

2.1 Visualizations that use lines

Naturally xkcd/657 itself (see Figure 1) is the single greatest example of a related work in this field, considering, of course, that it is precisely what we are trying to replicate.

Other examples include:

Napoleon's March to Moscow by Charles Joseph Minard (1)

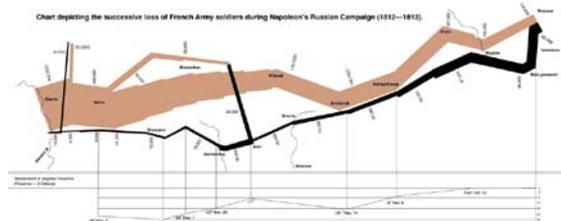


Figure 2: Napoleon's March to Moscow

While the lines in Figure 2 encode the army in a different manner than that of xkcd/657 the overall appearance of the visualization is very similar. There is also a great similarity with regards to the distortions that are added for the sake of clarity and aesthetic. A notable example is that the advancing and retreating army probably used some of the same roads but in Minard's representation the lines do not overlap but rather travel together.

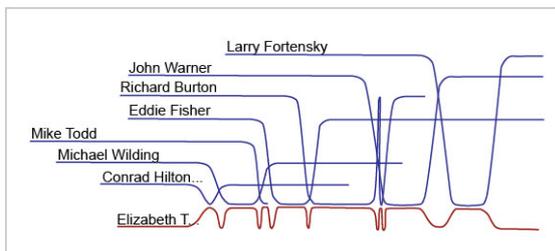


Figure 3: Genealogy Visualization project at Stanford

The above visualization is remarkably similar to what we are trying to do. Lines represent people and time is represented along the x-axis. The lines come together to show marriage and then return to their original vertical placement in the event of a divorce.

2.2 Tools that automate visualizations

As it happens pretty much every tool that allows users to create visualizations, whether by importing/analysing data or by direct manipulations fits into this criteria. Indeed even the simple pie chart tool in Microsoft Excel (2) (Figure 4) is in a sense allowing the user to quickly and effortlessly create something that at one point in time had to be hand drawn. Only a small selection of these tools will be mentioned here.

The popular spreadsheet applications, like Microsoft Excel, Apple Numbers (3) and Google Spreadsheets (4) have brought the basic chart creation functionality to the everyday consumer.

They are very heavily used and all work in roughly the same way: the user highlights some data in a table, selects the chart type (Figure 4) and then adjusts the colors and labels as necessary.

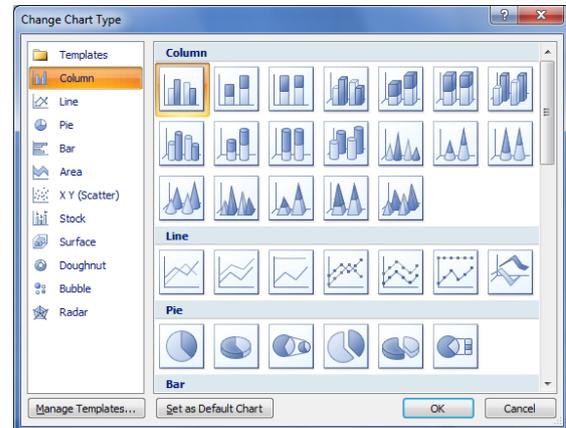


Figure 4: Microsoft Excel 2007 char type selection dialog

3 METHODS

PlotWeaver presents two challenges: The layout that requires data structures rich enough to represent the lines and their interactions, and the user interface which has to be intuitive enough for users to understand.

3.1 Layout

The layout is done to mimic (as much as possible) the visualization style of xkcd/657.

3.1.1 Line placement

Character lines flow from left to right, and at certain points come closer (10px) to each other to represent interaction. To encode this we divided the space vertically into pacing events or **Time Steps** that are defined by a change of character interaction.

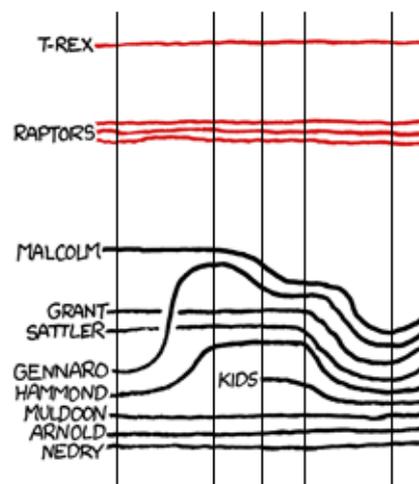


Figure 5: xkcd/657: Jurassic Park with time steps

Figure 4 demonstrates how these time steps, represented by the vertical lines, divide the space.

