Automatic Transitional Animation Between Visualizations

Abstract

We present a generic framework for animating between different visualizations using polygon morphing. The motivation for this work arises from using many different visualization techniques (e.g. scatter plots, parallel coordinates, etc.) to view complex, multi-dimensional data. When transitioning from a visual representation that the user is familiar with to one that is new or one that the user is unfamiliar with, the semantics of the new representation are not always clear. We provide a tool that can be used as a learning aid for a user to use their understanding of one representation to learn a new one. Using the Prefuse toolkit [7] we have implemented a tool that allows developers to easily create two or more representations and animate between them without explicitly describing paths or shapes in the animation.

1. Introduction

With all kinds of data being collected at an ever increasing rate for different applications and domains such as businesses, social media, bioinformatics and health-care, sciences, government and others, there is a need for smarter and more sophisticated visualization techniques to analyze and learn useful insights about these data sets. It is therefore not surprising to see more research published in topics such as novel visual mapping of data. Unfortunately, these new visualizations are not always immediately intuitive for us. There is usually some learning curve when it comes to understanding the visualizations and interpreting the results. The goal of this paper is to provide tools to facilitate the learning process of new visualization techniques – through the use of animation. The central idea is to introduce new visual mappings to users by first showing them visual representations of data using techniques that they are already familiar with. Then, we use animation to transform that visual mapping to the novel visual mapping that they are trying to learn, and vice versa, until the visual mapping of data using the two techniques becomes apparent. For example, if a user is familiar with scatter plots, but not with parallel coordinates [9], an animation showing how one or more scatter plots is transformed into parallel coordinates and back can illustrate the concept behind parallel coordinates. Learning based on something that the users are already familiar with can speed up their understanding of new ideas. This approach is also quite similar to one of the ways we learn new concepts: learn by example. In this context, the task of learning a new concept is replaced with identifying and replacing the variables in the familiar concept with those in the new concept. Using animation to find the connections between the variables will further facilitate the learning process.

In addition, we argue that animating between visualizations alleviates the cognitive task of switching between the semantics of different visualizations. For example, when looking at two visualizations: a familiar display and a novel display, side-by-side, it may be the case that *important* data points emphasized in the first view are not emphasized in the second view and vice versa. This adds the cognitive task of "switching" between the semantics of the new and old visualizations. Animation allows us to explicitly "connect the dots" between the two visualizations by illustrating how the visual representation of one datum moves or transforms to a different visual representation for the same datum.

Since we expect new visualization techniques to be created on a regular basis, we want to be able to create the animations that transforms from one representation into another fairly quickly without too much custom coding, beyond specifications of how data in each representations are mapped visually. So, the goal is to automate the process of transforming from one mapping to another. Using the scatter plot to/from parallel coordinates example, each data point in a scatter plot corresponds to a line segment between two parallel coordinates corresponding to the two axes of the scatter plot. Thus, we need to provide a means of transforming a data point at P(x,y) in a scatter plot illustration to a line segment from P1(x,y) to P2(x,y) in a parallel coordinate illustration. This can be achieved by having both P1 and P2 be co-located at P when they are in the scatter plot display; and have the two end points of the line segment move to their correct positions as the scatter plot display is transformed into a parallel coordinate display. However, what if the user is given a new visualization where each multivariate data point is represented by, say, an ellipse with the first and second axes of the ellipse mapped to different data attributes? Can we transform from the scatter plot to this new view? What about from parallel coordinates? Our animation framework gives a simple solution to such a problem.

2. Related Work

Animation has been used in many different ways within visualization from visualizing the performance of algorithms [4] to an automatic approach for visualizing time varying data [13]. This section will cover research related to using animation as a "learning" aid. We will also briefly cover some of the research in polygon morphing, though our primary contribution is a framework for automatic animation between visualizations. Before we delve into these though, we first describe some similarities and differences between our approach and those provided by brushing and linking [6]. This is an interactive technique used in conjunction with multiple views (usually using different visualization techniques e.g. scatter plots and parallel coordinates, etc.) wherein the user selects (brushes) a portion of the data in one display, and the corresponding (linked) data in the other displays are highlighted. This is an excellent technique for learning about how portions of a data set appear in different displays, and indirectly learn about the visual mapping of the different displays as well. Compared to what we present in this paper, brushing and linking are similar in that both use multiple views, and both are capable of showing relationships between multiple views. They are different in that brushing is typically used on parts of the data. Selecting the entire data set is possible but it would defeat the purpose of learning about relationships among the multiple views. Another aspect where the two are different is that brushing and linking is an interaction technique primarily geared towards understanding the data, while our animation technique is geared primarily towards understanding the novel visual mapping.

