

VISUS

Bachelorarbeit

Animated Scatter Plot Transitions

Vincent Brandt

Course of Study: Medieninformatik
Examiner: Prof. Dr. Daniel Weiskopf
Supervisor: Dipl.-Inf. Nils Rodrigues

Commenced: November 2, 2021
Completed: May 2, 2022

Abstract

In this thesis we present a model for transitions between different 2D scatter plots of the same underlying multivariate data. We examine three types of transitions: linear interpolation, rotation, and spline paths. These are implemented in a TypeScript library for web-based applications. In the library, we also provide user interface components, and utilize them for an example application for interactive use of our model. To evaluate the different transitions, we conducted an online user study with 25 participants. We compared the effectiveness of each method for tracing points or clusters through the animation. Using a questionnaire, we also surveyed the perceived difficulty of the tasks, as well as the appeal of each method. In the study, we found most transitions perform similarly. When tracing points through an animation, users generally preferred flatter transitions similar to linear interpolation and achieved worse performance especially in a 3-stage 3D rotation transition. We also found two view changes to be the longest transition for comfortably tracking points. When tracing clusters through an animation, all transitions were similar, though in questionnaires the same three-stage rotation performed worse again.

Contents

1	Introduction	11
2	Related Work	13
3	Concepts	15
3.1	Definitions	15
3.2	Transition Types	16
3.3	Path Transformations	21
4	Implementation	23
4.1	Abstract Transitions	23
4.2	User Interface	24
4.3	Transition Types	27
4.4	API Usage Example	29
5	User Study	33
5.1	Procedure	34
5.2	Results	38
5.3	Discussion	46
6	Conclusion and Outlook	47
	Bibliography	49

List of Figures

2.1	The grand tour finds a continuous sequence of planes in the n-dimensional space occupied by the dataset. Points are then projected onto the planes. Illustrated with 3 dimensions.	13
2.2	Illustration from <i>Rolling the Dice</i> [EDF08].	14
3.1	A path through a scatter plot matrix.	16
3.2	Frames from a straight transition. This transition is exchanging both dimensions.	16
3.3	A ROTPERSP transition performs a rigid body rotation in 3D space while also smoothly transitioning between orthographic and perspective projection.	17
3.4	A ROTSTAGED transition performs a rigid body rotation in 3D space with separate expansion and compression stages before and after the rotation.	17
3.5	Reprojection into 2D is required to swap out the depth dimension in multi-step transitions. Example using the ROTSTAGED transition.	18
3.6	Animation paths. The SPLINE transition preserves trajectories in intermediate views. The SPLINEBUNDLED transition additionally collects all points of a cluster into a point, between views.	18
3.7	SPLINETIMEOFFSET spreads the animations of every cluster across time.	19
3.8	Loose intermediaries will only approximate any intermediary views, making use of higher-order Bézier curves.	19
3.9	Example construction of guides.	20
3.10	Example construction of regular spline paths and bundled spline paths.	20
3.11	Comparison of methods for obtaining points on a cubic Bézier curve.	21
3.12	Examples for different types of path transformations.	22
3.13	Example Manhattan-ification possibilities of a the path. Path A avoids the center diagonal, while path B would pass through a highly ambiguous intermediary view with a lot of points overlapping each other.	22
4.1	User interface components for adjusting transition parameters.	24
4.2	A test application with all UI components integrated.	25
4.3	The transition path builder mode. The grid is highlighted in red and the user can click anywhere in the matrix to extend the path. A path transformation is selected in the controls below, so the transformed path is shown in gray. The blue outline indicates the current view.	26
4.4	User interface for configuring the transition builder. In the top row, users can choose the transition type and path transformation. Below are the parameters for that particular transition type.	26
4.5	The FileIo component presents a configuration dialog when importing CSV datasets. Dimensions may be read or ignored. Non-numeric dimensions are always ignored.	27

4.6	Circle segment used to interpolate the perspective value in the perspective rotation transition.	28
5.1	Users are prompted to follow a point through the transition. After the transition, users can click and drag to place a red marker.	34
5.2	Possible outcomes when tracking a cluster.	35
5.3	Users are prompted to follow a cluster through the transition. After the transition, users are asked select the new state of the cluster.	36
5.4	In the third section, users are presented with a transition and a questionnaire. . . .	37
5.5	Age distribution of study participants.	38
5.6	User familiarity with scatter plots and how frequently they work with scatter plots. Scores range from 1 (Strongly Disagree), 2 (Disagree), 3 (Neutral), 4 (Agree), to 5 (Strongly Agree).	39
5.7	User error distances. The left plot shows the euclidean distance, while the right plot shows the minimum of the X and Y distance. Since the transitions were swapping one dimension at a time, we can use the right plot to estimate how accurate users were one transition before the end.	39
5.8	The horizontal axis shows euclidean user error distances. The vertical axis shows the maximal density of points in the vicinity of the selected point over all views of the transition.	40
5.9	Number of participants who answered correctly for each type of cluster transition.	41
5.10	Euclidean user error distances for different transition lengths. Correlation lines drawn in blue.	42
5.11	Distribution of transition lengths at each staircasing step (combined data from all transition types).	42
5.12	Questionnaire answers for point tracking. Scores range from 1 (Strongly Disagree), 2 (Disagree), 3 (Neutral), 4 (Agree), to 5 (Strongly Agree).	43
5.13	Questionnaire answers for cluster tracking. Scores range from 1 (Strongly Disagree), 2 (Disagree), 3 (Neutral), 4 (Agree), to 5 (Strongly Agree).	44
5.14	Questionnaire answers from the third section of the study. Scores range from 1 (Strongly Disagree), 2 (Disagree), 3 (Neutral), 4 (Agree), to 5 (Strongly Agree). .	45

List of Listings

4.1	Definition of an example dataset.	29
4.2	Creation of a dimension descriptor.	29
4.3	Inference of dimension domains from data.	30
4.4	Creation of an SVG container.	30
4.5	Creation and configuration of a Scatterplot.	30
4.6	The Scatterplot is used to create points.	30
4.7	The Scatterplot transition builder interface.	31
4.8	A transition played using d3-transition.	31
4.9	A transition played using the Scatterplot animation timeline.	31

1 Introduction

Scatter plots are a popular method for visualization of two- or sometimes three-dimensional data due to their simplicity and relatively intuitive nature. However, multidimensional datasets may often feature more independent dimensions than can be visualized with scatter plots. Hence, they are often arranged into a scatter plot matrix that features every combination of dimensions in the dataset and the corresponding scatter plot.

A difficulty that arises from using multiple scatter plots in a matrix to visualize a single dataset is that data points are difficult to follow across the different views of the data. A simple solution may be the addition of identifying features such as color or shape. However, this approach only works for small data sets. With a rising number of data points, it becomes increasingly difficult to find a set of distinguishable visual identifiers. We present another approach, in which we use animation to create continuity between each of the scatter plot views.

Animation is particularly appealing, as motion naturally allows these relationships between data views to form, and preserves the mental map to a greater degree. Usage of identifying features, on the other hand, requires the viewer to create this link between the visualizations themselves.

By moving the individual data points from their locations in one scatter plot to their locations in another scatter plot, multidimensional data can be visualized over time in a 2D chart. However, this transition is subject to ambiguity from overlapping points. Due to the inherent complexity in the animation, it may be rather unclear which points moved where. Hence, it may be of interest to use a transition that tries to minimize this ambiguity as much as possible, while still preserving the ability to follow points and clusters across the dimensions.

Following previous work, we compare several different ways of accomplishing this transition: linear interpolation, 3D rotations, and spline-based transitions. These are combined with various methods for charting paths through a scatter plot matrix to create an animation from one scatter plot view to another. We implemented these methods in a Javascript library `d3-scattertrans` for use on the web.

In an online user study, we evaluated their effectiveness and general appeal for accurately tracking points and clusters. We found that there are minor but significant differences. When tracing points through an animation, users had a median error of 13% of the size of a scatter plot for its final location, except for a specific type of three-stage 3D rotation transition, which performed significantly worse with a median error of 16%. All transition types performed similarly for clusters. Questionnaire answers revealed that users generally preferred simpler transitions similar to linear interpolation. In explorative analysis, we also found that accuracy when tracking points tended to decrease after two transitions.

We conclude with a summary of our work and an outlook on future topics of interest.

2 Related Work

Numerous visualization techniques have tackled multidimensional datasets. The scatter plot matrix, or its generalization as the pairs plot [EGS+13], visualizes the data in a two-dimensional matrix of cells that each contain a plot of two of the dimensions in relation to one another. Each cell uses some type of ordinary two-dimensional plot, such as scatter plots in the case of the scatter plot matrix. On the other hand, the parallel coordinates [Ins85] method deals with dimensionality by using a more abstract representation of data points as lines that span multiple parallel coordinate axes. Recent extensions to this method, such as clusterflow parallel coordinates [RSLW20] also optimize this visualization for better recognition of clusters, such as by separating each cluster into its own area. On the concept of animated scatter plot transitions, parallel coordinates are actually remarkably similar in that each coordinate axis can be thought of as a one-dimensional scatter plot, and the parallel coordinates simply a static representation of an animation.

Animation in visualization is also not a new concept and is already an extensively documented topic. A previous taxonomy of transition types in data visualization [HR07] introduces several terms for describing such transitions. Of the types outlined, the most applicable transition type for multidimensional scatter plots is the substrate transformation, i.e. a change in how the data is mapped to the screen. We also highlight the notion of *staging* transitions, i.e. subdividing a transition into multiple different stages. This was found to improve performance in the accompanying experiments in the paper.

Several concrete methods for animation to visualize multidimensional scatter plots have already been proposed in previous work. Usually, the type of scatter plot that is animated is a two-dimensional plot, but methods for three-dimensional scatter plots also exist.

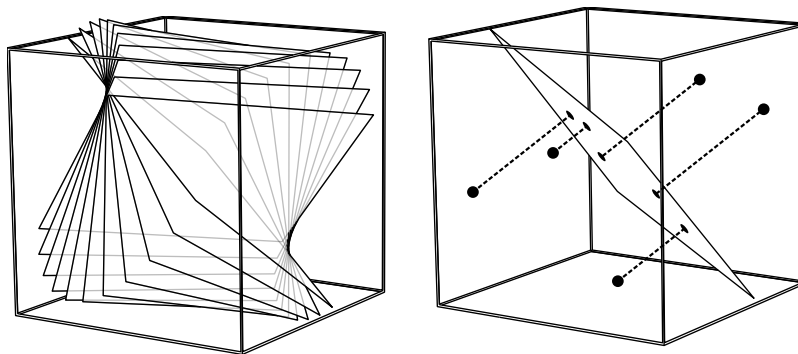


Figure 2.1: The grand tour finds a continuous sequence of planes in the n -dimensional space occupied by the dataset. Points are then projected onto the planes. Illustrated with 3 dimensions.

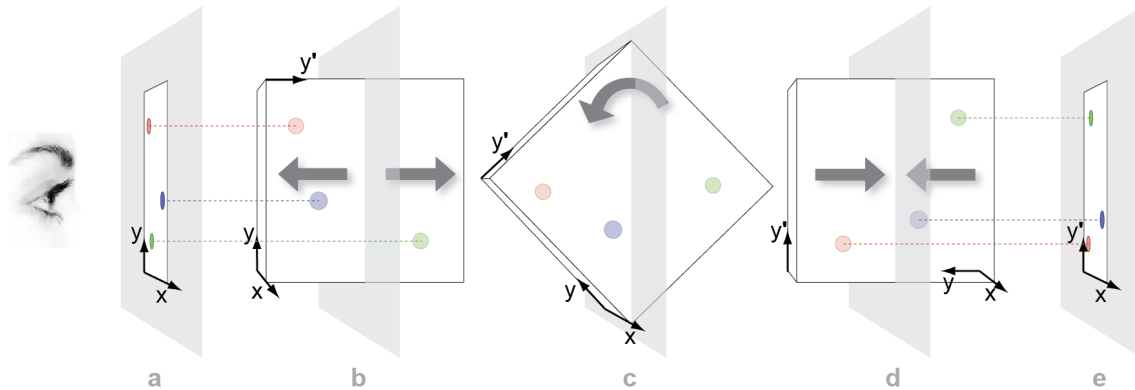


Figure 2.2: Illustration from *Rolling the Dice* [EDF08].

Few methods attempt to make use of the full n -dimensional space in which the points are located. One such method is the *grand tour* [Asi85]. It finds a continuous sequence of 2D planes inside the n -dimensional space visiting every dimension (Figure 2.1). From this, an animation is created by projecting data points onto each of these planes. The planes serve as the view the user sees.

More methods focus on a kind of scatter plot matrix and transitions between the included scatter plots. These transitions swap one or sometimes both of the dimensions used to map the data to the screen.