Each character that exists during the time step crosses it; and at the point of this crossing we define a character point, or **Char Point**, thus each character would have as many char points as time steps in her existence and each char point would have a logical coordinate (the horizontal part of which would be shared between all the char points at the given time stamp).

Each char point would also be part of a **Point Group**. The point groups group char points that are in the same time slice and that happen to be interacting. All char points within the same point group will necessarily maintain a constant distance from each other so as to visually convey the interaction.

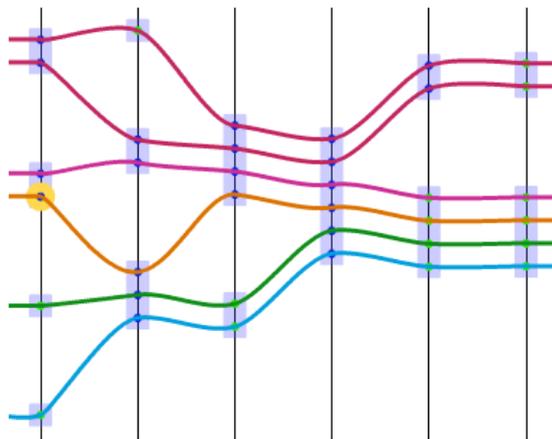


Figure 6: This PlotWeaver sample shows time slices, char points, and point groups can be seen represented with vertical lines, circles, and rectangles respectively. One char point is selected.

Since two char points in the same time slice that belong to different point groups must necessarily not be interacting the point groups are programmed to stay away from each other vertically by at least some minimal distance that is necessarily greater than the distance between adjacent char points in the same group.

Finally the lines are Bézier splines that go through every char point (as an anchor points). The angles of the tangents going through points within the same group are measured and averaged so that the lines appear to flow together more and to better mimic the layout of xkcd/657 (See Figure 6).

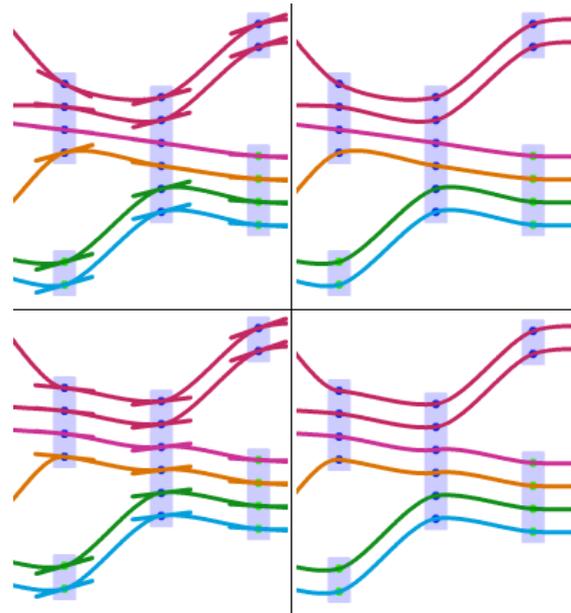


Figure 7: The naive approach without tangent averaging (above) is compared to the improved approach with tangent averaging (below). Shown with and without the tangents

3.1.2 Label placement

Just like in xkcd/657 there are three types of labels: labels placed at the start and end of lines that bear the character's name, labels placed over important event ovals (Figure 9), and labels placed over lines to mark the line during its path. The first and second kinds of labels are trivially implemented using the Flex 4 (5) Label class (6) and will not be farther disused here. The line labels are implemented as following a cubic Bézier curve – the curve is traced with small increments and letters are placed at the correct points along the curve and then rotated to align with the tangent at that point. This effect creates a text label that appears to follow the curve. Figure 7 demonstrates the curved text and also the effect of applying a glow filter (7) to create the label-breaks-line effect seen in xkcd/657.

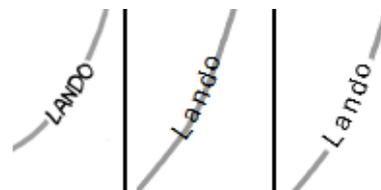


Figure 8: Effects of glow filter on labels. Shown (from left to right) original xkcd/657 label, line label without glow and line label with glow.

