The Panta Rhei: Modernizing the Marquee

Megan Monroe(✉) and Mauro Martino

IBM Research, Boston, MA, USA
madey.j@gmail.com

Abstract. Many multimedia visualizations abstract the underlying content into aggregate displays, requiring user interaction in order to expose the original text, images or video. The drawback of this approach is that unlike traditional, numerical data, multimedia data is readily interpretable. Users can catch an image or phrase out of the corner of their eye an immediately understand the content. However, this passive discovery cannot take place when content is only exposed through direct interaction. In this paper, we present the Panta Rhei, a peripheral display designed to avoid this pitfall by surfacing original content when the user is not actively engaged with the application. We provide the full implementation details, including the many ways in which the underlying parameters can be tuned to suit various objectives. Since the display can easily support text, images or videos, our goal is to enable more widespread discussion and experimentation involving this technique for multimedia visualization.

Keywords: Visualization · Animation · Passive · Engagement-versatile

1 Introduction

This work began as a larger project to visualize the news in real time. The goal was for users to understand which events were trending, who was involved, and which stories were sneaking under the radar. Accordingly, a number of visualization strategies were employed to extract and link entities [8], to identify keywords and concepts [13, 14], and to gauge sentiment [6]. However, there were two critical drawbacks to these aggregate displays:

1. Without interaction, the display remained relatively unchanged until the next big news story broke.
2. Without interaction, and with limited screen space, only one article was being displayed in detail at any given time.

In this paper we present the Panta Rhei (Greek for "everything flows"), a web-based animation that scrolls article titles across the browser window as an infinite, mesmerizing stream. The Panta Rhei is triggered when there has been no interaction for a set period of time. It demands no active attention or interaction, and yet was surprisingly adept at triggering serendipitous discovery from the periphery during its initial deployment [9].

The contribution of this paper is primarily technical. We detail the Panta Rhei's scrolling mechanism, which allows developers to feed in customized content without

R. Bernhaupt et al. (Eds.): INTERACT 2017, Part III, LNCS 10515, pp. 366–373, 2017.
DOI: 10.1007/978-3-319-67687-6_24

substantial code modifications. We test and report on the small number of underlying parameters that can be tuned to achieve a wide array of functional and aesthetic objectives, including piquing the user's attention when the display is receiving only peripheral attention.

2 Background

Many text and multimedia visualizations dedicate the majority of screen space to computed abstractions of the underlying content [1, 3, 7]. Even so, the seeming consensus is that the original content must be accessible to prevent misinterpretation [15]. This is typically accomplished through active interaction [10]. This work explores the other side of this equation, mapping the exposure of original content to physical time when interaction is not taking place or has become undirected.

The Panta Rhei is intended to serve as a periphery-passive or focus-passive display in an engagement-versatile application. However, unlike Tanahashi's Stock Lamp [12], the Panta Rhei displays unabstracted content. When a particular image or phrase catches the user's eye, they are seeing the *actual thing* that is interesting, not an abstracted aggregate. The Panta Rhei is also unique in that it is designed to be reusable. It is structured to allow developers to flexibly feed in any combination of text, images, and videos. The display can simply be plugged into existing applications to make them more engagement-versatile. Though its design adheres closely to related work involving ambient displays [5, 11], the aesthetic details are beyond the scope of this paper.

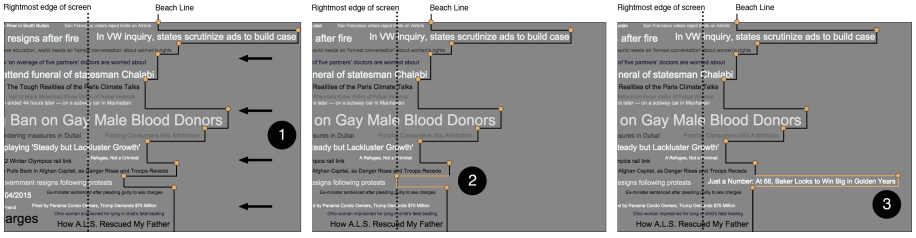

Fig. 1. The beachLine sits just offscreen (note that the dotted line represents the end of the screen) and works to ensure a steady stream of content. (1) Content scrolls steadily onto the screen, moving from right to left (2) when a tick has scrolled fully onto the screen, the beachLine is notified to backfill the empty space (3) a new tick is added to the beachLine, which is adjusted by the width of that tick.