Rolling the Dice [EDF08] and *RnavGraph* [WO11] demonstrate methods for 2D scatter plots based on 3D perspective rotation. Another similar work, *3D Scatterplot Navigation* [SW12], applies a similar rotation principle, but on 3D scatter plots. In *Rolling the Dice*, to transition to a different scatter plot, a three-stage animation first adds a third depth dimension in a transition to a perspective projection, then rotates the resulting 3D scatter plot to exchange one of the axes for the new depth axis, and finally returns to two dimensions. This transition requires that the previous and next scatter plots always share a dimension (Figure 2.2). We include this transition in our tool under the rotation transition. In *RnavGraph*, the animation contains only the higher-dimensional rotation, without any change to perspective. By remaining in the orthographic view, this removes the requirement for scatter plots to always share a dimension: rotations can be performed even beyond three dimensions, in which case it uses a technique more reminiscent of the previously mentioned grand tour. We do not make use of such higher-dimensional space, as it is more difficult to understand. *3D Scatterplot Navigation* proposes a similar technique. However, instead of using a perspective projection stage, it uses an orthographic projection in a specific angle that allows the scatter plot to visually rotate to swap up to two dimensions.

For navigation, *Rolling the Dice* and *3D Scatterplot Navigation* both use a traditional scatter plot matrix, respectively in two and three dimensions. *RnavGraph* abstracts this for the purpose of animated transitions into a navigational graph that contains only dimension pairs as nodes that can be transitioned between. However, this requires preparation of the dataset so that the navigational graph is not overwhelmingly large. In this work, we opted to use an ordinary 2D scatter plot matrix.

3 Concepts

In the following section we introduce a basic framework for modeling animated transitions between different 2D scatter plots of a single underlying dataset. All examples use the iris [Fis36] dataset unless stated otherwise.

3.1 Definitions

First, the multidimensional dataset is associated with a set of *dimension descriptors*. Each dimension descriptor is associated with a numerical dimension of the data. It defines the upper and lower bounds that will be shown, and a mapping function (such as linear or logarithmic mapping). This allows dimension descriptors to map data points from the dataset to a normalized value from 0 to 1, where 0 corresponds to a data point located at the lower bound of that dimension, and 1 corresponds to the upper bound.

These dimension descriptors may then be combined into *views* of the data. A view contains two dimension descriptors that each correspond to its X and Y axes. Views are then able to map data points to a normalized view space $[0, 1]^2$, and can be used to draw a scatter plot.

Then, to transition between these scatter plots, we will design all animations to exist in the normalized view space. This way, they are not affected by the scale or semantics of the data. Though, it should be noted that the normalized view space does not account for the aspect ratio of the final plot on the screen, and as such the animation works best in square plots.

For the purposes of this thesis, animations can exclusively affect the locations of data points on the screen, and cannot create points, destroy points, or modify their appearance.

When animating between scatter plots, we would like to include scenarios where the animation does not only transition between two views of the data, but is able to consider the entire set of views that will be visited. Hence, we introduce the notion of a transition *path* that visits all views in the transition in order. It is most easily thought of as a path through the scatter plot matrix (Figure 3.1).

Let transitions with paths longer than two views be “multi-step transitions,” and any view that is not the first or last view on this path be an “intermediary view.”

Thus, scatter plot transitions are modeled as a function of the dataset, transition path, data point, and current time that outputs a location in normalized view space. To actually be usable as a transition, the function should also be continuous in time, i.e. points should not teleport during the transition. Additionally, it should output point locations identical to the starting scatter plot at the beginning of the transition, and similarly output the ending scatter plot at the end of the transition.

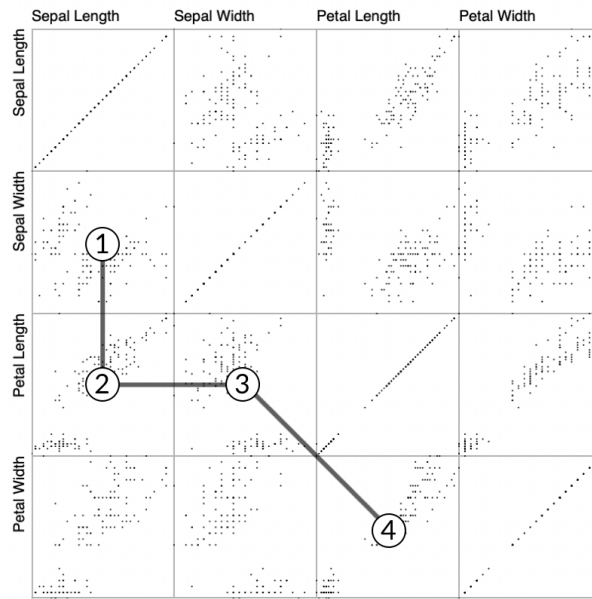


Figure 3.1: A path through a scatter plot matrix.

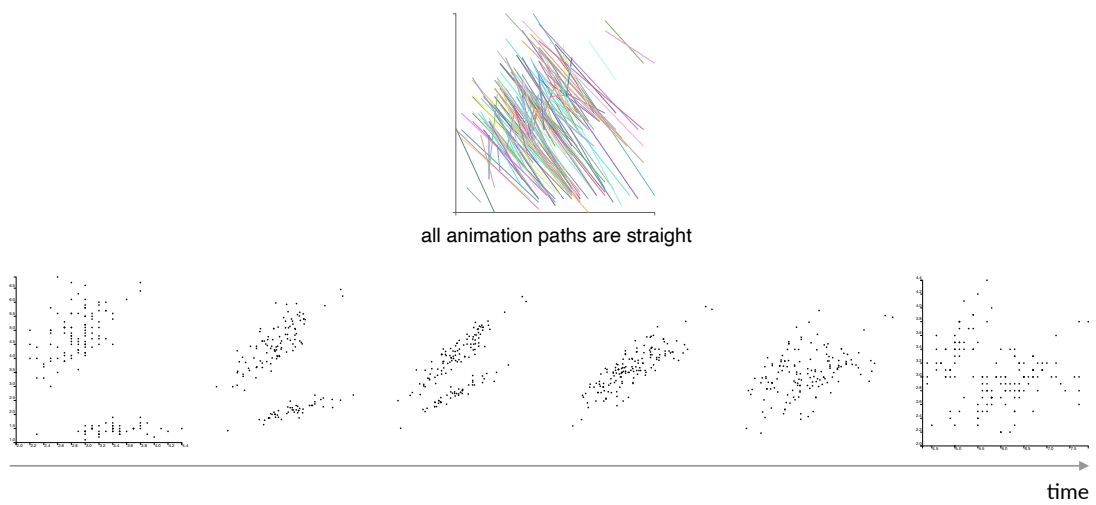


Figure 3.2: Frames from a straight transition. This transition is exchanging both dimensions.

3.2 Transition Types

Having defined this abstract model of transitions, we propose three specific types of transitions.

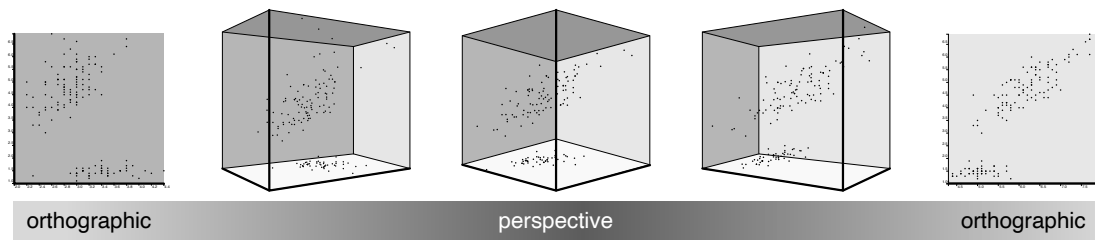


Figure 3.3: A ROTPERSP transition performs a rigid body rotation in 3D space while also smoothly transitioning between orthographic and perspective projection.

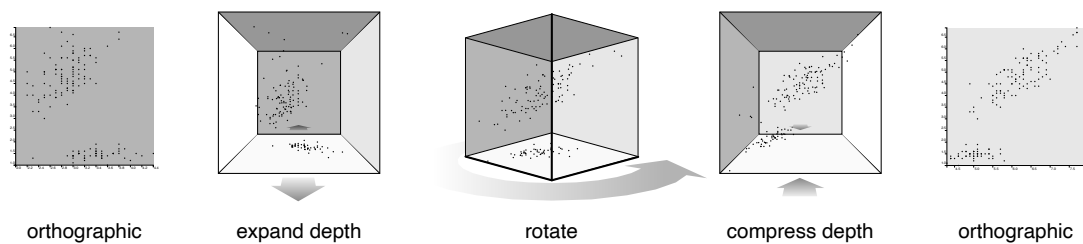


Figure 3.4: A ROTSTAGED transition performs a rigid body rotation in 3D space with separate expansion and compression stages before and after the rotation.

3.2.1 Straight

The “straight” transition (STRAIGHT) type performs simple linear interpolation of every point in the original scatter plot to the corresponding points in the new scatter plot (see Figure 3.2). For longer multi-step transitions, this process is simply repeated for each pair. This transition is not particularly interesting, but serves as a bare minimum for how transitions between scatter plots may work.

3.2.2 Rotation

The “rotation” transition type uses a method similar to that of Elmqvist et al. [EDF08]. The transition is based on adding a third, new dimension to the scatter plot in the depth axis. The scatter plot is then rotated in three-dimensional space on the X or Y axis such that the new third dimension replaces one of the previous two dimensions. As a result, it requires the two involved scatter plots to share exactly one dimension. The rotation is performed on the entire scatter plot as one rigid body.

This method allows for some variation in how exactly the rotation is accomplished. The projection method may range from an orthographic view (“orthographic rotation”, ROTORTHO) to a perspective view: in the case of the perspective view, the initially orthographic 2D projection of the scatter plot must first be pulled apart into three dimensions before the animation can occur. This may be accomplished either simultaneously with the animation (“perspective rotation,” ROTPERSP, Figure 3.3), or in separate stages before and after the rotation (“staged rotation,” ROTSTAGED, Figure 3.4).

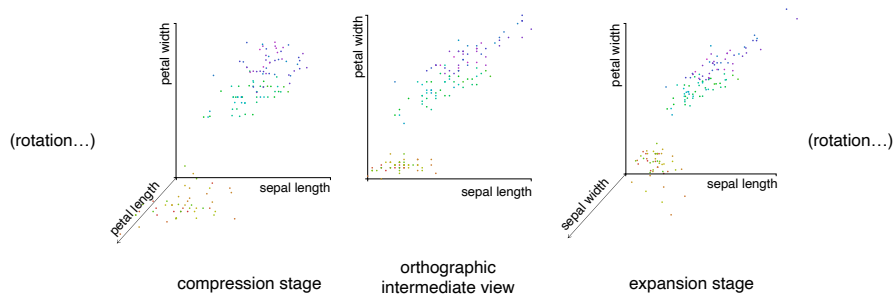


Figure 3.5: Reprojection into 2D is required to swap out the depth dimension in multi-step transitions. Example using the ROTSTAGED transition.

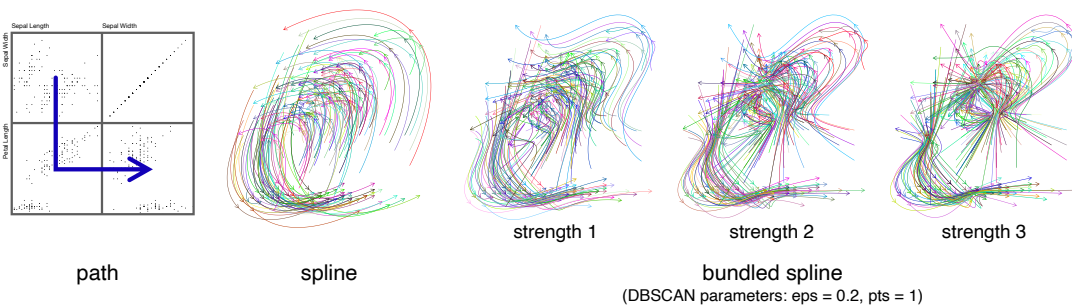


Figure 3.6: Animation paths. The SPLINE transition preserves trajectories in intermediate views. The SPLINEBUNDLED transition additionally collects all points of a cluster into a point, between views.

It should be noted that for multi-step transitions, this method does not allow for the scatter plot to remain three-dimensional. The reprojection into orthographic 2D is required for the depth dimension to be swapped out (Figure 3.5).

3.2.3 Spline

The “spline” transition type is based on moving data points on spline curves, and attempts to explicitly incorporate information about clusters in the underlying data into the transition.

In its most basic form (SPLINE), this transition method preserves the points’ direction of motion in intermediary views (regular “spline transition”). By moving in the same direction before and after a view, we make use of the continuity principle to distinguish points—especially overlapping ones. It should be noted that short transitions with only two scatter plot views have no intermediary views, so this method becomes equivalent to the straight transition type.