2.1 Animation as a Learning Aid

While some research indicates that animation can in fact be detrimental to learning [14], there is also research to suggest the contrary: that animation can be an effective tool used to help convey information that is difficult, if not impossible, to understand with static images [13]. Bartram also explains in her thesis that "Movement is reported as improving the visibility of targets embedded in 'random or cluttered' fields especially away from the center." In other words, animation can be an effective tool in noisy/cluttered displays. This is of particular importance with high dimensional datasets due to their often complex representations. Of course, one cannot forget the ACM distinguished doctoral dissertation of Marc Brown on algorithm animation. It has inspired many of the highly informative and educational videos of various sorting algorithms and other software algorithms.

2.2 Polygon/Mesh Morphing

There are a number of options available for creating animations of visualizations. Yu et al. were able to automate animations for time-varying data [15]. Their approach generates an **event graph** which, combined with story-telling techniques, is used to create animation.

Both VisIt [1] and ParaView [11] use a keyframe system where the user can setup a series of "key frames" and then use various types of interpolation to animate between them. This system can remove a great deal of work from creating an animation; however, as Akiba et al. [2] point out, creating such keyframes can be very difficult, especially if the task is required of someone not familiar with computer animation. Instead, Akiba et al. propose a template based animation system, which they argue is easier to use and "encourages expressiveness."

Our work is different in that the transformations and the resulting animations are derived based on the specifications of data mapping in individual displays. On the other hand, our work is similar in the sense we are aiming for an automatic animation as a learning tool. However, the tools mentioned thus far focus on creating animation from the data exploration process to help understand the *data* being visualized. Our work focuses on helping the user understand the semantics of the *visualization*.

Earlier, we mentioned the simple case of transforming points and line segments. A more general problem is the task of morphing n-sided polygons. Two of the methods we considered for morphing such polygons were the Extended Circular Image (ECI) [10] and mesh parameterization [8].

The method developed by Kamvysselis works by mapping the normals of each side of the source and target shapes onto the unit circle. He then interpolates between normals to compute the intermediate shapes. This technique produces very desirable results, however, it is constrained to convex polygons, and our morphing algorithm must handle both convex and concave polygons. While it is possible to transform concave polygons into a series of convex polygons, we felt this added an unnecessary complication for what we needed.

Mesh parameterization involves parameterizing the surface of the source and target mesh, often using Barycentric coordinates, and then developing a mapping between the two parameterizations. This technique is prevalent in 3D graphics, and if we extend our technique to 3D, this may be the method we use. However, since we are only dealing with 2D polygons in our current work, it was not necessary to use this technique here.

3. Animation of Information Visualization Techniques

To demonstrate animations among different methods of displaying data, we consider the problem of using four different visualizations to understand a high-dimensional dataset. The four techniques used are:

- Multiple scatter plots
- Parallel coordinates
- Comparative column [5]
- Bullseye [5]

Scatter plots are very common and have been in use in various disciplines through the ages. Parallel coordinates is a fairly new invention and is popular in the visualization community, but is not as well known in other disciplines e.g. statistics community. The latter two visualization techniques were introduced just this year and is unfamiliar to most readers at this time. We briefly explain these two techniques before proceeding with discussions on how to animate from one type of display to another.

The comparative column is a technique for visualizing the *difference* between two attributes (or dimensions) with respect to a third attribute. The difference is shown via the x-axis while the third attribute is mapped to the y-axis. For example, when comparing the average price of a gallon of gasoline in different states to the national average, the difference is displayed along the x-axis. If there's no difference, then the data point is located halfway between the left and right borders. If the state's average is higher or lower than the national average, then it is placed either to the left or right of the central line. The vertical axis could be mapped to another variable such as the variability of the price, or the rising/declining trend of the price, etc. We also added color as a redundant color coding for this representation. A point's hue and saturation is determined by its horizontal position while vertical positioning is mapped to brightness. This gives us the following color regions: *left = red, right = green, center = white, top = bright, bottom = black* (see Figure 4).