3.1.3 Decorations

Some extra decoration is added to emphasize important events like the death of a character (Figure 8).



Figure 9: An important event is marked with a gray bubble and the death of a character is signified by a bulb at the end of his line. *xkcd/657*, left and *PlotWeaver*, right.

These decorations while being trivial to implement add a lot to the finished visualization.

xkcd/657 had been hand drawn and in regular XKCD fashion its lines are not straight which gives it a remarkably warm feeling by making the character lines appear more 'human'. To simulate this hand drawn effect on the computer Perlin noise (8) displacement was used (see Figure 10).



Figure 10: Bitmap filled with Perlin noise - Perlin noise generates naturally looking noise by adding up different octaves (akin to music) and can be used to generate anything from clouds to sketch-like line

This effect greatly improves the 'warmth' of the graph and also serves to liken it even more to *xkcd/657*.

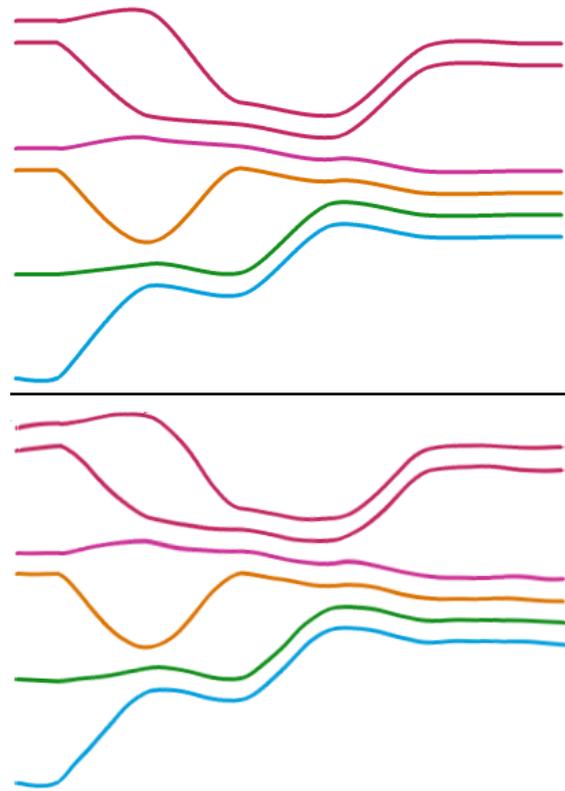


Figure 11: *PlotWeaver* lines with (below) and without (above) application of Perlin noise displacement to make the lines look more sketch like

3.2 User interface

The user interaction with *PlotWeaver* is designed to happen in three steps. First the user adds the time steps and the characters that are involved in the plot and weaves them together as necessary. Once the user is satisfied with the plot flow, she could spend some time untangling the lines to make the visualization more readable (with the aid of the untangle tool) and moving individual char points to make the plot more visual aesthetic and/or to convey some other meaning (with the aid of the force directed layout). Finally she will finish the visualization by adding labels to the appropriate points on the line.

Although these steps are by no means hard coded, indeed, user might want to go back and add remove or correct a character after having done the layout, they will be examined separately here for clarity.

3.2.1 Plot weaving

The tools provided allow for very straight forward plot weaving. Indeed, partly by design, partly by accident the plot weaving feels a lot like braiding hair (that is growing left to right).

The user is expected to make as many characters and time slices as required and then to weave them together using the 'Merge' and 'Split' tools.

3.2.2 Untangling the layout

After the user is finished with adding and weaving the characters she may use the genetic algorithm that would attempt to sort the characters in such a way as to make them intersect as little as possible. This greatly improves the look of the visualisation (see Figure 12).

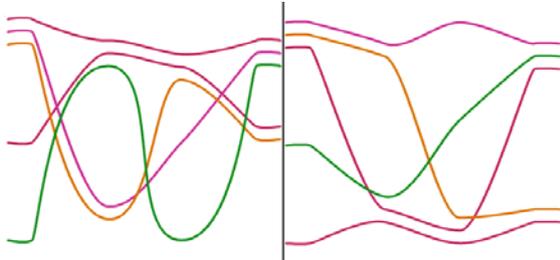


Figure 12: Before (left) and after (right) running a genetic algorithm to uncross the layout. The number of crossing is reduced from 10 to 2.