By separating data points into clusters using additional parameters in the transition, further adjustments can be made to the animation. Using the clustering information, the trajectories of all points in a cluster may be aligned to be parallel at intermediary views (SPLINECLUSTERED). As a result, motion of points in a cluster becomes more uniform. This is, however, a relatively subtle effect. For stronger identification of clusters at the expense of clarity of individual points, clusters can be made to bundle together (“bundled spline transition,” SPLINEBUNDLED, Figure 3.6). During

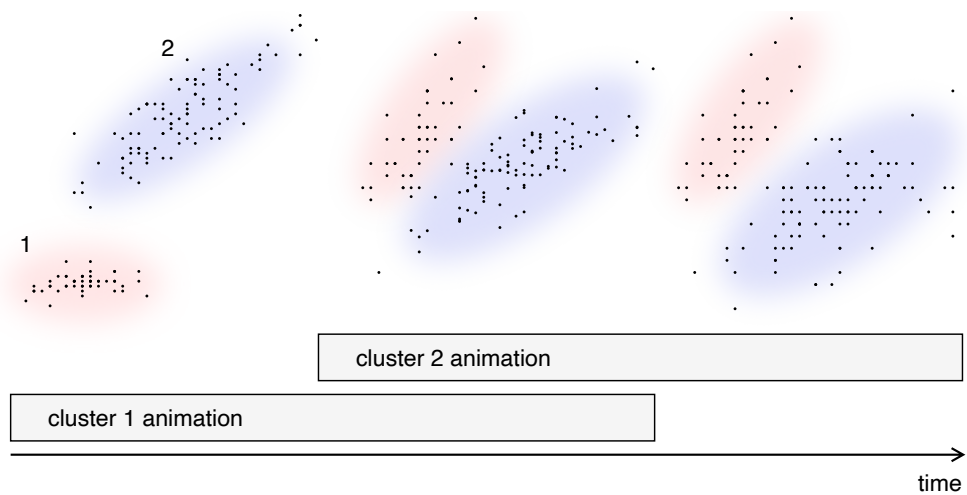


Figure 3.7: `SPLINETIMEOFFSET` spreads the animations of every cluster across time.

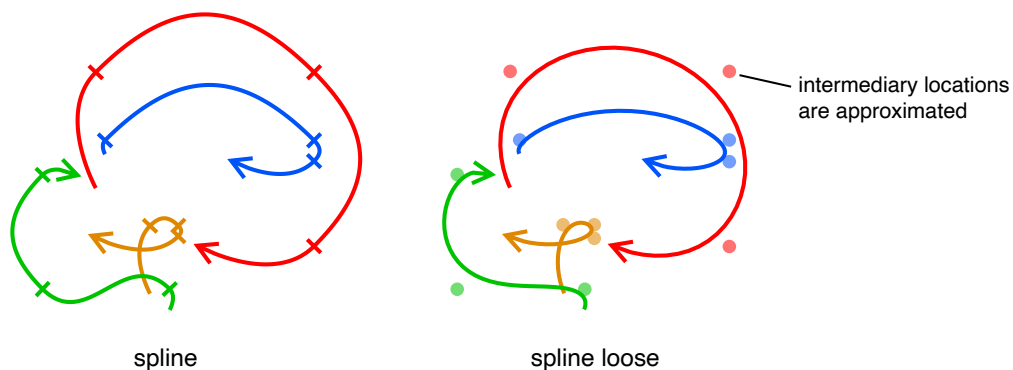


Figure 3.8: Loose intermediaries will only approximate any intermediary views, making use of higher-order Bézier curves.

a transition, this causes all points in a cluster to converge towards a single point before spreading out again. Different clusters are thus momentarily separated into these bundling points, and easier to distinguish. Another alternative mode staggers the animation of each cluster (“time-offset spline transition,” `SPLINETIMEOFFSET`, Figure 3.7). In this mode, each cluster departs and arrives from its starting and ending location at a different time.

Finally, spline transitions may also be used in a mode where the paths will merely approximate each intermediary view instead of exactly passing through them (“loose intermediaries,” `SPLINELOOSE`, Figure 3.8).

Bézier Path Construction

In this transition method, points move on spline curves (“point paths”). Since this method makes use of clustering, point paths are interrelated. They must hence be computed before the animation and cannot be created on the fly. To create the curves, we make use of Bézier splines.

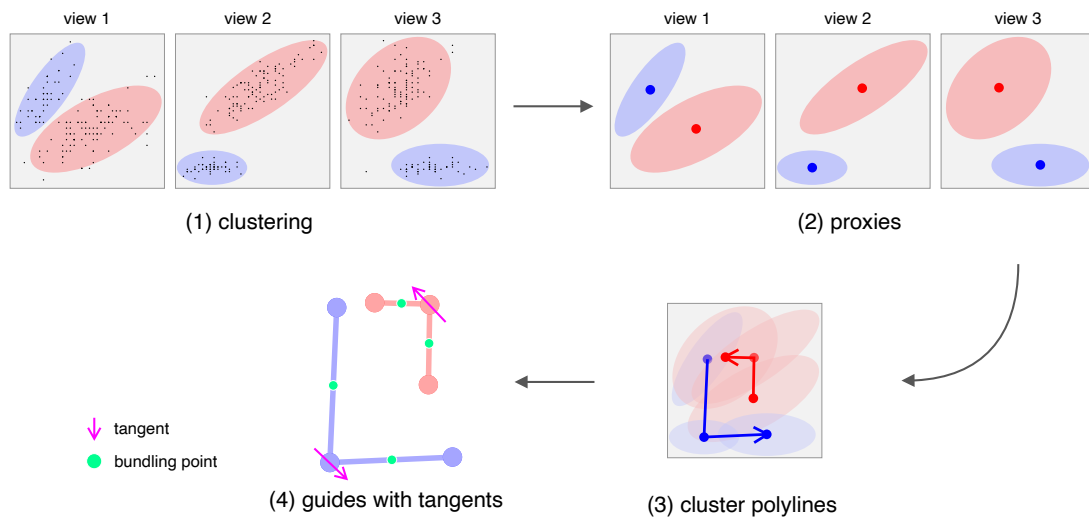


Figure 3.9: Example construction of guides.

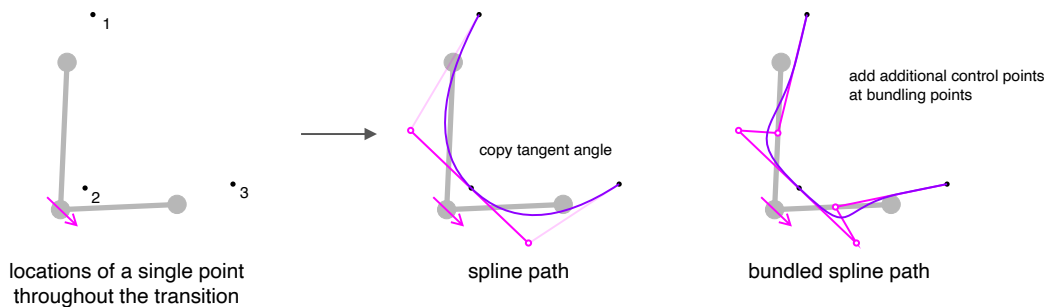


Figure 3.10: Example construction of regular spline paths and bundled spline paths.

First, points are partitioned into clusters using the DBSCAN [EK SX96] algorithm (Figure 3.9, step 1). These clusters are used as proxies for the actual points they represent (step 2). Alternatively, if clustering is not enabled, we will simply consider each point to be its own cluster.

Then, for each view of the transition, the clusters trace their paths as polylines (step 3). We create guides from these polylines by adding a tangent vector to every intermediary vertex and adding a “bundling point” to the middle of every edge (step 4). These guides will be used to derive the spline paths of every point within a cluster.

The way point paths are created from the guides differs depending on the features enabled. In the SPLINE transition, point paths are created by connecting each location of the point using a Bézier curve, with its control points laid out according to the guiding tangent vectors and an arbitrarily chosen length (we used half of the distance between points).

If bundling is enabled, additional Bézier control points are added at the locations of the bundling points on the guide edges. This causes the curves to bend towards that point (Figure 3.10). Bundling strength can be increased by adding more overlapping control points at the bundling points.

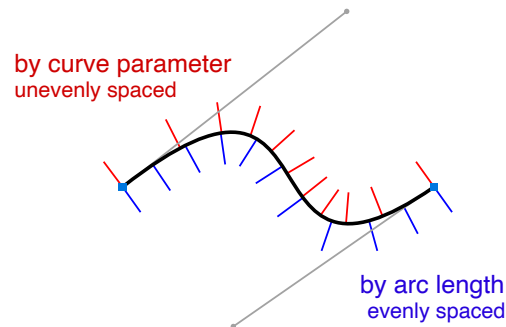


Figure 3.11: Comparison of methods for obtaining points on a cubic Bézier curve.

In the `SPLINELOOSE` mode, locations of intermediary views are not used as end points for the Bézier splines, but instead as control points for a single long curve. This smooths the paths, but they no longer pass through the point positions of intermediary views. Hence, we use this mode as a single, continuous animation, regardless of how many views are in the transition.

Note that these Bézier curves cannot be used for animation directly. Using the parameter of De Casteljau’s algorithm (often called t) as the current time will animate points unevenly and with inconsistent speed. Instead of moving points along the curve evenly with respect to t , points must be moved evenly with respect to the *arc length* of the curve to achieve even motion (Figure 3.11).

3.3 Path Transformations

When creating an animation between scatter plots, it may not be desirable to directly transition between them. Instead, a longer transition path may be useful. To ease tracking of points or clusters, a simple adjustment may be to split up transitions such that only one dimension changes at a time. In a scatter plot matrix, such transitions usually find geometric mappings. For example, changing one dimension at a time corresponds to visual paths that are composed of horizontal and vertical line segments exclusively. While the user could create this kind of transition path in the scatter plot matrix manually, we propose a system by which such paths are created automatically from user input. Input transition paths are specified by the user in the scatter plot matrix, and are then applied a geometric *path transformation* to achieve the desired path.

The simplest path transformation is the identity transformation, which just returns its input. Aside from that, we propose four path transformations (Figure 3.12).

The “Manhattan” transformation converts any diagonal path components to two path components in cardinal directions, resulting in a path the length of the Manhattan distance of the original. It also attempts not to cross the diagonal of the scatter plot matrix. This prevents transitions that remain on one side of the diagonal from passing through the highly ambiguous diagonal cells that just appear as a diagonal line (Figure 3.13). This transformation is also useful for rotation transitions, which cannot exchange both dimensions and so cannot be used with paths that have diagonal components. Another path transformation (“stairs”) performs a similar operation, but adds the further restriction that it must only move one space at a time, and must always move in a different direction. Finally,

3 Concepts

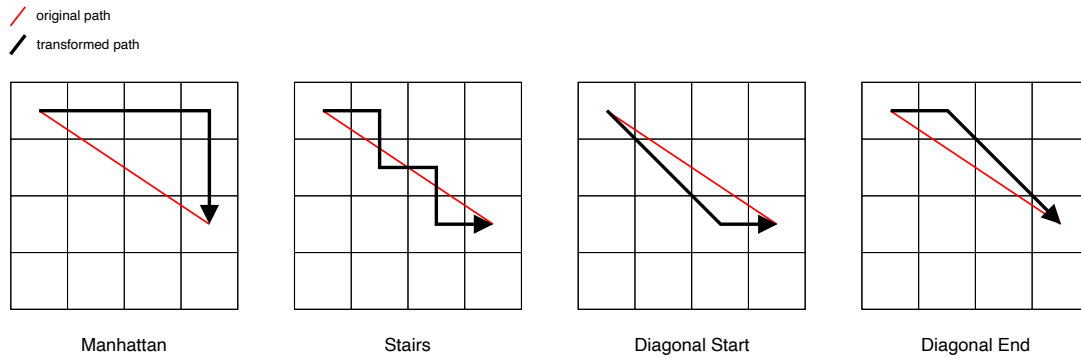


Figure 3.12: Examples for different types of path transformations.

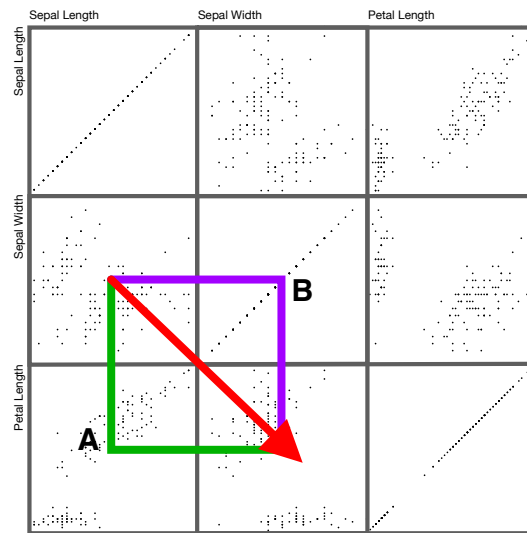


Figure 3.13: Example Manhattan-ification possibilities of a the path. Path A avoids the center diagonal, while path B would pass through a highly ambiguous intermediary view with a lot of points overlapping each other.

a pair of transformations (“diagonal start/end”) strictly move in cardinal-direction lines and 45° diagonals. The diagonal part may be placed before or after the cardinal component, creating the two variants.

4 Implementation

The `d3-scattertrans` library implements the concepts outlined in the previous section as a TypeScript library for web-based visualization. It provides 2D scatter plot transitions in a single Javascript file, used in conjunction with the popular `d3` visualization framework [Bos21].

The conceptual model exists analogously in the library implementation: dimension descriptors, views, transition paths, transition types, and transitions all exist as Javascript classes or objects. However, there are a few more details required that were left unspecified.

4.1 Abstract Transitions

The implementation makes a few additions to the conceptual model to support not only a number of preset transition types from in the library but to also allow it to be easily extensible.

4.1.1 General Transition Properties

To allow easy control over any animation, the animation timeline is always assumed to range over $[0, 1]$, regardless of how many views are in the transition.