The bullseye is used to compare several different graphs or attributes in a circular layout. In this case we are using each sector of the bullseye to represent a different attribute. One could think of these sectors as a series of scatter plots lined up side by side and then transformed into polar coordinates. We are visualizing three attributes in all the examples shown in this paper, which we will call a_1, a_2, a_3. For the

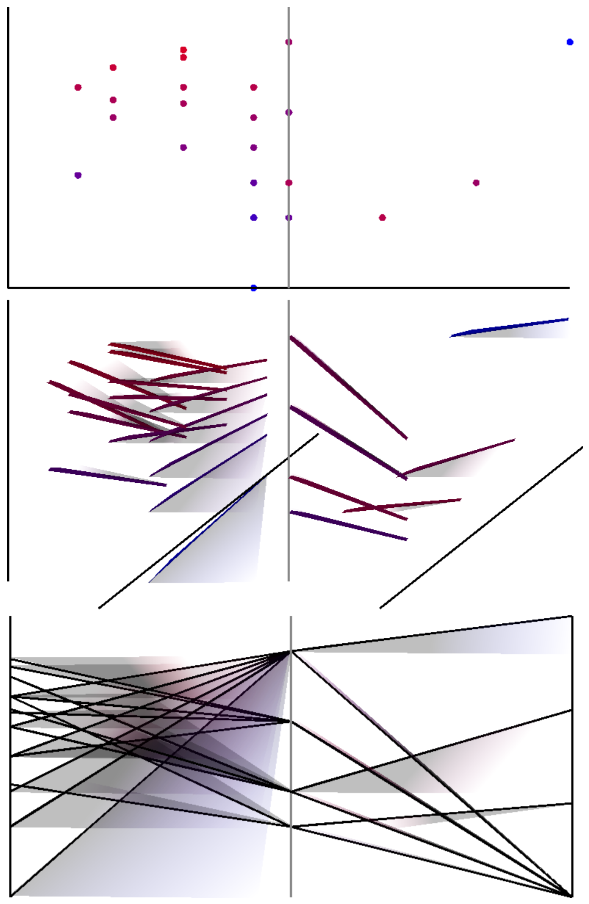

Figure 1. Example of an animation from two scatter plots to parallel coordinates.

scatter plots, the left scatter plot is a_2 vs a_1 and the right scatter plot is a_3 vs a_2. In general if we have n attributes, then the extents of the i^{th} scatter plot from the left would be $s(i) = [iW, (i+1)W] \times [0, H]$ where W and H are the width and height of each scatter plot respectively. We then define the the extent b of the i^{th} sector in the bullseye in polar coordinates as follows: $b(i) = \left[i\frac{2\pi}{n-1}, (i+1)\frac{2\pi}{n-1}\right] \times [0, R]$ where R is the radius of the bullseye and $n > 1$ is assumed. See [5] for more detailed information on how these techniques can be used.

Now that we have introduced the less familiar techniques used, we continue with the animations among them. We first note that if we were to create transitional animations between m visualizations, we would need to manually create $\sum_{i=1}^{m-1} i = \frac{m(m-1)}{2}$ animations, hence requiring 6 animations to be created for the examples shown in this paper. Some of these are shown in Figures 1 to 5. The animations among the

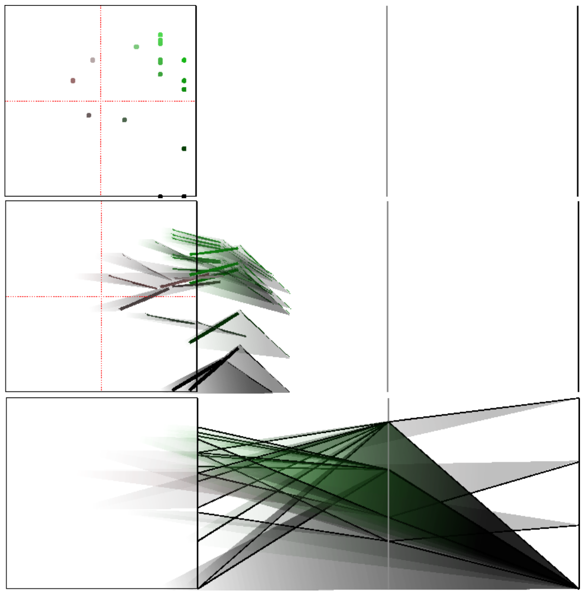
Figure 2. Example of animation from a comparative column display to parallel coordinates. Notice that early on in the animation we can get a sense of the mapping between points and polylines. For example, it is easy to see which type of polyline corresponds to darker (low valued) points.