3 Implementation

The Panta Rhei modernizes the classic news ticker marquee with inspiration from Steven Fortune's sweepline algorithm [4]. The web-based implementation scrolls a tightly-packed grid of heterogenous content across the browser window (or a specified of the browser window). Each item of content, referred to as a "tick", is

contained in a and can thus be comprised of any web element (image, video, text). This paper will focus on text, and consequently, a right-to-left scrolling such that the beginning of the text leads onto the screen. The display is powered by a data structure called the beachLine, a set of points that descends vertically from the top of the screen to the bottom. Each of these points keeps track of a tick that is currently in the process of scrolling onto the screen. Conceptually, the beachLine sits just out of view, beyond the rightmost edge of the screen, and is updated every time a tick scrolls fully into view (see Fig. 1).

New ticks are added to the display in a just-in-time fashion. That is, ticks are created such that their leftmost edge is flush against the rightmost edge of the screen and they immediately begin their scroll across the display. When a tick is created, it is armed with two future actions. First, the tick will notify the beachLine when it has scrolled fully onto the screen. This is the beachLine's cue to backfill the space that the tick is leaving in its wake. Second, the tick is set to remove itself from the page entirely once its rightmost edge has scrolled beyond the left side of the screen. This prevents the visualization from queuing up undue memory usage. These three stages of a tick's lifecycle are depicted in Fig. 2.

Fig. 2. (1) A new tick is created with its leftmost edge flush against the rightmost edge of the screen (2) once the tick has scrolled fully onto the screen it notifies the beachLine to add new content (3) when the tick has scrolled fully beyond the leftmost edge of the screen it removes itself from the page entirely.

3.1 Updating the beachLine

The beachLine is comprised of a set of points that descend vertically down the screen, each consisting of an x-coordinate, a y-coordinate, and the id of the tick that generated it. However, as it is shown in Fig. 1, it is easier to think of the beachLine as a series of vertical facings against which new content can be aligned. These facings can simply be extrapolated from the points.

When a newly created tick has scrolled fully onto the screen, it notifies the beachLine by submitting its y-coordinate (i.e. its vertical position on the screen). This initiates a two-phase process in which the beachLine first locates and determines the

height of the facing on which that y-coordinate falls. This height is served to the developer, who in turn supplies any , or tick, that does not exceed the allotted height. Thus, developers can employ a variety of content and layout strategies by customizing only a single function, getNewTick(), which is shown in Fig. 4. They can return a tick that exactly fits the space. They can return a smaller, vertically centered tick. They can add multiple ticks. Or no ticks at all.

The new content, consisting of zero or more ticks, is then is fed back to the beachLine to be incorporated into its point system. This is done by increasing the x-coordinates of the beachLine points by the width of the new ticks, and adding points as necessary. The four possible reconfigurations of the beachLine that can result from adding a new tick are shown in Fig. 3. What is critical to note about these updates is that, while the y-coordinates of each beachLine point represents a true y-position on the screen, the x-coordinates continue to increase monotonically as new ticks are added to the beachLine. Thus, the x-coordinate of a beachLine point represents only its *relative* x-position compared to the other points on the beachLine. Each new tick is also added to a pool of activeTicks, which indicates that the beachLine is expecting to eventually receive its backfill request. Ticks are removed from the pool as these requests are received.

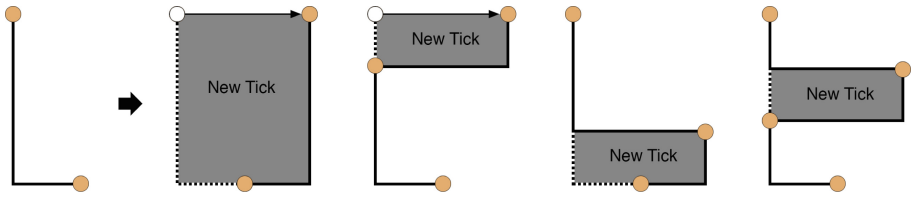

Fig. 3. Given a vertical facing on the beachLine, which is defined by two consecutive points (left), a new tick can be added in any of the four ways shown on the right.

3.2 Maintaining the beachLine

The continual updating of the beachLine eventually results in an extremely ragged edge, with vertical facings too short for the intended content. To account for this, two operations are performed on the beachLine each time a backfill/update call is made.

Shoring the beachLine: As discussed in the previous section, developers do not need to fully fill an open facing when it becomes available. However, the space that is not filled (now its own facing) becomes an empty section of the screen without an active backfill timer. If left unchecked, these unused facings slowly consume the entire screen.