However, users may still want to know more about the transition apart from its beginning and end. Thus, transitions have a `hasMeaningfulIntermediaries` property, which indicates whether pausing a multi-step transition at a certain point in time would create exactly the image of an intermediary view. The point in time at which such an image of an intermediary view appears is always assumed to be the same. In a multi-step transition, the animation timeline range of $[0, 1]$ is evenly divided into transitions between each view, so the time at which an intermediary view would be visible is exactly $\frac{i}{N-1}$, where $i \in \{0, \dots, N-1\}$ is the index of the intermediary view and N is the number of views in the transition. While most transitions discussed in this thesis fulfill this property, one type of transition that does not is `SPLINELOOSE`.

When creating transition paths, some transitions may also warrant additional restrictions. To handle this in a generic way, transition types have two additional properties. The `requiresCommonDimensions` property restricts transition paths such that any transition must feature at least one dimension in common. The `canSwapDimensions` property restricts transition paths such that swaps cannot be performed, i.e. transitioning from an (X, Y) view to a (Y, X) view. For example, the rotation transition requires the two views it is transitioning between to have a dimension in common, and it cannot perform a swap.

Parameter Type	Properties	User Interface
bool	default value	<input checked="" type="checkbox"/> Label
number	default value, bounds, whether to round to an integer	<div style="border: 1px solid gray; padding: 2px;">Label: 4.0</div> drag to adjust, click to enter value
enum	default value, possible choices	<div style="display: flex; gap: 10px;"> <div style="border: 1px solid gray; padding: 2px;">Choice 1 ⌵</div> <div style="border: 1px solid gray; padding: 2px;"><input checked="" type="checkbox"/> Choice 1 Choice 2</div> </div> click to expand
group	list of contained parameters, whether the group is nullable	<input checked="" type="checkbox"/> Group Label (group parameters here)
derived	derivation function	(none)

Figure 4.1: User interface components for adjusting transition parameters.

4.1.2 Transition Parameters

Another detail of interest is the configuration of transition parameters to create the different variations of transitions (e.g. `ROTORTHO` and `ROTPERSP`) and fine-tune their details. User interfaces may want to expose these transition parameters for configuration by the user. For this use case, we use a generic interface for declaring transition parameters. These exist declaratively on transition type definitions.

Figure 4.1 shows the different kinds of values supported as parameters. In addition to primitive types such as numbers or booleans, parameters may be derived to create a hidden parameter whose value is computed from other parameters, and they may also be a group of parameters that can be easily disabled or enabled. A `shouldShow` property also allows parameters to be hidden until a certain condition is met in other parameters' values.

For transitions that require precomputed metadata based on the dataset, a dataset parameter may also be passed during creation.

4.2 User Interface

Using this basic framework, we provide user interface components for loading, displaying, and controlling scatter plot transitions (Figure 4.2).

The `Scatterplot` class draws a scatter plot on the screen by mapping the normalized coordinates to pixel coordinates, and optionally to SVG elements. It also provides a method to use the `d3-transition` component of the `d3` framework to render a scatter plot transition. This allows integration with other `d3` animations and application of arbitrary easing functions. Alternatively, the component also provides a custom animation timeline that allows for more granular playback control.

This custom animation timeline may be connected to a `TimelineSlider`, which manages an interactive interface to control playback (Figure 4.2 bottom). This component is disabled until a transition is loaded into the `Scatterplot`.

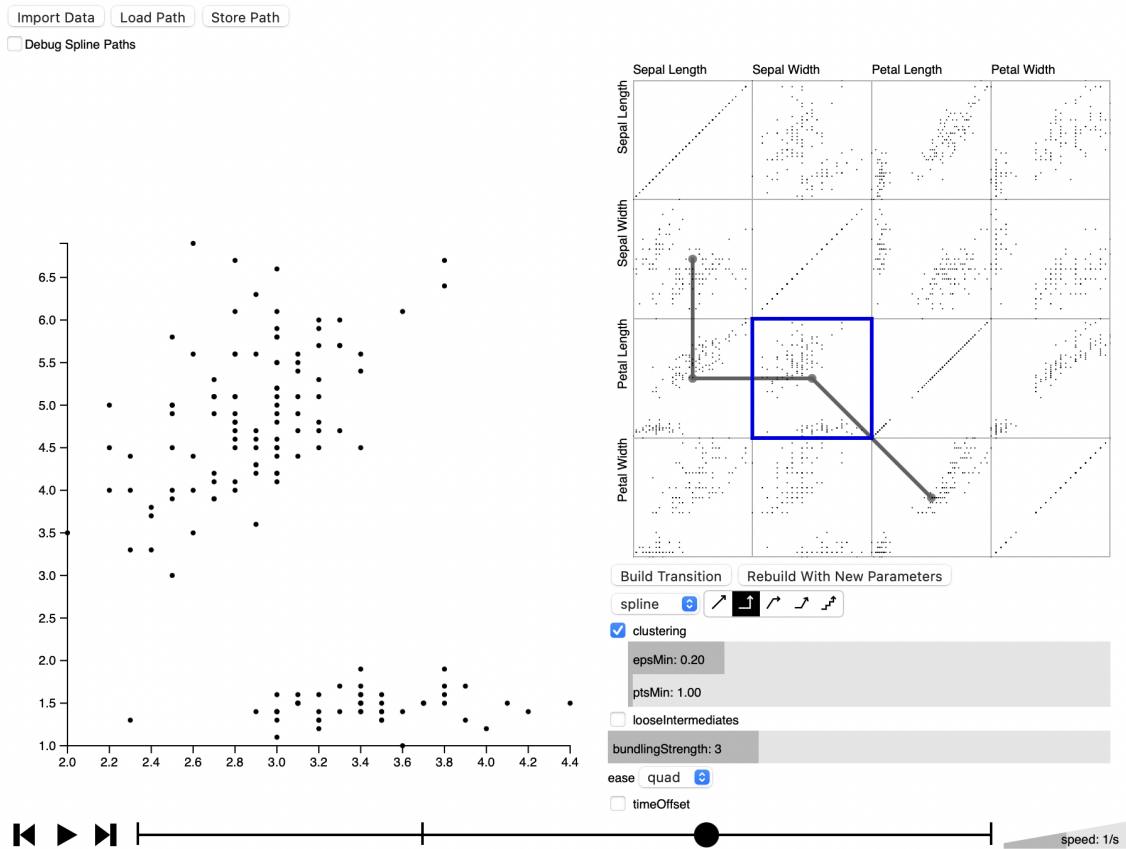


Figure 4.2: A test application with all UI components integrated.

For transitions with meaningful intermediary views, it draws vertical bars at their locations in time and automatically snaps the slider knob to the nearest view when close. Buttons are provided for controlling playback and skipping to the next or previous view. Holding the option/alternate key also allows backwards playback. Finally, a speed slider allows control of the animation speed.

To allow users to load a transition into the Scatterplot component, the ScatterplotMatrix component may be used to provide an interactive scatter plot matrix (Figure 4.2 right). This component also handles transition parameters and transition path transformations.

Clicking in the matrix causes a transition to be created and immediately start animating from the current view to the view under the cursor. For more complex transitions, users may click a button to enter the transition path builder mode. In this mode, longer transitions can be created by clicking cells in sequence. If a path transformation is active, the transformed path is shown as well (Figure 4.3).

Transitions created using the scatter plot matrix use the parameters set by the user below it. After picking a transition type and path transformation, the parameter definitions of that transition type are used to render a configuration interface (Figure 4.4).

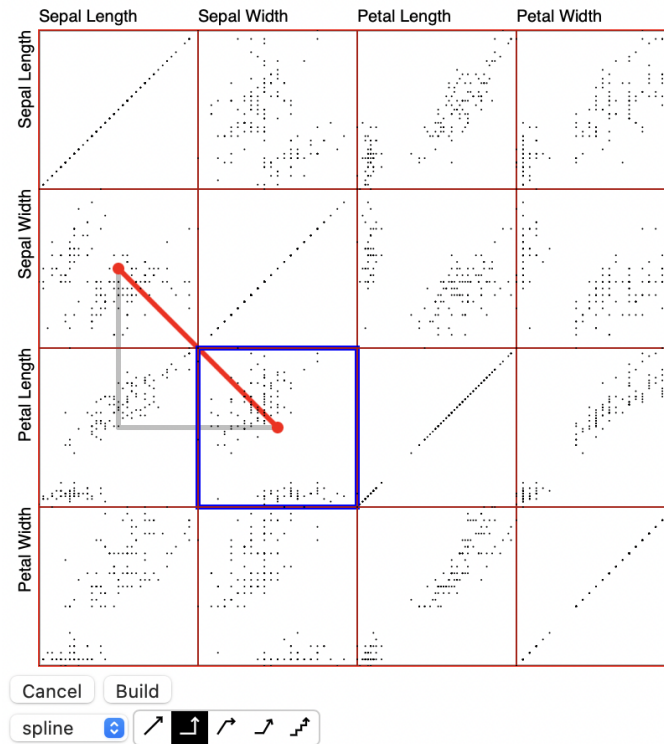


Figure 4.3: The transition path builder mode. The grid is highlighted in red and the user can click anywhere in the matrix to extend the path. A path transformation is selected in the controls below, so the transformed path is shown in gray. The blue outline indicates the current view.

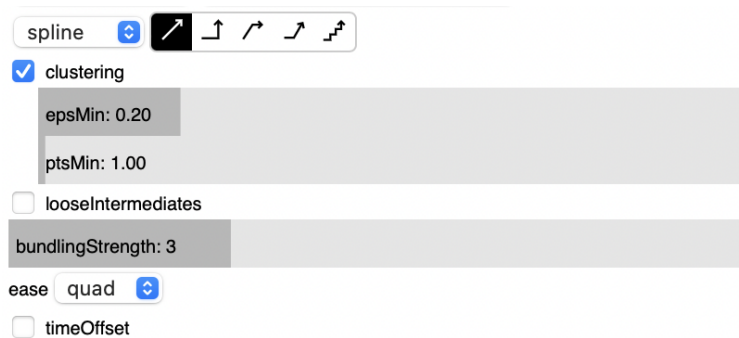


Figure 4.4: User interface for configuring the transition builder. In the top row, users can choose the transition type and path transformation. Below are the parameters for that particular transition type.

The screenshot shows a configuration dialog titled "Import Data" with the instruction "Specify how to import each data dimension." It contains five sections for different dimensions:

- Sepal Length:** Action: Read; Domain: 4.3 to 7.9; Log:
- Sepal Width:** Action: Read; Domain: 2 to 4.4; Log:
- Petal Length:** Action: Read; Domain: 1 to 6.9; Log:
- Petal Width:** Action: Read; Domain: 0.1 to 2.5; Log:
- Class:** (not a number); Action: Ignore

At the bottom, there are "Cancel" and "Import" buttons.

Figure 4.5: The FileIo component presents a configuration dialog when importing CSV datasets. Dimensions may be read or ignored. Non-numeric dimensions are always ignored.

Finally, a FileIo component can be used to load CSV datasets, and to load and store transition paths (Figure 4.2 top left, Figure 4.5). Transition paths are stored as simple CSV files. Two columns hold the names of the X and Y dimensions of every view in sequence.

4.3 Transition Types

4.3.1 Rotation

The rotation transition has several adjustable details in the implementation. During the transition, an orthographic projection matrix and a perspective projection matrix are linearly interpolated with a “perspective-ness” variable. The maximum of this value is controlled by the perspective parameter of the transition, with a range of $[0, 1]$, where 0 corresponds to the ROTORTHO transition, and 1 corresponds to the ROTPERSP transition.

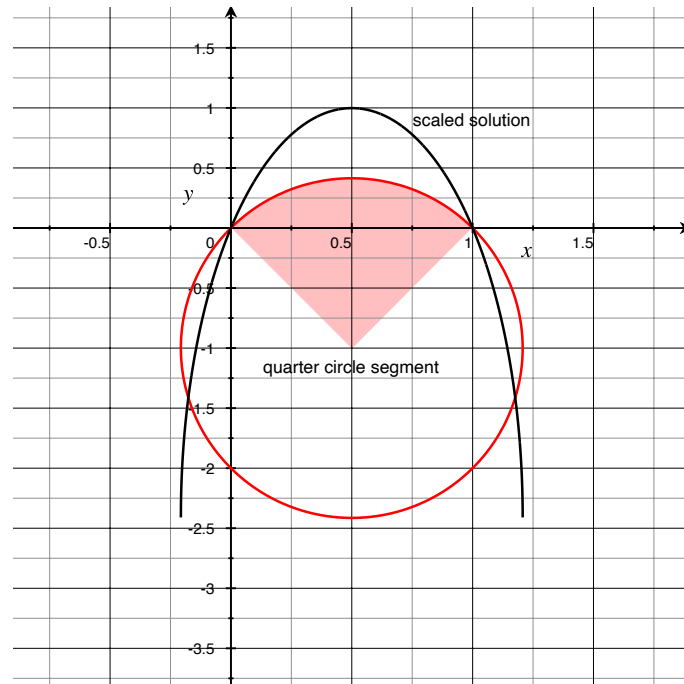


Figure 4.6: Circle segment used to interpolate the perspective value in the perspective rotation transition.