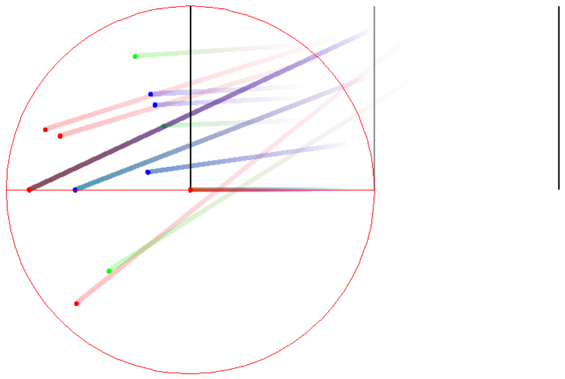
Figure 3. Example of animation from a scatter plot to a bullseye. This frame is near the end of the animation sequence.

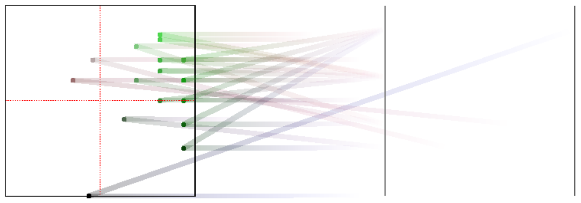
Figure 4. Example of animation from a scatter plot to a comparative column.

different visualization techniques are available from: http://avis.soe.ucsc.edu/images.anim/morphAnims.ogg Also, if we added another new visualization, we will need to manually create m new animations. Even using a keyframe-like system, this would become extremely tedious. A keyframe system would make automatic interpolation between similar shapes possible. For example, animating from one scatter plot to one bullseye sector is simply a matter of interpolating between shape positions and should only require two keyframes: a start and end position. However, what if we want to animate between a scatter plot and parallel coordinates or elliptical glyphs as described in the introduction? Simple keyframes are not a viable option in these situations.

We will now explore some of our examples more closely. Consider the blue dot in the bottom right of the left scatter plot in Figure 1 as well as the blue dot in the top right of the right scatter plot in the same figure. The animation gives the user a sense of the slope of each line associated with these points (please see the associated animation at http://avis.soe.ucsc.edu/images.anim/morphScatterPCoord.ogg).

Figure 2 shows a similar result. Once again it is fairly easy to see the final shape a point will turn into early on in the animation (please see http://avis.soe.ucsc.edu/images.anim/morphCCPCoord.ogg).

Figures 3 and 4 show the final frames of animating from multiple scatter plots to a bullseye and comparative column respectively while Figure 5 shows how the (x, y) coordinates of points map to (θ, r) in the bullseye.

4. Animation Framework

In order to provide an automated means of transforming a data point in one visual representation to another, the key ingredient is the ability to morph between two different representations of the same data point. We achieve this using a general interface described in Figure 7 using an abbreviated Unified Modeling Language (UML) [3] diagram. This represents the core design of our framework while Figure 8 describes the overall concept of the **TransitionRenderer**.

