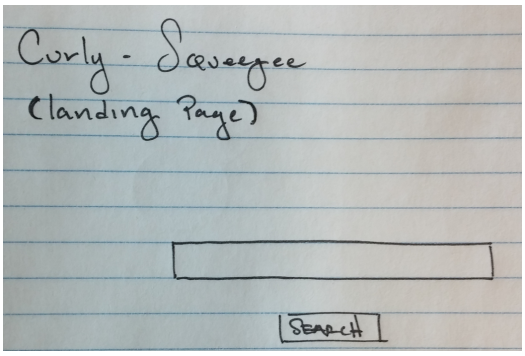
Due to our accelerated development schedule, we were able to move from sketch book to implementation very quickly. We brainstormed multiple views which could help answer the questions raised in section 2.1 for a variety of actors. We also sought to encourage user interactivity and re-use.

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

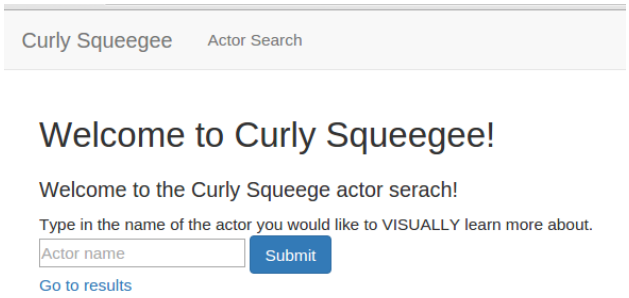
The following subsections discuss the implementation and styling each interface, and how or why it evolved during the life of the project.

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,

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.

4.2 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.

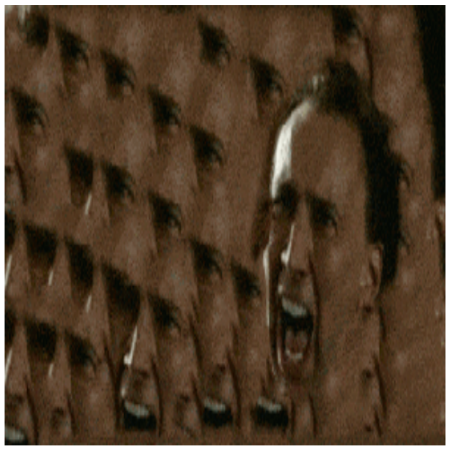


Figure 2: After the user submits a search query, the application waits for the API calls to complete and fetch the actor's filmography. An animated gif can help pass the time.

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

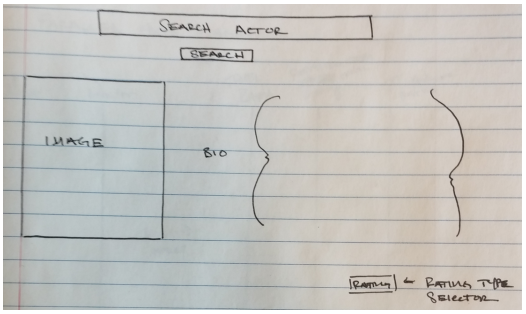
5 Visualizations

When the API calls are complete and the data is saved in our database, we load a "results page", which showcases our multiple views and actor information.

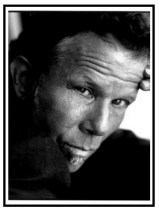
5.1 Actor Fact Box – Depricated

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



- **Name: Thomas Alan Waits**
- **Birthday: 7 December 1949**
- **Number of Films: 130**
- **etc:**

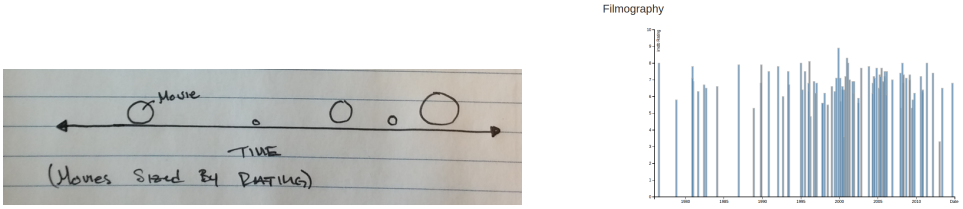
(b) Original implementation

Figure 3: 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 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 8), 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 4: 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 4b). 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