The `RotPersp` transition uses a circle segment to smoothly interpolate the perspective value between 0 and 1 with the transition. We chose approximately a quarter circle segment to match the rotation of the scatter plot, and such that the maximal perspective value is reached exactly in the middle of the transition. The $2 = (2x - 1)^2 + (y + 1)^2$ circle features such a circle segment above the $y = 0$ line ranging from $x \in [0, 1]$. We used a scaled solution for y to compute perspective $p = 2.41(\sqrt{2 - (2t - 1)^2} - 1)$, $t \in [0, 1]$ (Figure 4.6).

When using the `RotStaged` transition, however, we use a `zoomTime` parameter to control the amount of time taken up by the expansion and compression stages. The default value is 20% of the duration (for a total of 40%). To make the switch from one stage to the next less jarring, an animation easing function may also be applied internally to the timing of each stage to smooth the transition and effectively create a visual pause.

The field of view and camera distance of the perspective projection are also adjustable, since there is no single obvious location where the camera would be placed in 3D space.

4.3.2 Spline

The implementation of the spline transition is the most complex, as the clustering and the generation of paths is computationally nontrivial. These tasks are thus offloaded to a web worker and completed asynchronously.

The clustering itself is performed using the `fuzzy_dbscan` [Sch18] Rust library compiled to WebAssembly. The data points from the dataset and resulting cluster assignments are transferred over the language boundary using a packed binary format. While the library supports fuzzy clustering, we currently only pass in parameters for hard clustering and do not use fuzzy assignments.

Paths are then generated as outlined in section 3.2.3. The most computationally expensive part of path generation is the creation of lookup tables for arc length, taking up the majority of the runtime.

When the WebAssembly module is compiled with optimizations, the clustering and path generation is relatively instantaneous for small datasets, on a modern computer. For example, a transition using the `netperf` [War09] dataset (179 data points) with 9 views and clustering enabled takes about 150 ms to prepare on a recent laptop (Chromium on macOS 12, Apple M1 Max, 32 GB RAM).

4.4 API Usage Example

The following code creates a simple animation that transitions between two scatter plots of a 3D dataset using a `RotStaged` transition. We will assume that the D3 framework and the `d3-scattertrans` library have already been loaded.

First, we define our example dataset. This dataset has 3 dimensions: `x`, `y`, and `z`.

Listing 4.1 Definition of an example dataset.

```
const exampleData = [
  { x: 1, y: 2, z: 1 },
  { x: 1, y: 3, z: 2 },
  { x: 2, y: 4, z: 5 },
  { x: 3, y: 3, z: 2 },
  { x: 4, y: 2, z: 3 },
  { x: 5, y: 1, z: 2 },
];
```

Now, we must declare our data dimensions to define how they should be displayed. Here, we create a dimension for `x` with the domain `[1, 5]`. The string passed as the first argument must match the object key in the `exampleData`.

Listing 4.2 Creation of a dimension descriptor.

```
const xDomain = [1, 5];
const dimX = new d3.scatterTrans.Dimension('x', xDomain);
```

Dimensions can be created more conveniently by inferring the domain from the dataset.

4 Implementation

Listing 4.3 Inference of dimension domains from data.

```
const dimY = d3.scatterTrans.Dimension.fromData('y', exampleData);
const dimZ = d3.scatterTrans.Dimension.fromData('z', exampleData);
```

We now create an SVG element to hold the scatter plot, and append it to the HTML document.

Listing 4.4 Creation of an SVG container.

```
const size = 100; // pixels

const svg = d3.create('svg')
  .attr('width', size)
  .attr('height', size)
  .style('overflow', 'visible');
document.body.appendChild(svg.node());
```

Then, we create an instance of the Scatterplot component to easily display the scatter plot and make use of transitions. The Scatterplot is configured with the data, size on screen, and the currently visible dimensions.

Listing 4.5 Creation and configuration of a Scatterplot.

```
const scatterplot = d3.scatterTrans.scatterplot()
  .data(exampleData)
  .size(size, size)
  .view(dimX, dimY);
```

Using the Scatterplot, we then create SVG circles for each data point. Circles may be configured as needed. In this case, we color them blue.

Listing 4.6 The Scatterplot is used to create points.

```
const circlesContainer = svg.append('g');
const circles = scatterplot.createCircles(circlesContainer)
  .style('fill', 'blue');
```

Finally, we create a simple ROTSTAGED transition that swaps the Y dimension to dimZ. The Scatterplot component provides a convenient transition builder interface with functions toX(dim), toY(dim), and toView(x, y). Calling build() on the transition builder returns a Javascript Promise that resolves when the transition is ready.

Listing 4.7 The Scatterplot transition builder interface.

```
const transitionReady = scatterplot.transition(d3.scatterTrans.RotationTransition, {
  perspective: 1,
  staged: true
})
  .toY(dimZ)
  .build();
```

After the transition is ready, we can start playing the animation using the `d3-transition` component of the D3 framework. Here, we create a two-second animation with cubic easing.

Listing 4.8 A transition played using `d3-transition`.

```
transitionReady.then(() => {
  circles.transition()
    .duration(2000)
    .ease(d3.easeCubic)
    .call(selection => scatterplot.transitionCircles(selection));
});
```

Alternatively, we can use the custom animation timeline built into `Scatterplot`. The `Scatterplot` will automatically animate the last set of circles it created (in this case, the blue circles created earlier).

Listing 4.9 A transition played using the `Scatterplot` animation timeline.

```
transitionReady.then(() => {
  scatterplot.speed(0.5).playing(true);
});
```

5 User Study

We conducted a user study to evaluate the relative difficulty of using the various transition methods. We investigated different transition types for different purposes: the rotation transition was given closer attention for visualization of individual points, and the spline transitions was used primarily for clusters. All transitions were also evaluated for appeal in questionnaires.

The study was conducted online via LimeSurvey with custom Javascript and d3-scattertrans. It was designed for a duration of around 60 minutes. Participants were recruited through Amazon Mechanical Turk.

Our hypotheses are as follows:

(H1.1) Users can follow individual points with fewer errors using splines rather than rotation.

We suspect this because splines preserve the trajectory of motion, while rotation does not.

(H1.2) Users can follow individual points with fewer errors using ROTPERSP instead of ROTORTHO.

The perspective rotation provides more perceived spatial separation between the points than orthographic rotation.

(H1.3) Users can follow individual points with fewer errors using ROTSTAGED instead of ROTPERSP.

This would agree with [HR07], which states that staged transitions generally perform better.

(H1.4) Users can follow individual points with fewer errors if they are surrounded by fewer points during the transition.

Points that are surrounded by other points or overlap other points are likely to be difficult to follow.

(H2.1) Users can follow cluster movement with fewer errors using splines rather than ROTSTAGED.

We assume this due to the alignment of motion trajectories within clusters when using splines.

(H2.2) Users can follow cluster movement with fewer errors using SPLINEBUNDLED instead of SPLINE.

Bundling the points provides momentary visual separation of clusters that makes it easier to tell them apart.

(H2.3) Users can follow cluster movement with fewer errors using SPLINETIMEOFFSET instead of SPLINE.

Time offsets provide temporal separation of clusters that makes it easier to tell them apart.

(H2.4) Users can follow cluster movement with fewer errors using SPLINEBUNDLED instead of SPLINETIMEOFFSET.



Figure 5.1: Users are prompted to follow a point through the transition. After the transition, users can click and drag to place a red marker.

The time-offset transition still has spans time during which multiple clusters are moving simultaneously, presumably creating less strong separation than the bundling method.

5.1 Procedure

After asking users for basic demographic information, we introduce users to animated transitions between scatter plots. Following the explanation, we ask how generally familiar they are with scatter plots before continuing.

The main tasks are divided into four sections.

In the first section, participants are asked to follow a random point in the netperf [War09] dataset through a transition (Figure 5.1). We compared ROTORTHO, ROTPERSP, ROTSTAGED, and SPLINE with 10 trials each. Here, we aimed to answer (H1.1), (H1.2), (H1.3), and (H1.4).

The task shows a randomly chosen view of the data and highlights a point. After the user presses a button, the transition alternately swaps the horizontal and vertical dimension 3 times with a random transition path. Each transition takes 2 seconds, apart from the ROTSTAGED transition. We

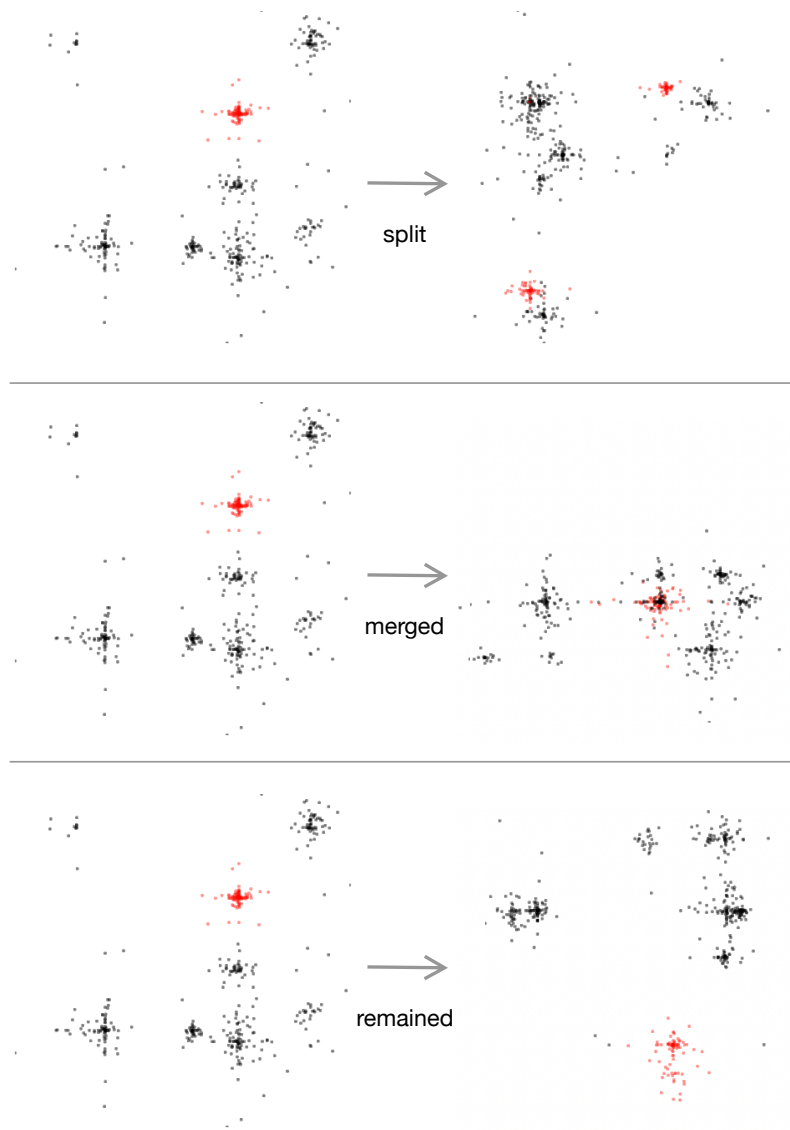


Figure 5.2: Possible outcomes when tracking a cluster.

increased the duration of ROTSTAGED to around 4 seconds such that the actual rotation stage takes the same amount of time as other transitions. Then, the user clicks a location in the final scatter plot to indicate where they believe the point ended up. The location circle is deliberately larger than a point to suggest a certain degree of error tolerance. From this location and the actual point location, we can obtain the user error distance. The user then presses a button to submit their answer and start the next trial. The user completes 10 data trials, and one validation trial not included in the analysis (see Section 5.1.1). After each set of trials, the user answers a questionnaire. Using a 5-point Likert scale from “Strongly Disagree” (1) to “Strongly Agree” (5), the user evaluates the speed of the animation, their impression of how well points can be tracked, and whether they would like to use this animation. Users are also given the opportunity to add any further comments in a text box.

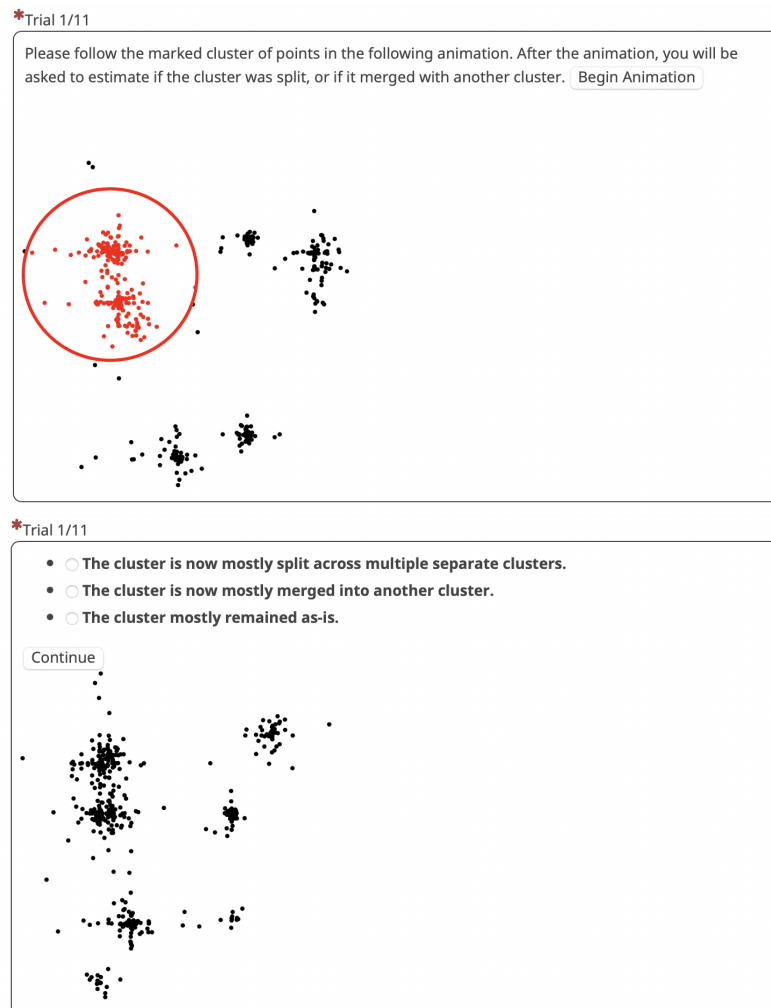



Figure 5.3: Users are prompted to follow a cluster through the transition. After the transition, users are asked select the new state of the cluster.