Our design is modeled directly from Figure 8. When a developer wishes to create a new visualization, he or she will create a new realization of the **Representation** interface where a method of the form

getShape(data:Datum):RepresentationItem

must be implemented. The **RepresentationItem** is a sim-

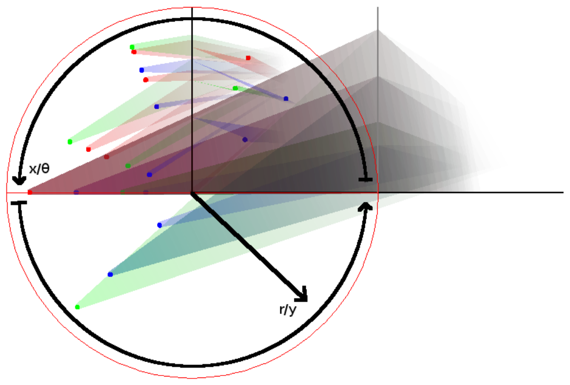

Figure 5. Example of animation from parallel coordinates to a bullseye display. This frame is near the end of the animation sequence. As in Figure 2, it is easy to see the mapping between points and lines. In this case, there is one line segment per point, unlike Figure 2 where one point is mapped to two line segments.

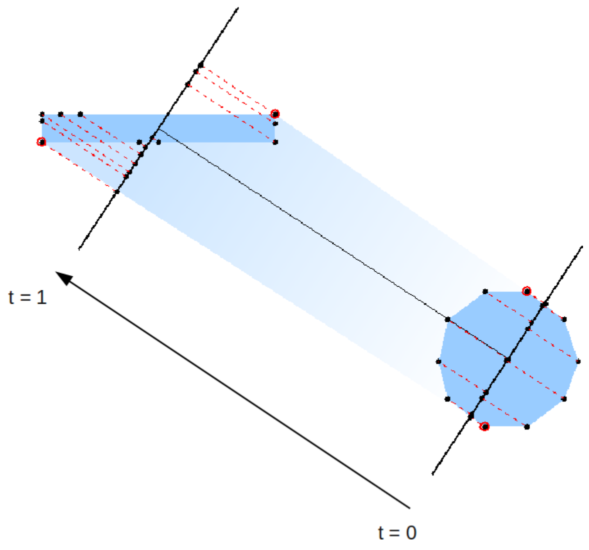

Figure 6. An illustration explaining how the projections are created. Each point circled in red is a point making part of the outline of the transition projection.

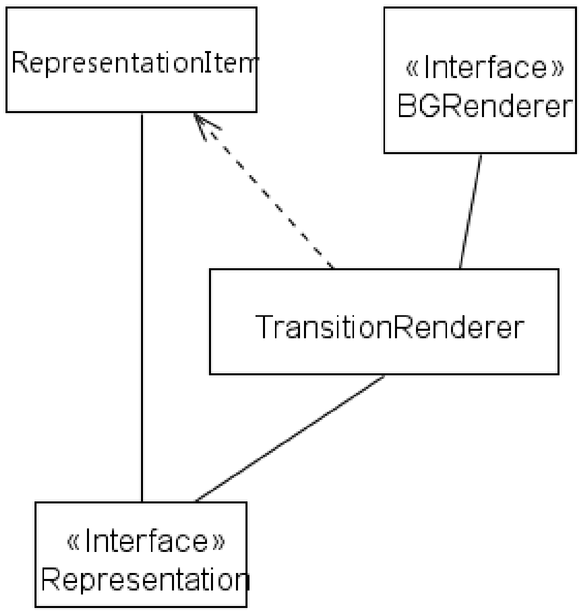

Figure 7. UML overview of the transition framework. The TransitionRenderer acts as the Prefuse renderer [7] and is initialized with one or more Representations. A developer can then animate between Representations by simply setting the *from* and *to* fields in the TransitionRenderer instance and calling the animation (e.g. using the ItemAction class in Prefuse).

ple container which stores an array of polygons, colors, and sizes. For example, the scatter plot and bullseye representations in the examples provided in this paper would return a **RepresentationItem** containing two polygons, each polygon in the shape of a circle. For the parallel coordinates implementation there would be two rectangles, and the comparative column representation would contain a single circle.