Because the x-coordinates of the beachLine points are relative, we do not know the absolute position of the beachLine at all times. However, we can infer it momentarily when a backfill request is initiated because such a request means that the rightmost edge of a tick, and thus its corresponding beachLine facing, is flush with the rightmost edge of the screen (see Stage 2 of a tick's lifecycle in Fig. 2). By obtaining the x-coordinate of this beachLine point, we can infer which sections of the screen have gone unfilled because their x-coordinates will be lower, meaning that these facings

have already scrolled onscreen. Thus, when a backfill request is received, these empty facings can be reclaimed by pushing their x-coordinates up to match facing being backfilled, a process referred to as "shoring" the beachLine.

Cleaning the beachLine: Once the beachLine has been shored, adjacent sections can be merged together to form larger facings, a process referred to as "cleaning" the beachLine. This process is dictated by a single parameter that tells the beachLine how close the x-coordinates of adjacent facings must be in order to be merged, which can be tuned to produce larger or smaller facings. When two facings are merged, the new facing assumes the larger of the two x-coordinates, which prevents ticks from overlapping, and the tick id associated with lower x-coordinate is removed from the activeTicks pool in order to prevent a duplicate backfill. Shored facings will always be merged when they are adjacent to the facing that generated the backfill request, since their x-coordinates will be equal. Thus, the core Panta Rhei algorithm is a self-perpetuating process of updating, shoring, and cleaning the beachLine. The pseudocode for this entire process is presented in Fig. 4.

4 Extensions and Control

While the process of updating, shoring, and cleaning the beachLine provides the core Panta Rhei functionality, there are some additional details and features of the implementation that allow developers to better control the display. While this list is not meant to be exhaustive, much of the Panta Rhei's additional functionality (precision tick slotting, handling screen resizing) is derived from either a slight modification to or a combination of the following strategies.

Override Backfills: For a number of reasons, it is necessary to backfill a section of the screen even if there is no corresponding entry in the activeTicks pool. This can be done by incorporating an "override" id into the checkActiveTicks() function that will always allow the backfillTick() function to proceed. In particular, this tactic is used to initialize the display.

Pausing and Restarting: Pausing the Panta Rhei's scrolling is accomplished with two actions. First, the activeTicks pool is emptied, ensuring that no new content will be added to the display when the backfill timers fire. Second, the animation assigned to each tick is halted and removed. Restarting the display then, requires an analogous process. First, every tick on the screen is reissued an animation that will complete its scrolling and remove it from the display. Second, any tick that is intersecting the rightmost edge of the display is reissued a backfill timer that will go off when the remainder of the tick has scrolled onto the screen.

Splitting the Stream: This feature was originally requested so that news headlines pertaining to multiple entities could be compared. Splitting the stream is accomplished by adding points of the form `{x: -1, y: y-positionOfSplit, id: "split"\}` to the beachLine. A "split" point acts like a pillar, shoved into a riverbed - content is forced to flow around it. The shoring and cleaning functions are updated to skip over any beachLine points with the "split" id and, similarly, the getFacing() function returns a

```
function backfillTick (id, y){
  //MAINTAIN BEACHLINE
  shoreBeachLine();
  cleanBeachLine(mergeDiff);

  if( ! checkActiveTicks(id) ) return;

  //GET FACING HEIGHT
  var facing = getFacing(y);
  var currY = facing.top;

  while(currY < facing.bottom){
    var height = facing.bottom - currY;

    //CREATE A NEW TICK
    var tick = getNewTick(height, currY);
    tick.setPosition(screen.width, currY);
    var tickID = tick.id;
    var tickW = tick.width;

    //UPDATE BEACHLINE
    updateBeachLine(tick);

    var phase2 = tickW / scrollSpeed;
    var phase3 = (tickW + screen.width)
                      / scrollSpeed;

    //BACKFILL TIMER
    setTimeout( backfillTick(tickID, y),
      phase2);

    //ANIMATION AND REMOVAL TIMER
    newTick.animate({ left: -tickW
        }, phase3, "linear", function() {
          $(this).remove(); });

    currY += tickH;
  }
}
```

Fig. 4. The self-perpetuating backfillTick() function drives the display using the beachLine and the activeTicks pool.

facing of 0 height if its parameter lands on a “split” facing. To remove a split, the id of the corresponding beachLine point needs only to be set to null. The negative x-value then ensures that the facing will be reclaimed during the next beachLine shoring.

Discovery: Our goal was for the Panta Rhei to function in the periphery, enabling serendipitous discovery without requiring interaction. This is accomplished with a patiently-paced animation, a non-intrusive color scheme, and a visual randomness to how the content is packed into the stream. However, the display can also demand the user's attention more actively when necessary. A particular headline, displayed in a unique color, in isolation, or in a rigid grid that defies the typical heterogeneity of the display all succeeded in our initial testing at piquing the user's attention even when they are not paying direct attention to the display. These three tactics are shown in Fig. 5.