The second section asks participants to state what happens to a cluster after a transition. Possible outcomes include that a cluster was split into multiple clusters, that it merged with another cluster, or that it remained as-is (Figure 5.2). Here, we aimed to address (H2.1), (H2.2), (H2.3), and (H2.4). In this section, we compare `ROTSTAGED`, `SPLINE`, `SPLINEBUNDLED` (strength 3), and `SPLINETIMEOFFSET` (Figure 5.3). For this task, we used synthetic data with obvious clusters. We hand-picked the transition tasks, as they were difficult to generate randomly.

The task shows one of 10 cluster transitions and highlights a cluster of points. After the user presses a button, the transition swaps both the horizontal and vertical dimension 2 times with a random transition path. For the rotation transition, which does not support this kind of path, we apply the Manhattan path transformation first. Again, each transition takes 2 seconds. After the transition, the user must decide whether the original cluster is now split across multiple clusters, merged with another cluster, or whether it remained a single cluster. After each set of 10 trials and one validation

Please evaluate the following animation.



	Disagree	Disagree	Neutral	Agree	Agree
I think the animation speed is easy to follow.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I think the trajectories of individual points are easy to follow.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I think the trajectories of clusters are easy to follow.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I think that I would like to use this animation frequently.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Figure 5.4: In the third section, users are presented with a transition and a questionnaire.

trial not included in the analysis, users again answer a questionnaire. This questionnaire is similar to that of point trials, except the user is now asked about their impression of how well clusters can be tracked.

The third section is a comparison of `STRAIGHT` and `ROTORTHO`, as they are visually very similar (Figure 5.4). This section consists entirely of two interactive transition examples and the same questionnaires from the previous questions. We prompt the user to select random points and clusters and try tracking through the animation to inform their judgement.

The final section aims to evaluate how long transitions can become before points cannot be tracked anymore, for `SPLINE`, `ROTPERSP`, and `ROTSTAGED`. We use 20 trials with the same setup as the first section. We also employ staircasing to increase the transition path length by 2 views for every 2 successive “correct” answers, and decrease it by one for every error. The path length is initialized to 3 views. An answer is considered correct if its error distance is less than 10% of the size of the scatter plot.

5.1.1 Validation

During recruitment, Mechanical Turk was set to only accept users with a HIT (human intelligence task) approval rate of 98% or higher and an approval count of 10,000 or higher. To validate submissions and filter bad data, we also added additional validation tasks to the first two sections. In

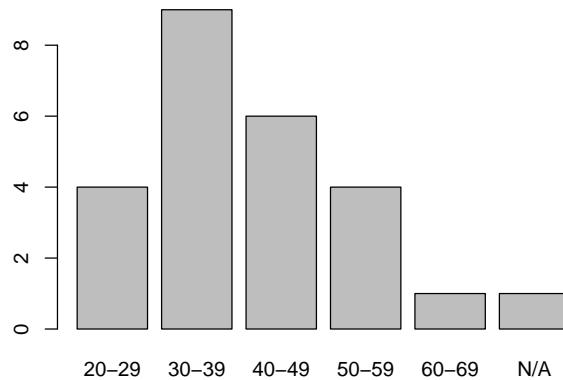


Figure 5.5: Age distribution of study participants.

the first section, we use an introductory example both as training and for validation. Each of the four transition types also feature an additional eleventh trial with an easy transition. We consider such a trial valid if the user error distance is at most 25% the size of the scatter plot. This corresponds to a circle approximately 20% of the area of the scatter plot. In the cluster tracking tasks, we also add an eleventh trial with an easy transition. We consider these trials valid if the user answers correctly. In these validation tasks, we required a soft bound of at least 50% correct responses to pass validation. Due to the nature of the last section, we did not add validation trials there. Finally, users are also required to take at least 23 minutes to submit their responses, as it is impossible watch all animations in less time.

5.2 Results

We received 25 responses to the study, all of which passed validation. There were 10 female, 13 male, and 2 unspecified participants. The mean age was around 40 (Figure 5.5).

Users were asked to rate their familiarity with scatter plots (Figure 5.6). For the question “I am familiar with scatter plots” users answered with a median of 4 (Agree). “I work with scatter plots frequently” received a median answer of 2 (Disagree).

5.2.1 Hypothesis-based Analysis of Point Tracking

Using the data from the first section of the study, we can compare how well each transition type performs for tracking points through a transition (Figure 5.7). A Kruskal-Wallis test showed that the type of transition significantly affects user accuracy, $H(3) = 12.061$, $p = 0.0072$. ROTORTHO (Mdn = 0.12) achieved lower error distances than SPLINE (Mdn = 0.13), ROTPERSP (Mdn = 0.14), and ROTSTAGED (Mdn = 0.16). Post-hoc Mann-Whitney tests were used to compare all pairs of groups.

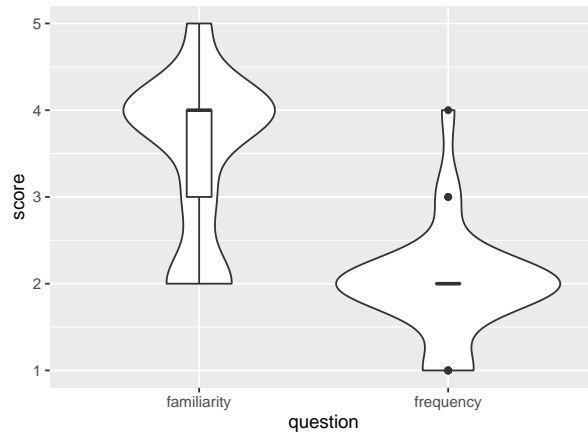


Figure 5.6: User familiarity with scatter plots and how frequently they work with scatter plots. Scores range from 1 (Strongly Disagree), 2 (Disagree), 3 (Neutral), 4 (Agree), to 5 (Strongly Agree).

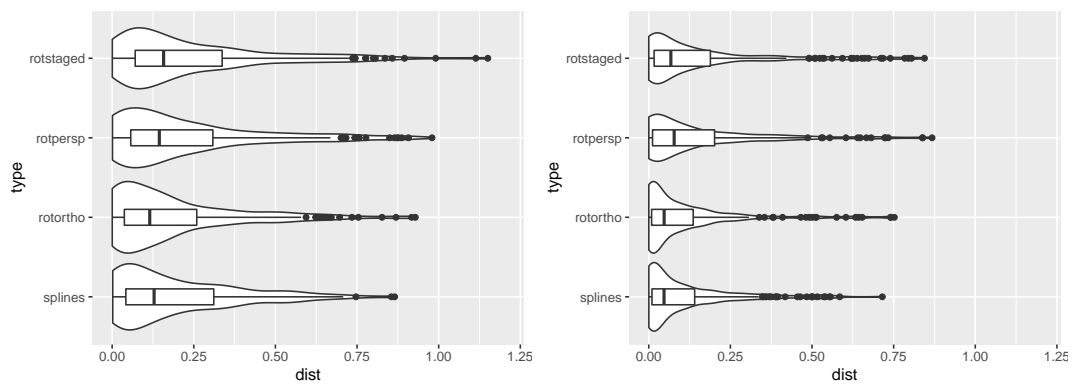


Figure 5.7: User error distances. The left plot shows the euclidean distance, while the right plot shows the minimum of the X and Y distance. Since the transitions were swapping one dimension at a time, we can use the right plot to estimate how accurate users were one transition before the end.

- **SPLINE** (Mdn = 0.13) performed better than **ROTSTAGED** (Mdn = 0.16). A Mann-Whitney test indicated that this difference was statistically significant, $U(N = 250) = 27526$, $|z| = 2.305$, $p = 0.021$. This partially agrees with hypothesis (H1.1), though we cannot draw any conclusions about any other type of rotation, as no other comparison between **SPLINE** and rotation was significant.
- **ROTORTHO** (Mdn = 0.12) performed better than **ROTPERSP** (Mdn = 0.14). A Mann-Whitney test indicated that this difference was statistically significant, $U(N = 250) = 27608$, $|z| = 2.254$, $p = 0.024$. This presents evidence contrary to hypothesis (H1.2).
- **ROTORTHO** (Mdn = 0.12) performed better than **ROTSTAGED** (Mdn = 0.16). A Mann-Whitney test indicated that this difference was statistically significant, $U(N = 250) = 26054$, $|z| = 3.216$, $p = 0.0013$. This partially disproves the combination of hypotheses (H1.2) and (H1.3), but we cannot answer either hypothesis individually.

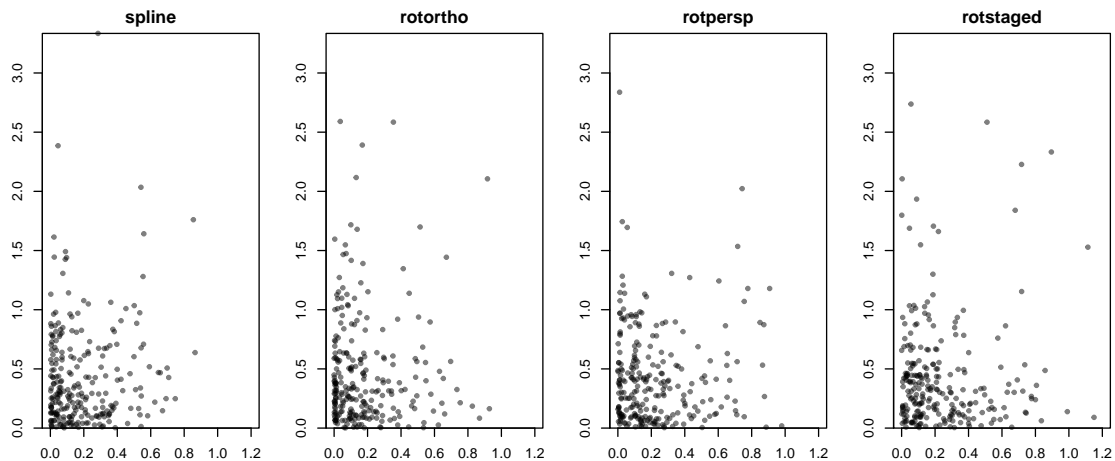


Figure 5.8: The horizontal axis shows euclidean user error distances. The vertical axis shows the maximal density of points in the vicinity of the selected point over all views of the transition.

- None of the remaining tests were significant:

SPLINE ROTORTHO $U(N = 250) = 32829, |z| = 0.977, p = 0.33$

SPLINE ROTPERSP $U(N = 250) = 29243, |z| = 1.242, p = 0.21$

ROTPERSP ROTSTAGED $U(N = 250) = 29751, |z| = 0.928, p = 0.35$

A Holm-adjusted pairwise Wilcoxon rank sum test gave fewer significant results. The only significant difference was between ROTORTHO and ROTSTAGED ($p = 0.0078$). From this, the most conclusive result we found is that ROTSTAGED performs significantly worse than any other transition type for point tracking.

Our main suspected indicator for task difficulty was the ambiguity caused by overlapping points, making it more difficult for users to track one point in particular. To estimate the density of points around a particular point, we used a gaussian kernel density estimator with a bandwidth 5% of the size of the scatter plot on the relative euclidean distances of all other points. We used a maximum of this metric over all views in a transition to measure the difficulty of a point tracking task (Figure 5.8). However, we found no significant correlations with user error distances:

- For SPLINE, a Pearson correlation test showed that the two were not significantly related, $r = 0.1, p = 0.22, N = 250$.
- For ROTORTHO, a Pearson correlation test showed that the two were not significantly related, $r = -0.02, p = 0.69, N = 250$.
- For ROTPERSP, a Pearson correlation test showed that the two were not significantly related, $r = -0.02, p = 0.76, N = 250$.
- For ROTSTAGED, a Pearson correlation test showed that the two were not significantly related, $r = 0.002, p = 0.97, N = 250$.

Hence, we cannot confirm hypothesis (H1.4).