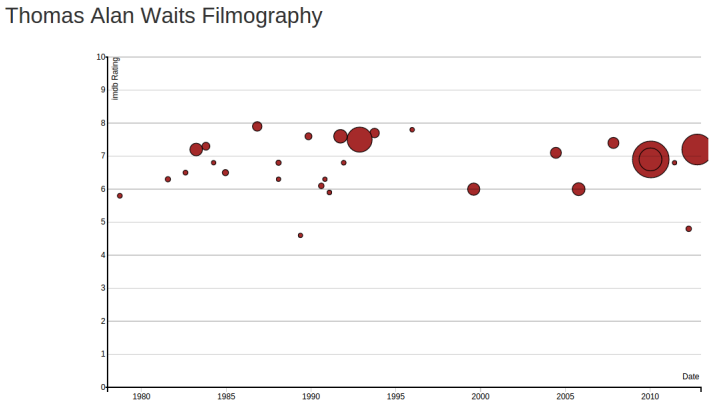


Figure 5: XY Plot of Tom Waits filmography. circles are sized by number of IMDB votes.

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.

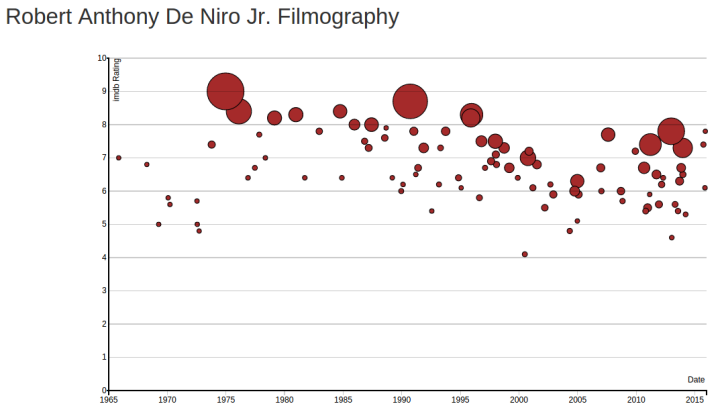
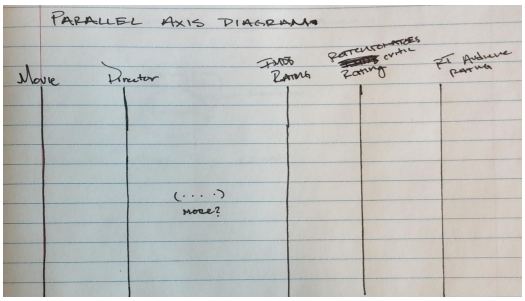


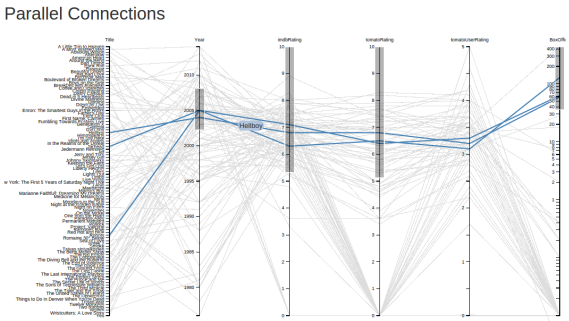
Figure 6: We've seen better times.

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. We looked forward to determining the highest grossing Schwarzennegar movie from the 80's, or.... **More**



(a) Original design sketch

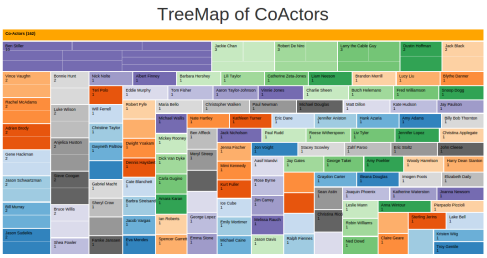


(b) Current implementation

Figure 7: Parallel Axis Coordinate visualization development

5.4 Treemap Visualization of co-workers

The treemap visualization was a last-minute addition to our project. Despite TA recommendation for such a view, We did not orginally plan to visualize a given actors relationship to other co-stars. However, after the "Visualizing Maps and Trees" lecture, we saw a way to group co-stars which have appeared in multiple films alongside the queried actor.