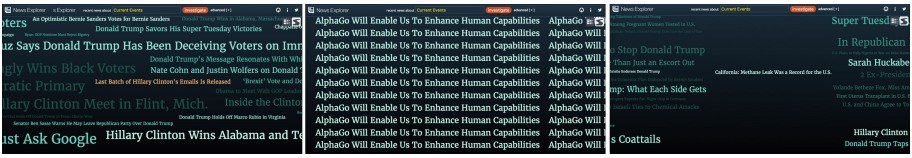

Fig. 5. The Panta Rhei can draw attention to a particular tick using color (top), uniformity (middle), or isolation (bottom). (Color figure online)

5 Conclusion

The Panta Rhei is a reusable, peripheral display that surfaces original content comprised of text, images, or any other element that can be placed within a web-based . It's initial deployment within a corporate communications team, tasked with tracking online media, yielded a surprising array of immediate and actionable insights. "Most of our tools don't bring light to a story until it is trending, so since [this article] was not trending yet we had not noticed it on any other tool in the room," a team member reported of one particular instance when she spotted a headline in which the purchase price of a recent acquisition had been misprinted. The responsible publication was immediately contacted to have the misprint corrected. Going forward, the Panta Rhei will be subjected to more formal testing in order to quantify its ability to generate such insights with more empirical rigor.

References

1. Collins, C., Carpendale, S., Penn, G.: Docuburst: visualizing document content using language structure. In: Proceedings of the Eurographics/IEEE-VGTC Symposium on Visualization (EuroVis), vol. 28, no. 3, pp. 1039–1046 (2009)
2. D3.js - Data Driven Documents. http://d3js.org/. Accessed 23 Mar 2016
3. Eler, D.M., Nakazaki, M.Y., Paulovich, F.V., Santos, D.P., Andery, G.F., Oliveira, M.C.F., Neto, J.B., Minghim, R.: Visual analysis of image collections. Vis. Comput. **25**(10), 923–937 (2009)
4. Fortune, S.: A sweepline algorithm for voronoi diagrams. In: Proceedings of the Second ACM Symposium on Computational Geometry, pp. 313–322 (1986)

5. Jafarinaimi, N., Forlizzi, J., Hurst, A., Zimmerman, J.: Breakaway: an ambient display designed to change human behavior. In: CHI EA '05 CHI '05 Extended Abstracts on Human Factors in Computing Systems, pp. 1945–1948 (2005)
6. Pang, B., Lee, L.: Opinion mining and sentiment analysis. Found. Trends Inf. Retrieval **2**(1–2), 1–135 (2008)
7. Rao, D., McNamee, P., Dredze, M.: Newslab: exploratory broadcast news video analysis. In: IEEE Symposium on Visual Analytics Science and Technology, pp. 123–130 (2007)
8. Rao, D., McNamee, P., Dredze, M.: Entity linking: finding extracted entities in a knowledge base. In: Poibeau, T., Saggion, H., Piskorski, J., Yangarber, R. (eds.) Multi-source, Multilingual Information Extraction and Summarization. Theory and Applications of Natural Language Processing, pp. 93–115. Springer, Heidelberg (2012). doi:10.1007/978-3-642-28569-1_5
9. Rosenman, M.F.: Serendipity and scientific discovery. J. Creative Behav. **22**, 132–138 (1988)
10. Shneiderman, B.: The eyes have it: a task by data type taxonomy for information visualizations. In: IEEE Symposium on Visual Languages, pp. 336–343 (1996)
11. Skog, T., Ljungblad, S., Holmquist, L.E.: Between aesthetics and utility: designing ambient information visualizations. In: Proceedings of the 9th Annual IEEE Conference on Information Visualization, INFOVIS 2003, pp. 233–240 (2003)
12. Tanahashi, Y., Ma, K.L.: Stock lamp: an engagement-versatile visualization design. In: Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, CHI 2015, pp. 595–604 (2015)
13. Viégas, F.B., Wattenberg, M., Feinberg, J.: Tag clouds and the case for vernacular visualization. In: ACM Interactions, vol. XV, no. 4, July/August (2008)
14. Viégas, F.B., Wattenberg, M., Feinberg, J.: Participatory visualization with wordle. IEEE Trans. Vis. Comput. Graph. **15**(6), 1137–1144 (2009)
15. Yatani, K., Novati, M., Trusty, A., Truong, K.N.: Review spotlight: a user interface for summarizing user-generated reviews using adjective-noun word pairs. In: CHI 2011 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 1541–1550 (2011)