Once a developer has all of his or her Representations created, he or she then passes them to the **TransitionRenderer**. Once this step is complete, all that is needed to animate between representations are the following steps (each being one method call to the **TransitionRenderer**):

1. `setFrom(r:Representation)` *sets the source representation*
2. `setTo(r:Representation)` *set the target representation*
3. `runAnimation(t:Double)` *start the animation lasting for* `t` *milliseconds*

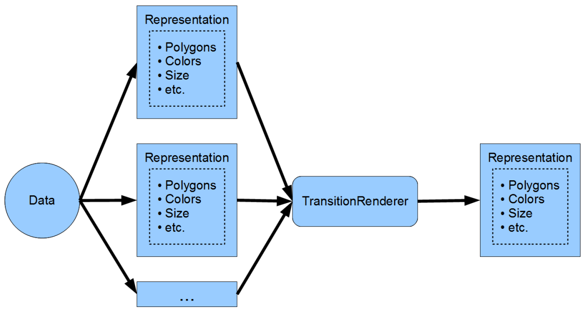

Figure 8. The TransitionRenderer can take in several *Representation*s, but only interpolates between two of them for a single animation.

4.1 Polygon Interpolation

As Figure 7 might suggest, within the **TransitionRenderer** is where the whole of the polygon morphing work is done as well as interpolation between color and any other visual features contained within the **RepresentationItem**. As mentioned earlier, each datum is described as an array of polygons, and our algorithm for interpolating between these two arrays of polygons is as follows:

1. Make all polygons in each array have the same number of points
2. Linearly interpolate between the points of *matching* polygons (see Figure 9)

Matching polygons can be described as follows: let the source and target arrays of polygons be $\texttt{polys}_1$ and $\texttt{polys}_2$ respectively and let $\texttt{polys}_1.\texttt{length} \geq \texttt{polys}_2.\texttt{length}$, then

$$\texttt{polys}_1[i] \textit{ matches } \texttt{polys}_2[min(i, \texttt{polys}_2.\texttt{length}-1)] \tag{1}$$

or

$$\texttt{polys}_1[i] \textit{ matches } \texttt{polys}_2\left[\left\lfloor i\frac{\texttt{polys}_2.\texttt{length}}{\texttt{polys}_1.\texttt{length}}\right\rfloor\right] \tag{2}$$

Either can be used.

There is an optional intermediate step between 1 and 2 which is to ensure that the points of each *matching* polygon line up. That is, to avoid a point on the left side of the source polygon getting interpolated with a point on the right side of the target shape. If this were the case, the result would be a "flipping" or "mirroring" effect of each datum's representation in the animation. We have left this step out for now since such an effect may be desirable.

We will now describe the morphing method more formally. Let the method which does the shape morphing described above be called ϕ. We will also use the following notation:

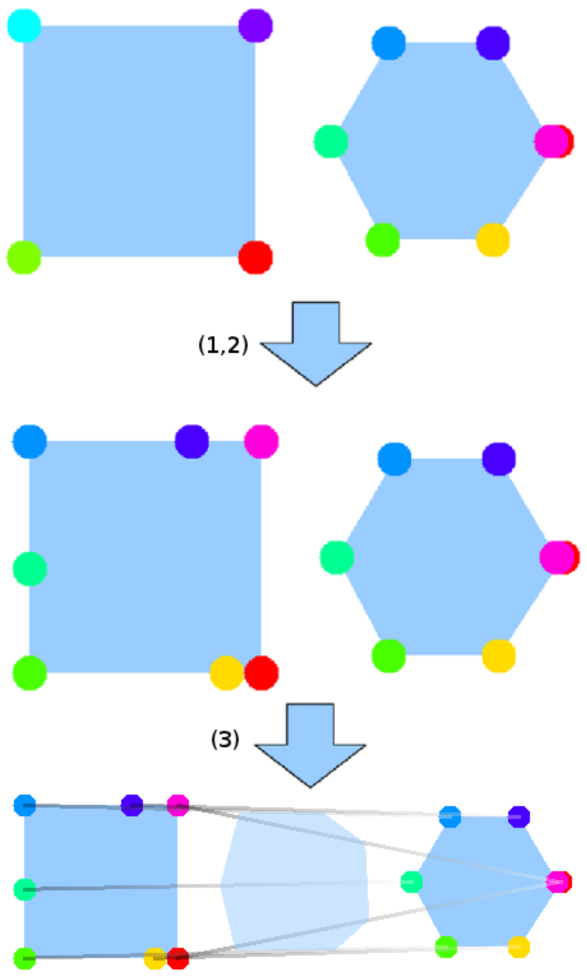

Figure 9. Illustrating how points are added to polygons to make all polygons have an equal number of points.