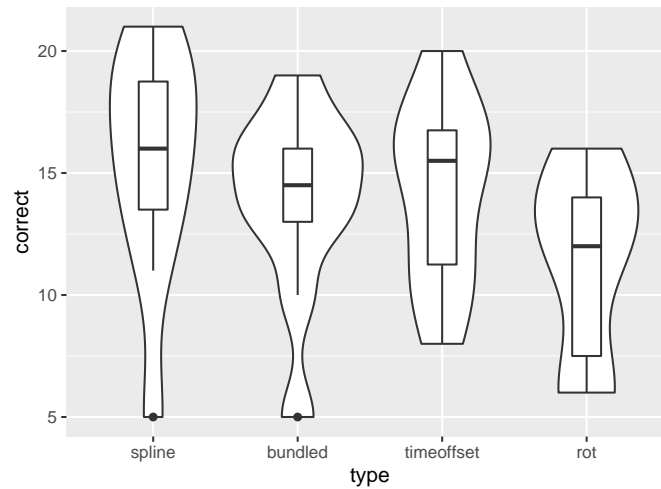


Figure 5.9: Number of participants who answered correctly for each type of cluster transition.

5.2.2 Hypothesis-based Analysis of Cluster Tracking

In the cluster tracking tasks, a Kruskal-Wallis test showed that the type of transition does not significantly affect the number of correct responses, $H(3) = 5.270$, $p = 0.15$. Hence, we can reach no conclusions about hypotheses (H2.1), (H2.2), (H2.3), (H2.4). However, there are still minor differences in the distributions (Figure 5.9). Out of 25 participants, a median of 16 participants answered correctly for `SPLINE`, and a median of 15.5 answered correctly for `SPLINE``TIMEOFFSET`. This is slightly better performance than `SPLINE``BUNDLED` (Mdn = 14.5) and `ROT``STAGED` (Mdn = 12). These deviations also agree with the questionnaire results, which do have statistically significant differences (see Section 5.2.4).

5.2.3 Explorative Analysis of Point Tracking

We examined the relation between transition length and error distance, using data from the final section of the study (Figure 5.10). Using combined data from all transition types, a Pearson correlation test showed that the two were significantly related, $r = 0.27$, $p \ll 0.001$, $N = 1500$.

- One transition (Mdn = 0.018) performed better than two transitions (Mdn = 0.084). A Mann-Whitney test indicated that this difference was statistically significant, $U(N_{1 \text{ tr.}} = 487, N_{2 \text{ tr.}} = 414) = 65181$, $|z| = 9.152$, $p \ll 0.001$.
- Two transitions (Mdn = 0.084) performed better than three transitions (Mdn = 0.115). A Mann-Whitney test indicated that this difference was statistically significant, $U(N_{2 \text{ tr.}} = 414, N_{3 \text{ tr.}} = 395) = 71746$, $|z| = 3.015$, $p = 0.003$.
- Three transitions (Mdn = 0.115) performed marginally better than four transitions (Mdn = 0.121). A Mann-Whitney test indicated that this difference was not statistically significant, $U(N_{3 \text{ tr.}} = 395, N_{4 \text{ tr.}} = 204) = 38956$, $|z| = 0.664$, $p = 0.51$.

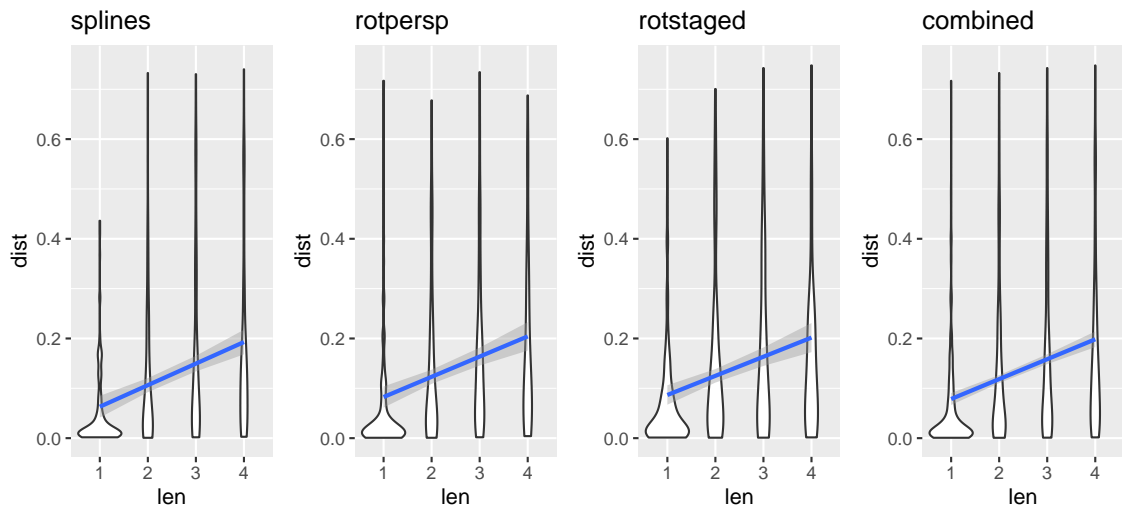


Figure 5.10: Euclidean user error distances for different transition lengths. Correlation lines drawn in blue.

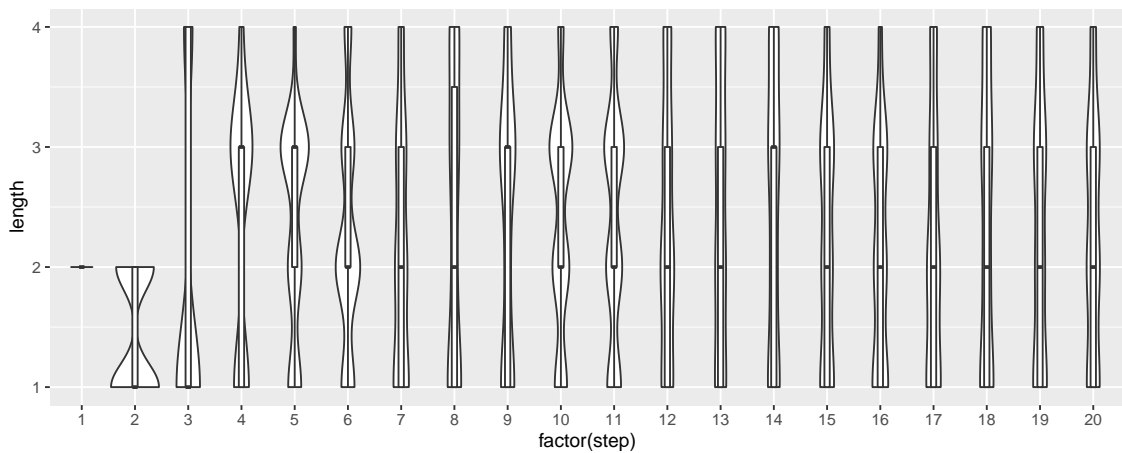


Figure 5.11: Distribution of transition lengths at each staircasing step (combined data from all transition types).

As would be expected, tracking points through longer transitions is more difficult, causing error distances to increase. The staircasing converged to a median of 2 transitions after 20 steps (Figure 5.11), suggesting that this would be the longest transition that users can track points through comfortably.

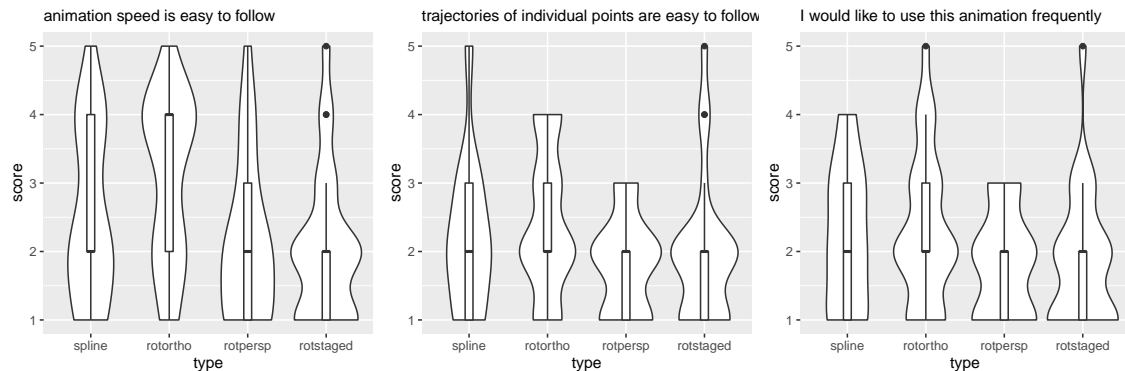


Figure 5.12: Questionnaire answers for point tracking. Scores range from 1 (Strongly Disagree), 2 (Disagree), 3 (Neutral), 4 (Agree), to 5 (Strongly Agree).

5.2.4 Explorative Analysis of Questionnaire Results

Point Tracking

Figure 5.12 shows the distribution of questionnaire answers for each transition type. In the questionnaire, users rated ROTORTHO as having easier to follow animation speed than other animation types. A Kruskal-Wallis test showed that the type of transition significantly affects this score, $H(3) = 10.189$, $p = 0.017$. Post-hoc Mann-Whitney tests were used to compare all groups.

- ROTORTHO (Mdn = 4 (Agree)) performed better than ROTPERSP (Mdn = 2 (Disagree)). A Mann-Whitney test indicated that this difference was statistically significant, $U(N = 25) = 441.5$, $|z| = 2.591$, $p = 0.0096$.
- ROTORTHO (Mdn = 4 (Agree)) also performed better than ROTSTAGED (Mdn = 2 (Disagree)). A Mann-Whitney test indicated that this difference was statistically significant, $U(N = 25) = 453$, $|z| = 2.825$, $p = 0.0047$.
- No other Mann-Whitney tests had statistically significant results:

SPLINE	ROTORTHO	$U(N = 25) = 240$, $ z = 1.467$, $p = 0.14$
SPLINE	ROTPERSP	$U(N = 25) = 369.5$, $ z = 1.144$, $p = 0.25$
SPLINE	ROTSTAGED	$U(N = 25) = 379.5$, $ z = 1.351$, $p = 0.18$
ROTPERSP	ROTSTAGED	$U(N = 25) = 321.5$, $ z = 0.174$, $p = 0.86$

Users rated the statement that trajectories of individual points are easy to follow with a median of 2 (Disagree) for all transition types. A Kruskal-Wallis test showed that the type of transition did not significantly affect this score, $H(3) = 5.133$, $p = 0.16$. Finally, when asked whether they would like to use this animation frequently, users again rated all transitions with a median of 2 (Disagree). A Kruskal-Wallis test again showed that the type of transition did not significantly affect this score, $H(3) = 4.631$, $p = 0.20$.

In the free-form answers, users consistently highlighted visually merging points as a point of difficulty (13 mentions). Participant 10 emphasized the reprojection in intermediary views of rotation transitions as another point of difficulty. Two participants highlighted ROTPERSP as being

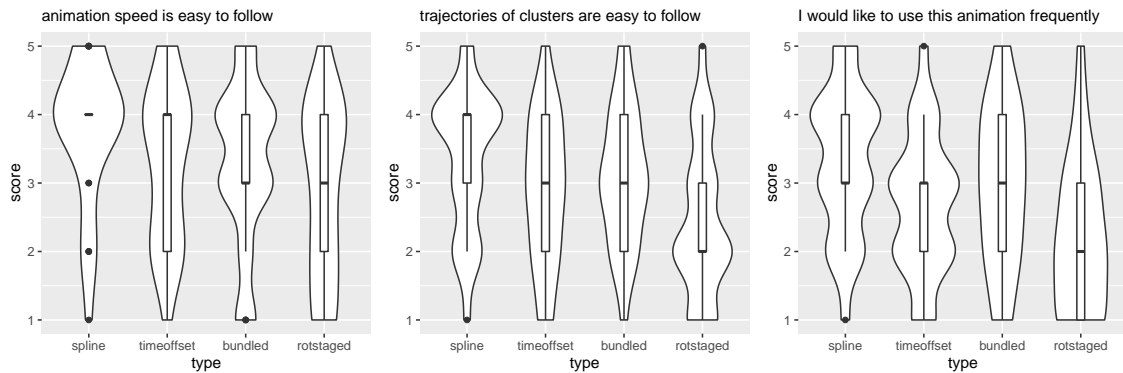


Figure 5.13: Questionnaire answers for cluster tracking. Scores range from 1 (Strongly Disagree), 2 (Disagree), 3 (Neutral), 4 (Agree), to 5 (Strongly Agree).

smooth but still confusing. Two other participants noted that the ROTSTAGED was less smooth and more random than other transitions, likely due to its three stages each a different type of animation.

Cluster Tracking

In the questionnaire, users rated SPLINE and SPLINETIMEOFFSET (both Mdn = 4 (Agree)) as having easier to follow animation speed than other animation types (Figure 5.13). A Kruskal-Wallis test showed that the type of transition not quite significantly affects this score, $H(3) = 7.047$, $p = 0.07$. Post-hoc Mann-Whitney tests were used to compare all groups.