The genetic algorithm uses the character sort order as the gene it then mutates it by permutating it at random. No crossover function is implemented. It was found to be much more effective to simply represent one single character ordering rather than one for each time slice. This follows logic since if the character is on top in time slice N then said character will likely remain on top in time slice N+1. Any permutations applied to the sort order of characters in a single time slice must necessarily be applied to every subsequent time slice as well.

3.2.3 Smoothing the layout

Once the layout has been satisfactory untangled the user might wish to apply the force directed layout which will work to reduce the distance between lines and eliminate zigzags (see Figure 13).

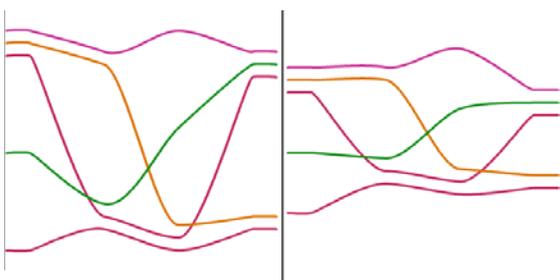


Figure 13: Before (left) and after (right) running the force directed layout to smooth the lines.

The force directed layout works on three forces: a spring like force between adjacent char points on the same line that pulls them closer together, a magnetic like force between adjacent char points on the same line that snaps them completely straight if they come close to being straight and a magnetic like repulsion force between adjacent point groups on the same time slice that makes sure that

uninteresting characters do not appear to be interacting.

The user is able to move (using the Move tool) any char points to a different predefined location. There is also the option of pinning a char point in place such that any subsequent runs of the layout force directed layout algorithm will not affect it. In this way a user might choose to keep some special character's line completely straight, even though the force directed layout might want to bend it left to its own devices.

3.2.4 Label placement

Line labels (see also section 3.1.2) can be randomly scattered throughout the length of the lines with the 'Relabel' tool but in general the user will probably want to position the labels herself. Label positioning is trivially achieved with the 'Label' tool that, once selected, toggles the label at the char point at which it's clicked.

3.2.5 Saving, Loading and Exporting

PlotWeaver makes use of the FileReference (9) feature built into Adobe Flash Player 10 (10) which allows the application to save and load files (with the user's permission). The content of the visualization is, therefore, saved as XML (11) formatted PLT (plot) files.

PlotWeaver can also export the visualization to PNG and JPEG file formats.

4 RESULTS

We have already shown many screenshots demonstrating certain PlotWeaver outputs in the preceding sections. We now look at some real world plots. Unfortunately the nature of the layout requires very wide landscape images which cannot be inserted into this paper without losing a lot of detail. The example visualisations in this section are also available online, together with their PlotWeaver project files (12).

4.1 The Star Wars Trilogy

The first example presented here is an attempt to replicate the Star Wars (13) section of xkcd/657. Figure 14 demonstrates a side by side comparison of the original and the PlotWeaver replica. It should be noted that the replica as made from the xkcd/657 visualization (and not from actually watching or remembering the movies) and that special care was put into ordering the characters in an ordering similar to xkcd/657.

The replica visualization indeed looks very similar to the original, all be it a bit less tidy. This example is complicated enough to reveal several improvements that could be done to the layout algorithm; in particular the uncrossing algorithm should not, if at all possible, cross the lines of