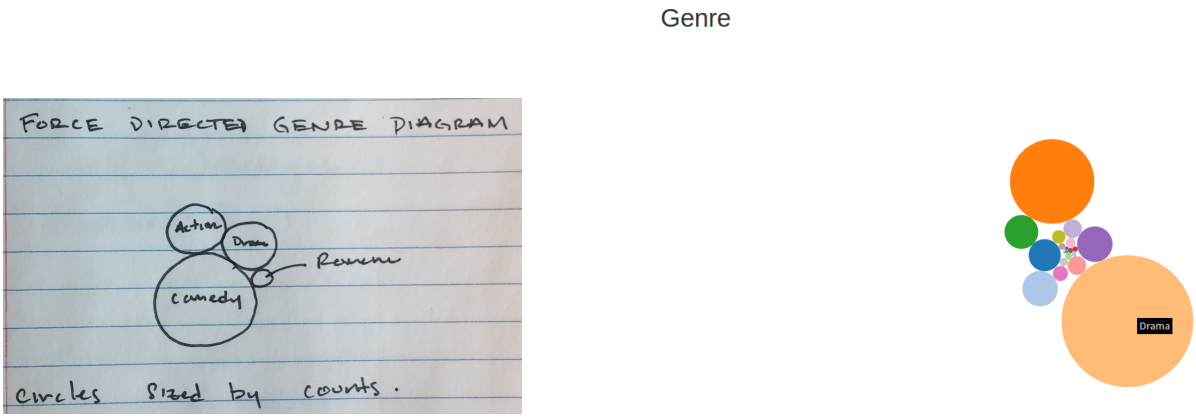
(a) Collection of co-stars for Owen Wilson.

(b) Common films between Owen and Ben Stiller

Figure 8: Tree Map visualization for Owen Wilson's career

This view was accomplished in a similar way to the GenreVis view.....

5.5 Genre Visualization



(a) Original design sketch **(b)** Current implementation

Figure 9: Genre visualization development

The GenreVis view (figure 9) 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 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 thrille'r or 'romantic comedy') would count each descriptor towards the actors genre aggregate.

This view has remained fairly stagnant since we added it, in favor of working on other aspects of the project.

TODO: link views.

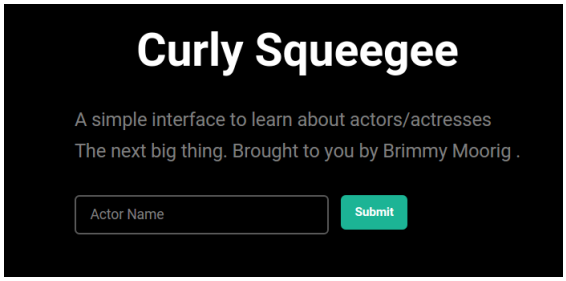


Figure 10: Final version of landing page.

6 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 the **XXXXX** located here for it's stripped-down appearance, and then **This JS stuff** for the smooth strolling site-linking functionality. Since all of our content is essentially on on page, we wanted a clean way to transition between views.

7 Connected Views

We chose to interconnect the timelineVis and parallel axis coordinates visualizations so any selected movies in one view would be highlighted in another. This is especially useful when brushing multiple items in the parallel axis coordinate view and seeing the highlighted distribution of films in the timeline graphic.

This is most useful when visualizing the temporal distrobution of films brushed from the parallel axis plot. We thought this functionality would help uncover any time trends with given film earnings, directors, etc.

The results of these searches are dependent on the actors, of course.

8 Project Feedback

After we settled on our design and project implementation, we had the opportunity to present a "sales pitch" for Curly-Squeegee to another project team to get their feedback. We met with 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.

9 Team Evaluation

Brian Kimmig: Brian was responsible for the 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, and the Actor filmography visualization, along with site design and formatting. He also was the project scribe and responsible for project documentation.

10 Future Work

Awesome project and we want to keep working. Lots of stuff we could do

- Add selection/links/buttons to change search query from filmography to director/producer, soundtrack, or other creative capacity. Additionally, these multiple datapoints could be overlaid on the same graph and separated with color.
- Search for multiple actors and compare their filmographies on the timeline visualization.
- 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.