- $\mathbb{P}^k_{n_1,n_2,\ldots n_k}$ = k polygons where n_i is the number of points in the i^{th} polygon
- $\mathbb{P}^k_n$ = k polygons where each polygon $p \in \mathbb{P}^k_n$ has n points.
- Let $\mathrm{p} : \mathbb{P}^k_{n_1,n_2,\ldots n_k}$, then $\mathrm{p}_i[j] \in \mathbb{R}^2$ = the j^{th} point in the i^{th} polygon of p
- Let $\mathrm{p} : \mathbb{P}^k_{n_1,n_2,\ldots n_k}$, then $|\mathrm{p}_i|$ = the number of points in the i^{th} polygon

$\mathbb{P}^k_{n_1,n_2,\ldots n_k}$ and $\mathbb{P}^k_n$ are synonymous with the array of polygons in a **RepresentationItem**. Then ϕ has the following type:

$$\phi : \mathbb{P}^k_{n_1,n_2,\ldots n_k} \times \mathbb{P}^l_{m_1,m_2,\ldots m_l} \times \mathbb{R} \mapsto \mathbb{P}^k_n$$

where $n = max(n_1, n_2, \ldots, n_k, m_1, m_2, \cdots, m_l)$ and $k \geq l$.

The general process of ϕ in pseudo code is as follows:

```
$\phi$(p1:$\mathbb{P}^k_{n_1,n_2,\ldots n_k}$, p2:$\mathbb{P}^l_{m_1,m_2,\ldots m_l}$, t:$\mathbb{R}$):
    (pHighRes1, pHighRes2) $\leftarrow \gamma(p1, p2)$
    return($\phi'$(pHighRes1, pHighRes2, t))
```

The method γ ensures all polygons have the same number of points (i.e. step one of the algorithm described at the beginning of section 4.1) and has the following type:
$\gamma : \mathbb{P}^k_{n_1,n_2,\ldots n_k} \times \mathbb{P}^l_{m_1,m_2,\ldots m_l} \mapsto \mathbb{P}^k_n \times \mathbb{P}^l_n$

γ does the following given two arrays of polygons:

1. Find the polygon with the most points. Call this polygon M.
2. For each polygon p in each array, insert points into p until $|p| = |M|$ (i.e. p has the same number of points as M).

ϕ' does the actual work of morphing each polygon within the two arrays and has the following type:
$\phi' : \mathbb{P}^k_n \times \mathbb{P}^l_n \times \mathbb{R} \mapsto \mathbb{P}^k_n$
The algorithm is quite simple. Assuming that every polygon has the same number of points, ϕ' linearly interpolates between the points of *matching* polygons (where *matching* is the same as described in equations 1 and 2). In pseudo code:

```
$\phi'\left(\texttt{p1} : \mathbb{P}^k_n, \texttt{p2} : \mathbb{P}^l_n, \texttt{t} : \mathbb{R}\right)$ :
    Let p:$\mathbb{P}^k_n$
    Then
    for i $\leftarrow$ 1 to k
        Let $i' \leftarrow \left\lfloor i\frac{l}{k} \right\rfloor$
        for j $\leftarrow$ 1 to n
            $\texttt{p}_i[j] \leftarrow \texttt{p1}_i[j] + t\,(\texttt{p2}_{i'}[j] - \texttt{p1}_i[j])$
    return p
```

Where $k \geq l$ is assumed.

5. Implementation

For visualizations and animations, we are building off of the Prefuse framework created by Heer et al. [7] using Java. While Prefuse uses Java2D to render, our technique can be applied to any system which uses polygons for rendering, and could (in theory) easily translate to a system such as OpenGL using a polygon triangulation method [12].

Since we are using Java2D as the rendering engine, performance is limited to a relatively small number of items. On a modern desktop, performance starts to lag considerably at around 200 polygons, with 13 points per polygon. This is not necessarily a problem since our aim is to help understand the semantics of new visualizations with respect to old ones, which may be more successful with less clutter on the screen.