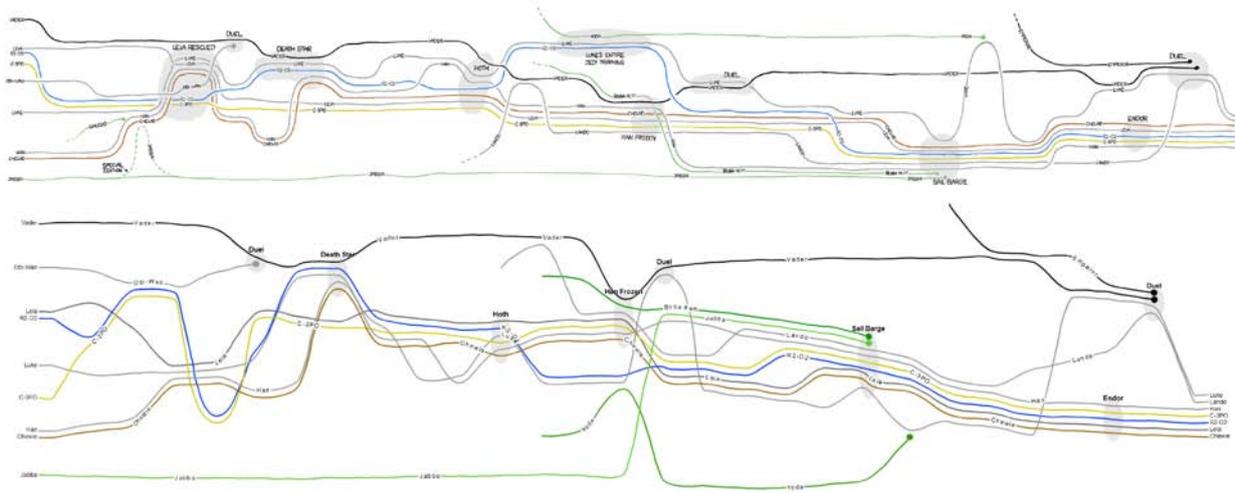


Figure 14: A comparison between the original XKCD/657 Star Wars visualization (above) and the same visualization remade in PlotWeaver (below). The PlotWeaver clone took 33min to make from scratch. Due to the huge width of these images they are not properly displayed here.

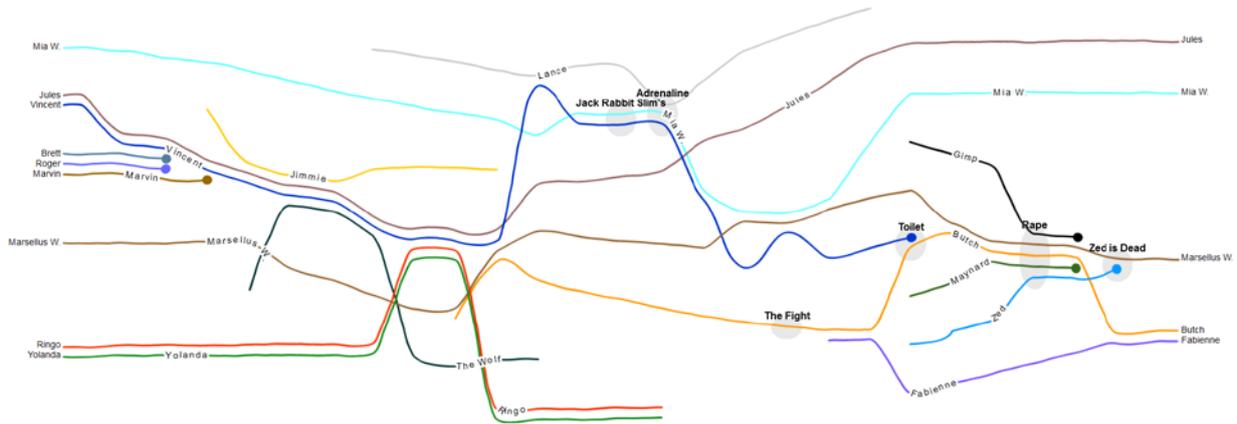


Figure 15: A PlotWeaver plot of the movie Pulp Fiction (20). Constructed in sequence with watching the movie (154min).

characters that are interacting as it makes the visualization look messy.

This plot replication was done with the aim of demonstrating an impressive plotting time. The whole thing was done in 33 minutes, which is indeed a very good result when compared to the time needed to produce the (hand drawn) original, which probably took many hours.

4.2 Pulp Fiction

The second plot demonstration is of the 1994 Quentin Tarantino classic, Pulp Fiction Figure 15. This plot was done concurrently with the author watching the movie (on a computer equipped with two screens). A decision was made to not represent the minor characters from start to finish and thus the visualization looks very neat and tidy.

It should be noted that the act of construction the plot while (re)watching the movie is not only doable, it is a profoundly fun zen-like experience that connects one with the movie.

We highly recommend you give it a go.

5 DISCUSSION

5.1 Effectiveness of PlotWeaver

We hope that the screen shots and examples demonstrated in the previous section allow the reader to form their own opinion of the success or failure of PlotWeaver at achieving its goal of effectively replicating xkcd/657 style. Due to the complicated nature of xkcd/657 and the intricacies added to it by hand it would be unreasonable to expect PlotWeaver to be able to replicate it point for point. The users who have used PlotWeaver thus far are in consensus that is indeed very good, and that the slightly inferior quality of visualisation produced (when compared to xkcd/657) is more than justified by how quick and 'fun' the process of making it is.

adjustments of the constants in the fitness and energy functions will produce visibly better results.