- SPLINE (Mdn = 4 (Agree)) performed better than ROTSTAGED (Mdn = 3 (Neutral)). A Mann-Whitney test indicated that this difference was statistically significant, $U(N = 25) = 437.5$, $|z| = 2.574$, $p = 0.01$.
- No other Mann-Whitney tests had statistically significant results:

SPLINE	SPLINETIMEOFFSET	$U(N = 25) = 389.5$, $ z = 1.586$, $p = 0.11$
SPLINE	SPLINEBUNDLED	$U(N = 25) = 394$, $ z = 1.666$, $p = 0.10$
SPLINETIMEOFFSET	SPLINEBUNDLED	$U(N = 25) = 311$, $ z = 0.020$, $p = 0.98$
SPLINETIMEOFFSET	ROTSTAGED	$U(N = 25) = 366.5$, $ z = 1.084$, $p = 0.28$
SPLINEBUNDLED	ROTSTAGED	$U(N = 25) = 363$, $ z = 1.009$, $p = 0.31$

Unlike with points, users found differences in how easy trajectories of clusters are to follow. A Kruskal-Wallis test showed that the type of transition significantly affects this score, $H(3) = 9.440$, $p = 0.024$. Post-hoc Mann-Whitney tests were used to compare all groups.

- SPLINE (Mdn = 4 (Agree)) received better scores than ROTSTAGED (Mdn = 2 (Disagree)). A Mann-Whitney test indicated that this difference was statistically significant, $U(N = 25) = 453$, $|z| = 2.816$, $p = 0.005$.
- SPLINEBUNDLED (Mdn = 3 (Neutral)) also performed better than ROTSTAGED (Mdn = 2 (Disagree)). A Mann-Whitney test indicated that this difference was statistically significant, $U(N = 25) = 418.5$, $|z| = 2.122$, $p = 0.034$.

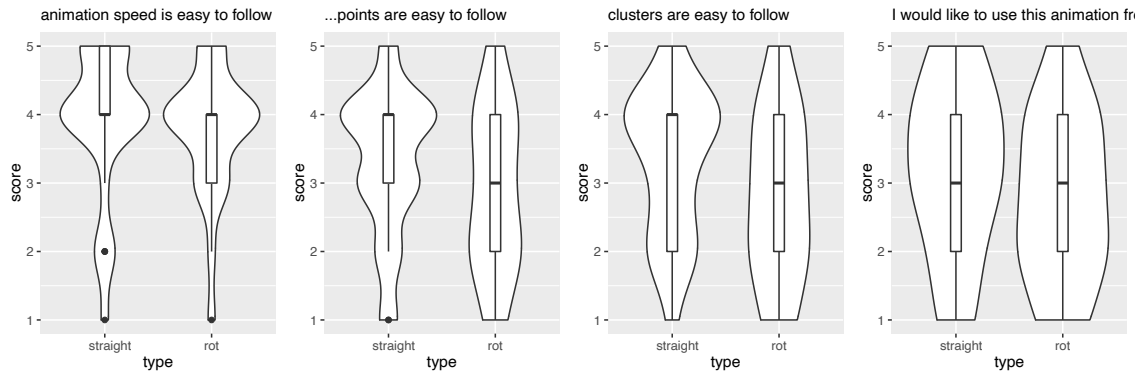


Figure 5.14: Questionnaire answers from the third section of the study. Scores range from 1 (Strongly Disagree), 2 (Disagree), 3 (Neutral), 4 (Agree), to 5 (Strongly Agree).

- No other Mann-Whitney tests had statistically significant results:

<code>SPLINE</code>	<code>SPLINETIMEOFFSET</code>	$U(N = 25) = 391.5, z = 1.591, p = 0.11$
<code>SPLINE</code>	<code>SPLINEBUNDLED</code>	$U(N = 25) = 368.5, z = 1.125, p = 0.26$
<code>SPLINETIMEOFFSET</code>	<code>SPLINEBUNDLED</code>	$U(N = 25) = 285.5, z = 0.536, p = 0.59$
<code>SPLINETIMEOFFSET</code>	<code>ROTSTAGED</code>	$U(N = 25) = 386, z = 1.465, p = 0.14$

Users also expressed differing opinions about whether they would like to use the animation frequently. A Kruskal-Wallis test showed that the type of transition significantly affects this score, $H(3) = 12.688, p = 0.005$.

- `SPLINE` performed slightly better than `SPLINETIMEOFFSET` (both Mdn = 3 (Agree)). A Mann-Whitney test indicated that this difference was statistically significant, $U(N = 25) = 416.5, |z| = 2.081, p = 0.037$.
- `SPLINE` and `SPLINEBUNDLED` (both Mdn = 3 (Agree)) also performed better than `ROTSTAGED` (Mdn = 2 (Disagree)). Mann-Whitney tests indicated that these differences were statistically significant, (`SPLINE`) $U(N = 25) = 477.5, |z| = 3.278, p = 0.001$, (`SPLINEBUNDLED`) $U(N = 25) = 427, |z| = 2.276, p = 0.023$.
- No other Mann-Whitney tests had statistically significant results:

<code>SPLINE</code>	<code>SPLINEBUNDLED</code>	$U(N = 25) = 366.5, z = 1.07, p = 0.28$
<code>SPLINETIMEOFFSET</code>	<code>SPLINEBUNDLED</code>	$U(N = 25) = 267, z = 0.903, p = 0.37$
<code>SPLINETIMEOFFSET</code>	<code>ROTSTAGED</code>	$U(N = 25) = 395.5, z = 1.662, p = 0.10$

In the free-form answers, three users noted the simplicity and understandability of the spline transitions as a positive point. Users did not agree with each other on specific spline types. The rotation transition was mostly described negatively (8 answers) as opposed to positively (2 answers).

Straight Transition and Orthographic Rotation

In the comparison of straight transition and orthographic rotation, we found that users rated the two rather similarly (Figure 5.14). Mann-Whitney tests indicated that there were no statistically significant differences regarding

- animation speed, $U(N = 25) = 352.5, |z| = 0.841, p = 0.40,$
- point tracking, $U(N = 25) = 381, |z| = 1.370, p = 0.17,$
- cluster tracking, $U(N = 25) = 391, |z| = 1.572, p = 0.12,$
- or whether users would like to use it frequently, $U(N = 25) = 370, |z| = 1.135, p = 0.26.$

Though, the results did skew more favorably towards the straight transition. In this section's free-form answers, two participants again noted the point of difficulty caused by merging points. In different answers, two participants expressed a preference for the straight trajectories in STRAIGHT as opposed to the rotation, while two other participants noted the opposite.

5.3 Discussion

From the user study, we have found that ROTSTAGED is singled out as the least suitable type of transition for points. Though not statistically significant, the best performing transition type, by median, was ROTORTHO. This is at least a little surprising, as increasing the complexity of the animation seemingly worsened its performance, contrary to our hypotheses. Further, we found users can accurately track points through approximately two transitions, after which performance will wane.

In the case of clusters, we did not find a significant result regarding transition types from the objective data, but users rated ROTSTAGED less favorably than other transition types. This may also be because the transition is longer due to the Manhattan path transformation. The best performing transition type, by median, was SPLINE, but this was also not a statistically significant result.

We found no significant differences between STRAIGHT and ROTORTHO according to questionnaires. The slight bias towards STRAIGHT may be due to the rotation transition's tendency to escape the bounding box of the scatter plot as the cube rotates, since its diagonal is longer than its side length.

Generally, users tended to prefer or perform well with transitions that were relatively simple, such as STRAIGHT, ROTORTHO, and SPLINE.

6 Conclusion and Outlook

In this work, we have presented a model of 2D animated scatter plot transitions by example. We have also examined how different types of transitions compare in a user study. We found relatively minor but statistically significant differences in suitability for a particular task between transition types.

We started with a theoretical concept of scatter plot transitions and introduced three types of transitions: the straight transition, which performs simple linear interpolation, rotation transitions, which are based on rotation in an additional spatial dimension, and spline transitions, which make use of smooth Bézier curves. We used this model to develop a TypeScript implementation of these concepts and accompanying interactive user interface components.

In a user study, we examined how effective each type of transition was for different tasks. We found that `ROTSTAGED` performed the worst, with no significant results between other transition types. We also found that transition length strongly correlates with task difficulty, and that two view changes are the median for a comfortable transition length when tracking points.

In conclusion, tracking points and clusters across scatter plots of one dataset remains a difficult task, and we did not find many differences between types of transitions. Creating more complex transitions using e.g. bundling or time offsets seemingly had little effect on the results. This seems to suggest that using more complex animations for these kinds of scatter plot transitions may only have diminishing returns.

Outlook

While we only examined animation on its own, it may be practical to combine animation with other identifying features to achieve potentially better performance than is presented here. Points may be easier to keep track of if marked with colors, shapes, or other means of distinguishing them. Another avenue of improvement may be the addition of more interactivity to the animations, as the animations presented here only have time as a freely adjustable variable.

The `d3-scattertrans` library and its conceptual model, while extensible, remains relatively rudimentary. The library itself may benefit from better integration with the `d3` ecosystem. The model also does not add any constraints on animations themselves, leaving them as black boxes that output point locations. It may be of interest to create a model that explicitly incorporates principles of animation and has a more transparent view of the animations.

In the user study, we did not specifically recruit users from a demographic already very familiar with scatter plots. Especially given the relevance of scatter plots in more niche applications, such a demographic is a likely use case for this kind of system. We do not know what kind of effect familiarity with scatter plots in general, or familiarity with this specific system of transitions would

have on their performance. If users are given more time to understand multidimensional datasets in scatter plots and to familiarize themselves with each specific type of transition, it may be possible that they would achieve different results.

Bibliography

- [Asi85] D. Asimov. “The Grand Tour: A Tool for Viewing Multidimensional Data”. In: *SIAM Journal on Scientific and Statistical Computing* 6.1 (1985), pp. 128–143. DOI: [10.1137/0906011](https://doi.org/10.1137/0906011) (cit. on p. 14).
- [Bos21] M. Bostock. *D3.js - Data-Driven Documents*. 2021. URL: <https://d3js.org> (cit. on p. 23).
- [EDF08] N. Elmqvist, P. Dragicevic, J.-D. Fekete. “Rolling the Dice: Multidimensional Visual Exploration using Scatterplot Matrix Navigation”. In: *IEEE Transactions on Visualization and Computer Graphics* 14.6 (2008), pp. 1539–1148. DOI: [10.1109/TVCG.2008.153](https://doi.org/10.1109/TVCG.2008.153) (cit. on pp. 14, 17).
- [EGS+13] J. W. Emerson, W. A. Green, B. Schloerke, J. Crowley, D. Cook, H. Hofmann, H. Wickham. “The Generalized Pairs Plot”. In: *Journal of Computational and Graphical Statistics* 22.1 (2013), pp. 79–91. DOI: [10.1080/10618600.2012.694762](https://doi.org/10.1080/10618600.2012.694762) (cit. on p. 13).
- [EK SX96] M. Ester, H.-P. Kriegel, J. Sander, X. Xu. “A density-based algorithm for discovering clusters in large spatial databases with noise”. In: AAAI Press, 1996, pp. 226–231 (cit. on p. 20).
- [Fis36] R. A. Fisher. “The Use of Multiple Measurements in Taxonomic Problems”. In: *Annals of Eugenics* 7.2 (1936), pp. 179–188. DOI: [10.1111/j.1469-1809.1936.tb02137.x](https://doi.org/10.1111/j.1469-1809.1936.tb02137.x) (cit. on p. 15).
- [HR07] J. Heer, G. Robertson. “Animated Transitions in Statistical Data Graphics”. In: *IEEE Transactions on Visualization and Computer Graphics* 13.6 (2007), pp. 1240–1247. DOI: [10.1109/TVCG.2007.70539](https://doi.org/10.1109/TVCG.2007.70539) (cit. on pp. 13, 33).
- [Ins85] A. Inselberg. “The plane with parallel coordinates”. In: *The Visual Computer* 1 (1985), pp. 69–91. DOI: [10.1007/BF01898350](https://doi.org/10.1007/BF01898350) (cit. on p. 13).
- [RSLW20] N. Rodrigues, C. Schulz, A. Lhuillier, D. Weiskopf. “Cluster-Flow Parallel Coordinates: Tracing Clusters Across Subspaces”. In: *Proceedings of Graphics Interface 2020*. GI 2020. University of Toronto: Canadian Human-Computer Communications Society / Société canadienne du dialogue humain-machine, 2020, pp. 382–392. ISBN: 978-0-9947868-5-2. DOI: [10.20380/GI2020.38](https://doi.org/10.20380/GI2020.38) (cit. on p. 13).
- [Sch18] C. Schulz. *fuzzy_dbscan*. 2018. URL: https://github.com/schulzch/fuzzy_dbscan (cit. on p. 29).
- [SW12] H. Sanftmann, D. Weiskopf. “3D Scatterplot Navigation”. In: *IEEE Transactions on Visualization and Computer Graphics* 18.11 (2012), pp. 1969–1978. DOI: [10.1109/TVCG.2012.35](https://doi.org/10.1109/TVCG.2012.35) (cit. on p. 14).
- [War09] M. O. Ward. *XmdvTool Home Page: Downloads: DataSets*. 2009. URL: <https://davis.wpi.edu/xmdv/datasets.shtml> (cit. on pp. 29, 34).

[WO11] A. Waddell, R. Oldford. “RnavGraph: A visualization tool for navigating through high-dimensional data”. In: Jan. 2011, pp. 3294–3303 (cit. on p. 14).

All links were last followed on April 25, 2022.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

place, date, signature