6. User Study

We performed a short, informal user study with four participants with the following backgrounds: PhD visualization student, retired high school math teacher, chemist, and the managing editor of a PR firm. We will refer to them as V, M, C, and E respectively. The task for each participant was to match the points in two scatter plots to their respective line segments in parallel coordinates (essentially what is happening in figure 1). They were first given two static images to work with (the start and end of the animation). Then, each participant was asked the following questions before watching the animation:

1. How long did it take to finish?
2. Are you confident in your answers?

After these questions were answered, each participant was asked to watch the animation from scatter plot to parallel coordinates, both forward and reverse. After this was done, they were asked:

1. Is it easier to tell which points correspond to which lines after watching the animation?
2. Would watching the video first have reduced the time it took to match each line? If so, by how much?

V was the only participant with prior knowledge of parallel coordinates. Both V and M took about ten minutes and were both fairly confident in their answers (M more than V). C took about two minutes with moderate confidence. E answered the first two questions only after watching the animation, but took about 7 minutes and was very confident. V felt the animation didn't help very much, but also said it was essentially "stating the obvious." M said that understanding the mapping between each visualization was most important for understanding how parallel coordinates work, but that the animation made this mapping "very clear." E felt that understanding the mapping between the two visualizations was "impossible" without the animation. C was not entirely sure how useful the animation was as C had to replay it several times for it to make sense, and even then C was unsure about a few points.

We realize that this study is quite informal and very preliminary. It does show some promise of our approach. We plan to polish the animations further and run a more formal user study.

7. Conclusion and Future Work

In summary, we have implemented a framework in Java that automatically animates between different displays using polygonal *representations*. With the ever-increasing number of visualizations of complex and high-dimensional data, it is often difficult to jump from the semantics of one visualization to the next. Providing a mechanism to automatically animate between these visualizations drastically reduces the work required of the developer (by a factor of $\frac{m(m-1)}{2}$) as well as reduces the cognitive load of the user.

We believe there is much room for improving the performance of the mesh interpolation algorithm in order to accommodate large-scale visualizations. We are also currently working on a version which uses hardware rendering which should greatly increase performance.

References

[1] VisIt, https://wci.llnl.gov/codes/visit/.

[2] H. Akiba, C. Wang, and K.-L. Ma. Aniviz: A template-based animation tool for volume visualization. *IEEE Computer Graphics and Applications*, 30:61–71, 2010.

[3] G. Booch, I. Jacobson, and J. Rumbaugh. *OMG Unified Modeling Language Specification*, 2000.

[4] M. Brown and R. Sedgewick. Techniques for algorithm animation. *IEEE Software*, 2:28–39, 1985.

[5] N. Cesario, A. Pang, and L. Singh. Visualizing node attribute uncertainty in graphs. In *SPIE Conference on Visualization and Data Analysis (VDA)*, 2011.

[6] S. G. Eick and G. J. Wills. High interaction graphics. *European Journal of Operational Research*, 81(3):445 – 459, 1995.

[7] J. Heer, S. K. Card, and J. A. Landay. prefuse: a toolkit for interactive information visualization. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '05, pages 421–430, New York, NY, USA, 2005. ACM.

[8] K. Hormann, B. Lévy, and A. Sheffer. Mesh parameterization: theory and practice. In *ACM SIGGRAPH 2007 courses*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

[9] A. Inselberg, B. Dimsdale, A. Chatterjee, and C.-K. Hung. Parallel coordinates: survey of recent results. In *Human Vision, Visual Processing, and Digital Display IV*, volume 1913, pages 582–599, Feb. 1993.

[10] M. Kamvysselis. 2d polygon morphing using the extended gaussian image, 1997.

[11] Kitware. *ParaView Guide*. Kitware, Version 3 (February 2008). http://www.paraview.org.

[12] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, 1998. Hardback ISBN: 0521640105; Paperback: ISBN 0521649765.

[13] B. L. Ruth, L. R. Bartram, and S. F. University. Enhancing information visualization with motion, 2001.

[14] B. Tversky, J. B. Morrison, and M. Btrancourt. Animation: can it facilitate? *International Journal of Human-computer Studies / International Journal of Man-machine Studies*, 57:247–262.

[15] L. Yu, A. Lu, W. Ribarsky, and W. Chen. Digital storytelling: Automatic animation for time-varying data visualization. *Computer Graphics Forum (Special Issue of Pacific Graphics 2010)*, 34(5), 2010.