Another important area of improvement would be to make the layout algorithms clever enough so that when a user does a structural change to the plot data (such as add a character or an event) the layout does not have to reset itself to the base positioning.

The layout algorithm can also be extended to encode more information in the vertical position of the character. Just like in the Lord of the Rings part of xkcd/657, where up and down correspond loosely to northwest and southeast respectively. One other application of this could be a strict correspondence where, if the entirety of the plot is divided exactly between two places (say two planets), then those could, perhaps, be represented with the top and bottom halves of the visualisation.

6.1.5 Fat Splines

Just like in the Lord of the Rings part of xkcd/657 the thickness of the lines can be used to encode extra information (such as strength of an army).



Figure 18: A fat spline (16)

There already exist components for implementing variable thickness splines (as shown in Figure 11) but it would need to be deconstructed a bit so that tangent angle averaging (Section 3.1.1) can be implemented.

6.2 Functionality altering future work

This section will discuss the more imaginative and complex future work that could be done on PlotWeaver.

This section is ranked from the more interesting and doable to the more outrageous and insane additions.

6.2.1 Wiki-like movie plot database

A collaborative, freely editable, Wikipedia (17) like web site could be set up where users can freely edit and create the visual representations of the plots of movies. This crowd sourcing would generate interesting usage patterns, and no doubt, will eventually include most of the movies ever made.

6.2.2 Automatic movie script/IMDB (18) parsing

A movie script or IMDB quotes page could be imported into PlotWeaver and with the application of natural language processing which would then automatically produce the plot visualization.

6.2.3 Movie parsing with face detection

A movie could (in theory) be parsed frame by frame and character interactions could be determined from their occurrences within the same frame.

ACKNOWLEDGMENTS

The author wishes to thank Randall Munroe, Mike Bostock and Jeffrey Heer, for their inspiration, feedback and guidance.

REFERENCES

1. **Minard, Charles J.** Napoleon's March to Moscow.
2. **Microsoft.** Excel. [Online] <http://microsoft.com/excel>.
3. **Apple.** Numbers. [Online] <http://www.apple.com/iwork/numbers/>.
4. **Google.** Docs. [Online] <http://docs.google.com/>.
5. **Adobe.** Flex 4. [Online] <http://opensource.adobe.com/wiki/display/flexsdk/Gumbo>.
6. —. Flex 4: Label. [Online] http://help.adobe.com/en_US/Flex/4.0/langref/spark/components/Label.html.
7. —. Flex 4: GlowFilter. [Online] http://help.adobe.com/en_US/Flex/4.0/langref/spark/filters/GlowFilter.html.
8. *Improving Noise.* **Perlin, Ken.** <http://mrl.nyu.edu/~perlin/paper445.pdf>.
9. **Adobe.** Flex 4: FileReference. [Online] http://help.adobe.com/en_US/Flex/4.0/langref/flash/net/FileReference.html.
10. —. Flash Player 10. [Online] www.adobe.com/products/flashplayer.
11. XML. [Online] <http://www.w3.org/XML/>.
12. **Ogievetsky, Vadim.** PlotWeaver project page. [Online] <https://graphics.stanford.edu/wikis/cs448b-09-fall/FP-OgievetskyVadim>.
13. **Lucas, George.** *Star Wars*. Lucasfilm, 1977.
14. Scalable Vector Graphics. [Online] <http://www.w3.org/Graphics/SVG/>.
15. **Adobe.** Illustrator. [Online] www.adobe.com/products/illustrator/.
16. **McLaren, Daniel.** Actionscript 3.0 Splines with Variable Thickness. [Online]

<http://danielmclaren.net/2008/02/actionsript-3.0-splines-with-variable-thickness>.

17. **Wikipedia**. [Online] <http://www.wikipedia.org/>.

18. **IMDB**. [Online] <http://www.imdb.com/>.

19. **Munroe, Randall**. Movie Narrative Charts. *xkcd*. [Online] <http://xkcd.com/657>.

20. **Tarantino, Quentin**. *Pulp Fiction*. A Band Apart, 1994.